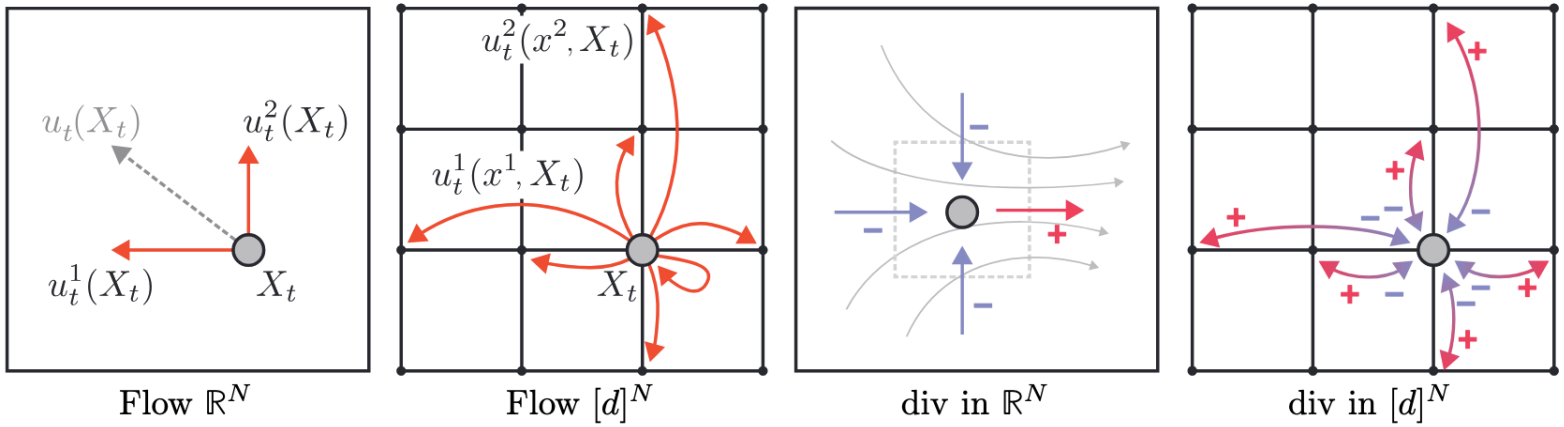
Discrete Flow Matching

[NeurIPS 2024]

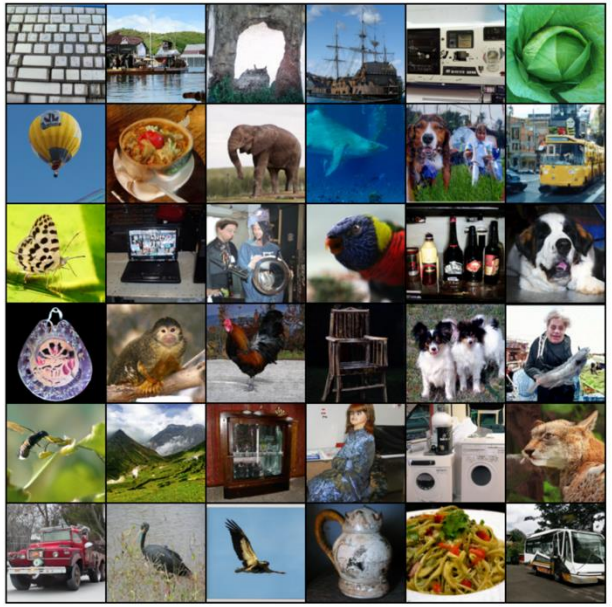
Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky T. Q. Chen, Gabriel Synnaeve, Yossi Adi, Yaron Lipman



Presented by Tony Qin

Introduction

Flow Matching for Discrete Data



$$x \in \mathbb{R}^d$$

Lipman 2022, “Flow Matching for Generative Modeling”

```python
def binary_search(arr, x):
    start = 0
    end = len(arr)-1

    # While performing binary search
    while start <= end:
        mid = (start + end) // 2
        # If x is greater
        if arr[mid] < x:
            start = mid + 1
        # If x is smaller
        elif arr[mid] > x:
            end = mid - 1
        else:
            return mid
    return -1
```

$$x = (x^1, x^2, \cdots, x^N) \in D = [d]^N$$

Gat 2024, “Discrete Flow Matching”

