

# Limitations of Discrete Diffusion Models (dLLMs)

December 2025

# dLLMs are all the rage

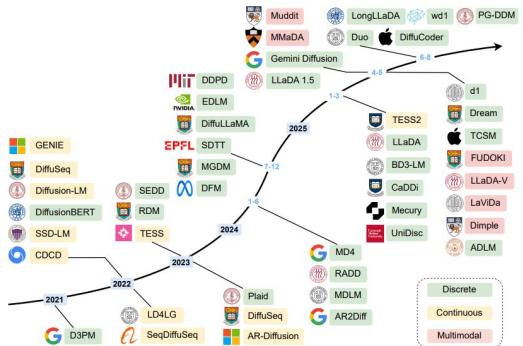
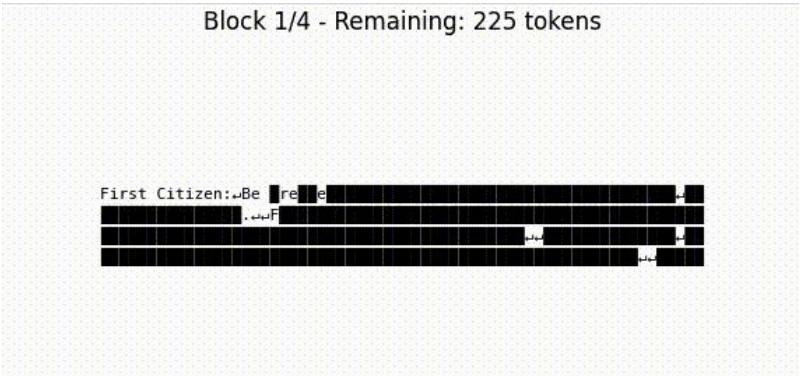


Fig. 1. Timeline of Diffusion Language Models. This figure highlights key milestones in the development of DLMs, categorized into three groups: continuous DLMs, discrete DLMs, and recent multimodal DLMs. We observe that while early research predominantly focused on continuous DLMs, discrete DLMs have gained increasing popularity in more recent years.



# dLLMs are all the rage

- They're weird

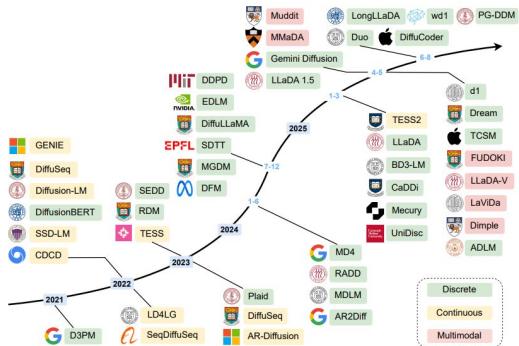
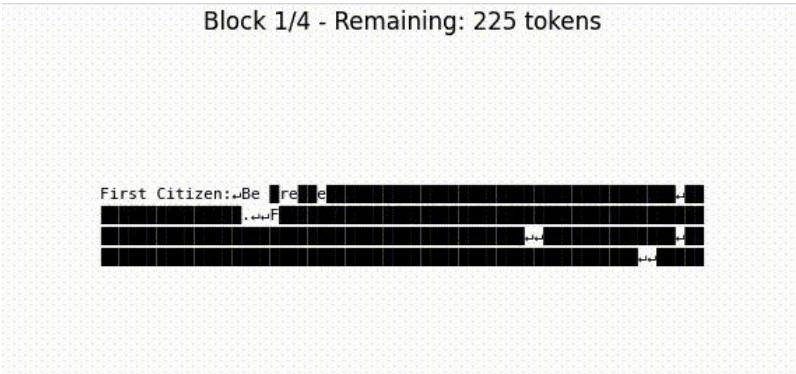


Fig. 1. Timeline of Diffusion Language Models. This figure highlights key milestones in the development of DLMs, categorized into three groups: continuous DLMs, discrete DLMs, and recent multimodal DLMs. We observe that while early research predominantly focused on continuous DLMs, discrete DLMs have gained increasing popularity in more recent years.