Flow Matching

Training objective:

$$L_{FM}(\theta) = \mathbb{E}_{t,p_t(x)} \| v_t(x) - u_t(x) \|^2$$

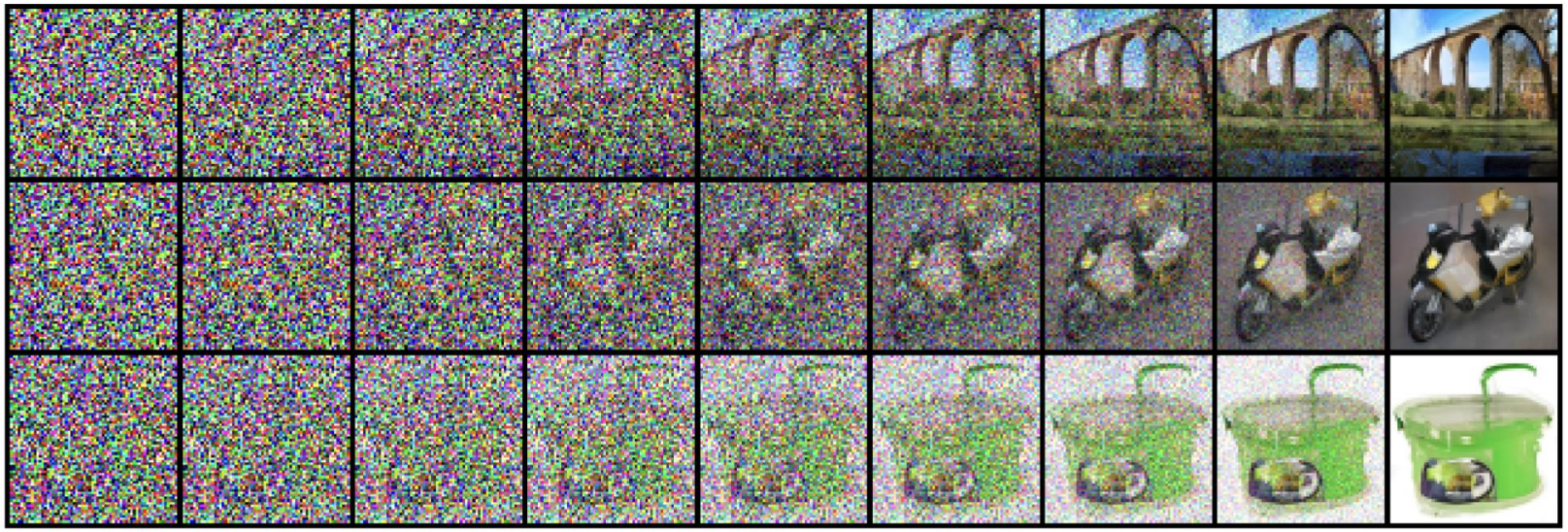
$$L_{CFM}(\theta) = \mathbb{E}_{t,q(x_1),p_t(x|x_1)} \| v_t(x) - u_t(x|x_1) \|^2$$

Sampling step:

$$X_{t+h}^i = X_t^i + h u_t^i(X_t)$$

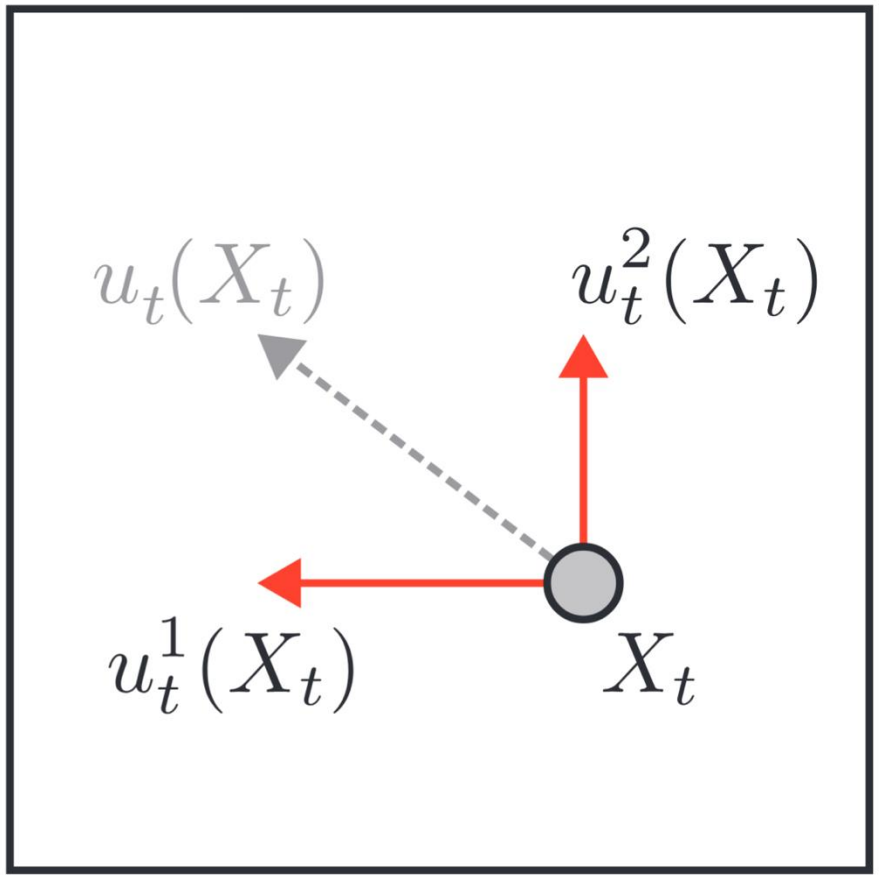
Flow Matching

Sampling step: $X_{t+h}^i = X_t^i + hu_t^i(X_t)$



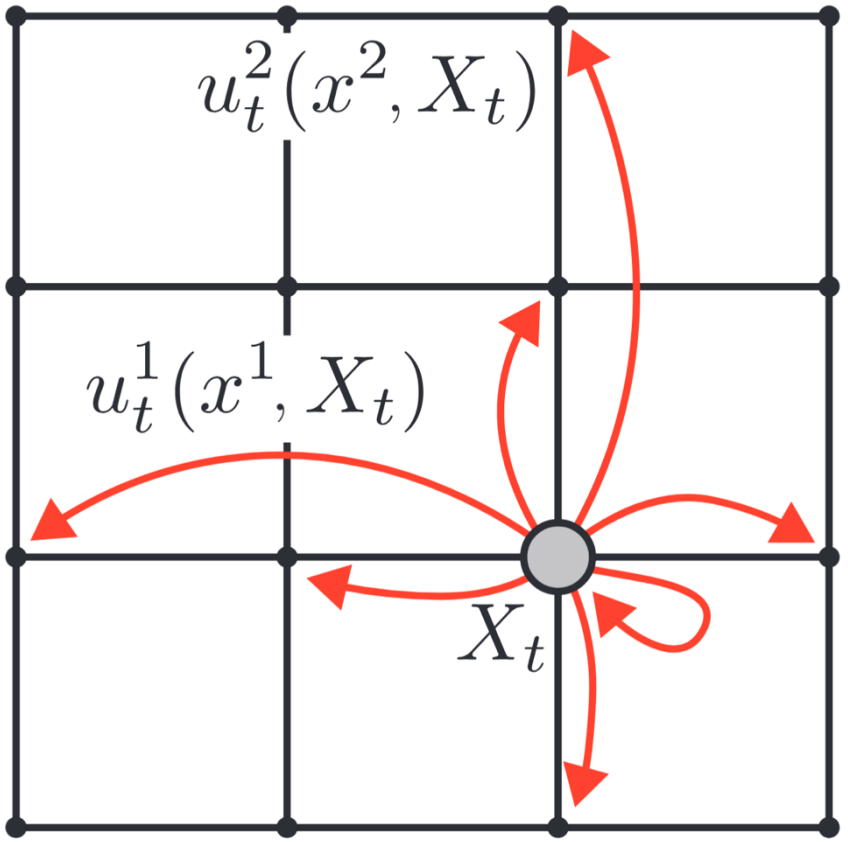
We want to learn a velocity $u_t(x)$ to transform a sample x_0 from a noisy distribution to a sample x_1 from a data distribution