# dLLMs are all the rage

- They're weird
- They're fun

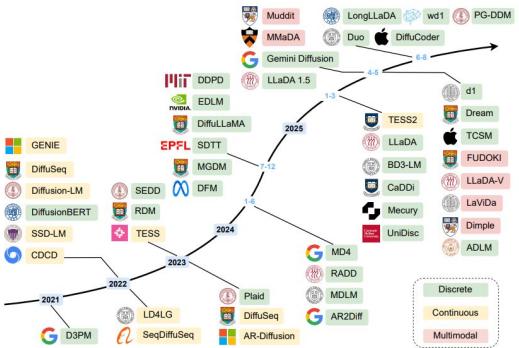
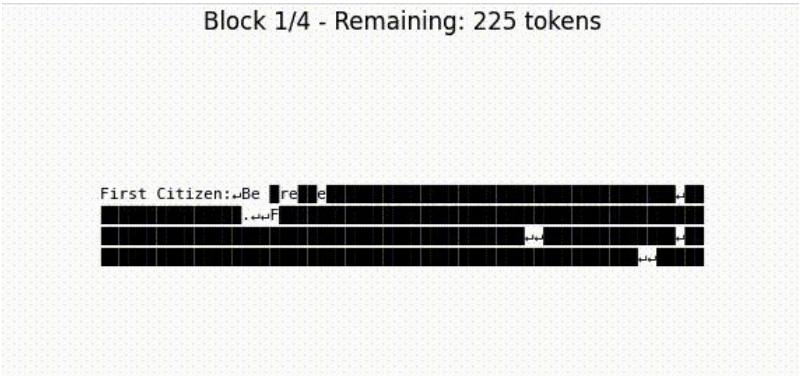


Fig. 1. Timeline of Diffusion Language Models. This figure highlights key milestones in the development of DLMs, categorized into three groups: continuous DLMs, discrete DLMs, and recent multimodal DLMs. We observe that while early research predominantly focused on continuous DLMs, discrete DLMs have gained increasing popularity in more recent years.



# dLLMs are all the rage

- They're weird
- They're fun
- They (seem to) scale OK

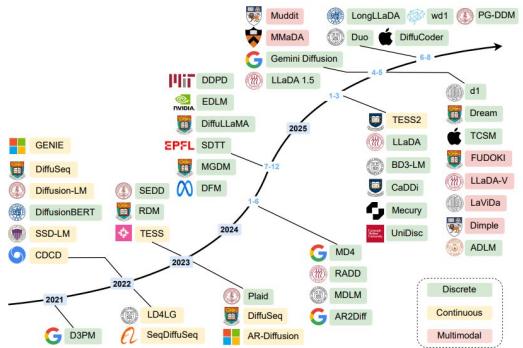
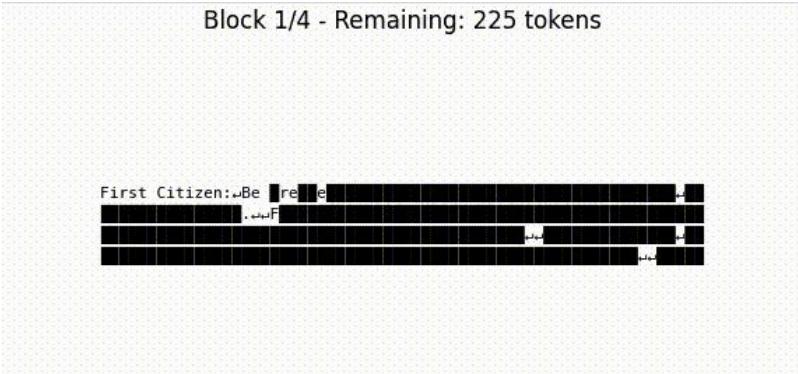


Fig. 1. Timeline of Diffusion Language Models. This figure highlights key milestones in the development of DLMs, categorized into three groups: continuous DLMs, discrete DLMs, and recent multimodal DLMs. We observe that while early research predominantly focused on continuous DLMs, discrete DLMs have gained increasing popularity in more recent years.



# dLLMs are all the rage

- They're weird
- They're fun
- They (seem to) scale OK
- They're better than ARMs in *some* tasks
  - E.g. reversal curse, sudoku