Sampling step: $X_{t+h}^i = X_t^i + hu_t^i(X_t)$



Example velocity in $\mathbb{R}^2$

Discrete sampling step: $X_{t+h}^i \sim \delta_{X_t^i}(\cdot) + hu_t^i(\cdot, X_t)$

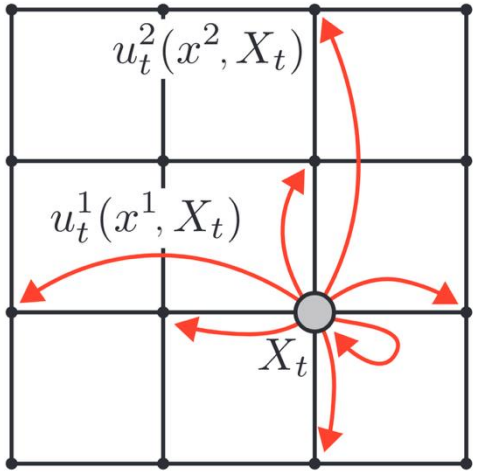


Example velocity in $[0, 1, 2, 3]^2$

Discrete sampling step: $X_{t+h}^i \sim \delta_{X_t^i}(\cdot) + h u_t^i(\cdot, X_t)$

$$\sum_{x^i \in [d]} u_t^i(x^i, z) = 0$$

$u_t^i(x^i, z) \geq 0$ for all $i \in [N]$ and $x^i \neq z^i$



$$z = [\text{green, dog}]$$

$u_t^1(\text{green}, [\text{green, dog}]) = -0.7$
$u_t^1(\text{brown}, [\text{green, dog}]) = 0.3$
$u_t^1(\text{white}, [\text{green, dog}]) = 0.2$
$u_t^1(\text{cat}, [\text{green, dog}]) = 0.01$
$u_t^1(\text{dog}, [\text{green, dog}]) = 0.008$
…

$u_t^2(\text{dog}, [\text{green, dog}]) = -0.4$
$u_t^2(\text{brown}, [\text{green, dog}]) = 0.02$
$u_t^2(\text{white}, [\text{green, dog}]) = 0.01$
$u_t^2(\text{cat}, [\text{green, dog}]) = 0.13$
$u_t^2(\text{monkey}, [\text{green, dog}]) = 0.08$
…

Training Discrete Flow Matching

Conditional Velocity

Consider:

$$p_t\left(x^i \middle| x_0, x_1\right) = (1-t)\delta_{x_0}\left(x^i\right) + t\delta_{x_1}\left(x^i\right)$$

What is the velocity that will generate this probability path?

Conditional Velocity

Consider:

$$p_t\left(x^i \middle| x_0, x_1\right) = (1-t)\delta_{x_0}\left(x^i\right) + t\delta_{x_1}(x^i)$$

What is the velocity that will generate this probability path?

Theorem 3:

$$u_t^i(x^i, z|x_0, x_1) = \frac{1}{1-t}\left[\delta_{x_1}\left(x^i\right) - \delta_z(x^i)\right]$$

Marginal Velocity

Consider:

$$p_t(x^i) = \sum_{x_0, x_1} p_t(x|x_0, x_1)\pi(x_0, x_1)$$

What is the velocity that will generate this probability path?

Theorem 3:

$$u_t^i(x^i, z|x_0, x_1) = \frac{1}{1-t}\left[\delta_{x_1}(x^i) - \delta_z(x^i)\right]$$

Marginalize:

$$u_t^i(x^i, z) = \sum_{x_0, x_1} \frac{1}{1-t}\left[\delta_{x_1}(x^i) - \delta_z(x^i)\right] p_t(x_0, x_1|z)$$

$$= \frac{1}{1-t}\left[\sum_{x_0, x_1} \delta_{x_1}(x^i) p_t(x_0, x_1|z) - \delta_z(x^i)\right]$$

Marginal Velocity

$$u_t^i(x^i, z) = \frac{1}{1-t}\left[\sum_{x_0,x_1} \delta_{x_1}(x^i) p_t(x_0, x_1|z) - \delta_z(x^i)\right]$$

$$= \frac{1}{1-t}[p_{1|t}(x^i|z) - \delta_z(x^i)]$$

Where

$$p_{1|t}(x^i|z) = \sum_{x_0,x_1} \delta_{x_1}(x^i) p_t(x_0, x_1|z)$$

$p_{1|t}(x^i|z)$ is the probability that x^i is the true token at $t = 1$ given the training data and the current sample z at time t

Training Objective

$$p_{1|t}(x^i|z) = \sum_{x_0,x_1} \delta_{x_1}(x^i) p_t(x_0, x_1|z)$$

$$L(\theta) = -\sum_{i\in[N]} \mathbb{E}_{t,(X_0,X_1),X_t} \log p_{1|t}(X_1^i|X_t;\theta)$$

We want the predicted probability of the true token x_1^i to be large, averaged over the whole dataset. This approximates $p_{1|t}$.

Experiments

Experiments: Language Modeling

METHOD	NFE	LLAMA-2↓	LLAMA-3↓	GPT2↓	ENTROPY
Data	-	7.0	9.4	14.7	7.7
Autoregressive	1024	31.4	54.8	45.3	7.1
Savinov et al. (2021)	200	29.5	45.1	34.7	5.2
Austin et al. (2021a)	1000	697.6	768.8	837.8	7.6
Han et al. (2022)	>10000	73.3	203.1	99.2	4.8
Lou et al. (2023)	256 / 512 / 1024	38.6 / 33.7 / 27.2	69.2 / 58.6 / 43.9	64.3 / 53.4 / 40.5	7.8 / 7.7 / 7.6
Campbell et al. (2024)	256 / 512 / 1024	38.5 / 33.5 / 28.7	69.0 / 56.5 / 46.5	65.2 / 53.3 / 43.0	7.8 / 7.7 / 7.6
FM (equation 9)	256 / 512 / 1024	34.2 / 30.0 / 22.5	58.5 / 48.8 / 33.8	54.2 / 43.5 / 29.3	7.7 / 7.6 / 7.2
FM (equation 10)	256 / 512 / 1024	30.0 / 27.5 / 22.3	48.2 / 43.5 / 31.9	47.7 / 41.8 / 28.1	7.6 / 7.5 / 7.1

Table 2 Generative perplexity on unconditional text generation compared to prior work. All models are sampled without the use of temperature or corrector steps. Double precision sampling results are reported in Table 5.

Number of function evaluations (NFE): higher is more expensive

Perplexity: lower is better

Entropy: higher is better

Experiments: Code Generation

Method	Data	HumanEval↑			MBPP (1-shot)↑		
		Pass@1	Pass@10	Pass@25	Pass@1	Pass@10	Pass@25
Autoregressive	Text	1.2	3.1	4.8	0.2	1.7	3.3
	Code	14.3	21.3	27.8	17.0	34.3	44.1
FM	Text	1.2	2.6	4.0	0.4	1.1	3.6
	Code	6.7	13.4	18.0	6.7	20.6	26.5
FM (Oracle length)	Code	11.6	18.3	20.6	13.1	28.4	34.2

Table 4 Execution based code generation evaluation.

Experiments: Code Generation

Method	Data	HumanEval↑			MBPP (1-shot)↑		
		Pass@1	Pass@10	Pass@25	Pass@1	Pass@10	Pass@25
Autoregressive	Text	1.2	3.1	4.8	0.2	1.7	3.3
	Code	14.3	21.3	27.8	17.0	34.3	44.1
FM	Text	1.2	2.6	4.0	0.4	1.1	3.6
	Code	6.7	13.4	18.0	6.7	20.6	26.5
FM (Oracle length)	Code	11.6	18.3	20.6	13.1	28.4	34.2

Table 4 Execution based code generation evaluation.

HumanEval: dataset of Python problems

M. Chen *et al.*, "Evaluating Large Language Models Trained on Code," Jul. 14, 2021, *arXiv*: arXiv:2107.03374. doi: 10.48550/arXiv.2107.03374.

MBPP: dataset of Python problems

J. Austin *et al.*, "Program Synthesis with Large Language Models," Aug. 16, 2021, *arXiv*: arXiv:2108.07732. doi: 10.48550/arXiv.2108.07732.

Experiments: Code Generation

Method	Data	HumanEval↑			MBPP (1-shot)↑		
		Pass@1	Pass@10	Pass@25	Pass@1	Pass@10	Pass@25
Autoregressive	Text	1.2	3.1	4.8	0.2	1.7	3.3
	Code	14.3	21.3	27.8	17.0	34.3	44.1
FM	Text	1.2	2.6	4.0	0.4	1.1	3.6
	Code	6.7	13.4	18.0	6.7	20.6	26.5
FM (Oracle length)	Code	11.6	18.3	20.6	13.1	28.4	34.2

Table 4 Execution based code generation evaluation.

Model trained on mostly text data performs worse

Model trained on mostly code performs better

Experiments: Code Generation

Method	Data	HumanEval↑			MBPP (1-shot)↑		
		Pass@1	Pass@10	Pass@25	Pass@1	Pass@10	Pass@25
Autoregressive	Text	1.2	3.1	4.8	0.2	1.7	3.3
	Code	14.3	21.3	27.8	17.0	34.3	44.1
FM	Text	1.2	2.6	4.0	0.4	1.1	3.6
	Code	6.7	13.4	18.0	6.7	20.6	26.5
FM (Oracle length)	Code	11.6	18.3	20.6	13.1	28.4	34.2

Table 4 Execution based code generation evaluation.

Why does knowing the correct code length improve code generation performance so much?

Conclusion

Strengths

- Extends continuous flow matching to discrete data
- First non-autoregressive method succeeding in non-trivial code generation
  - Can also do code infilling

Weaknesses

- Generates fixed length data
- Worse than autoregressive methods on code

Questions

“Is there a better method that doesn't need the oracle length to provide good results?”

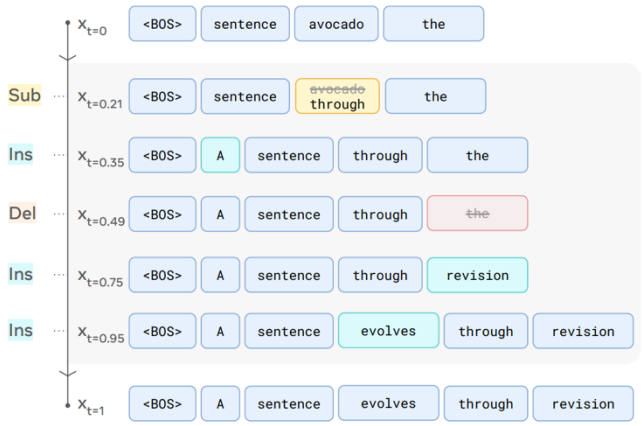


Figure 1 Edit Flow sampling process. Starting with x_0 containing random tokens or an empty sequence, the model applies edits to x_t and reaches a cohesive sentence at time $t = 1$.

Havasi et al. 2025 “Edit Flows: Flow Matching with Edit Operations”

Questions

“Since the model updates multiple tokens in parallel, how does it maintain consistency between tokens when generating code where small mistakes can break the whole program?”

Discrete sampling step: $X_{t+h}^{i} \sim \delta_{X_t^i}(\cdot) + hu_t^i(\cdot, X_t)$

As $h \to 0$, each sampling step will update very few tokens

Questions

“How can this model handle extra long input (millions of scale?) And how to optimize this kind of model (like we use kv cache?)”

This method may struggle with long inputs since it would require more sampling steps to maintain consistency. This would be very computationally expensive.