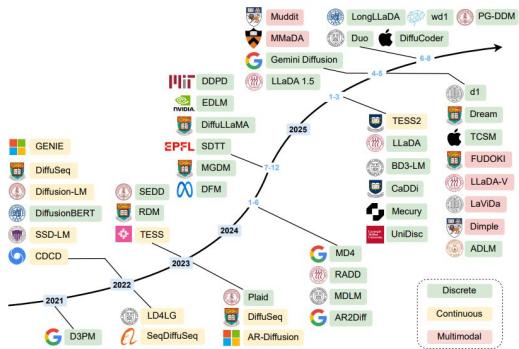
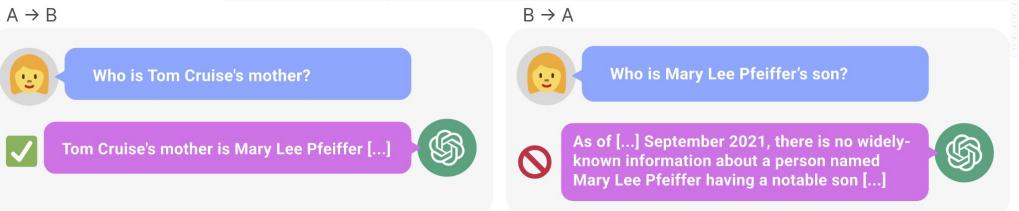
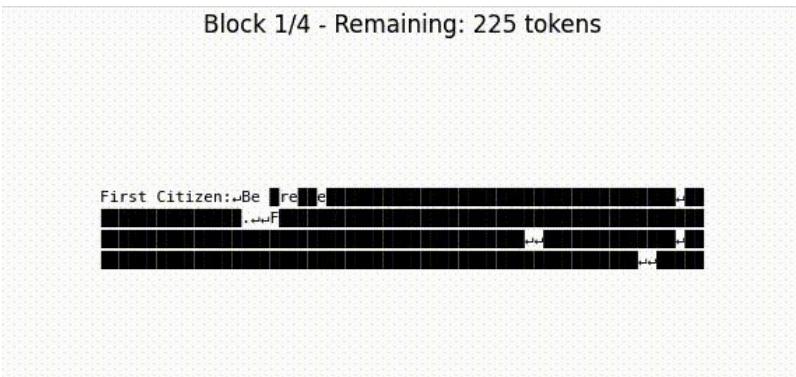


Fig. 1. Timeline of Diffusion Language Models. This figure highlights key milestones in the development of DLMs, categorized into three groups: continuous DLMs, discrete DLMs, and recent multimodal DLMs. We observe that while early research predominantly focused on continuous DLMs, discrete DLMs have gained increasing popularity in more recent years.



# dLLMs are all the rage

- They're weird
- They're fun
- They (seem to) scale OK
- They're better than ARMs in *some* tasks
  - E.g. reversal curse, sudoku
- They're fast!

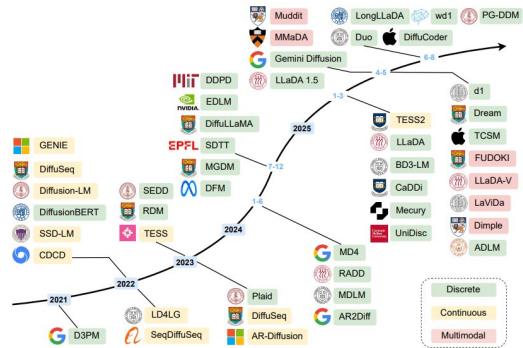
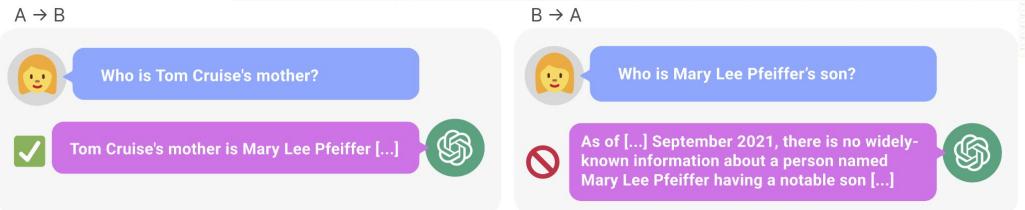
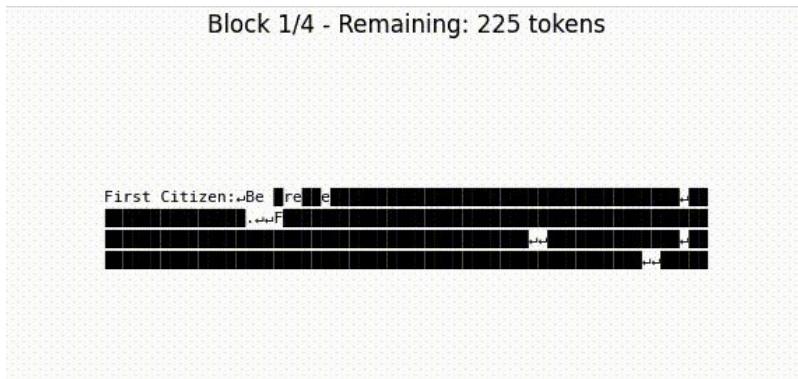
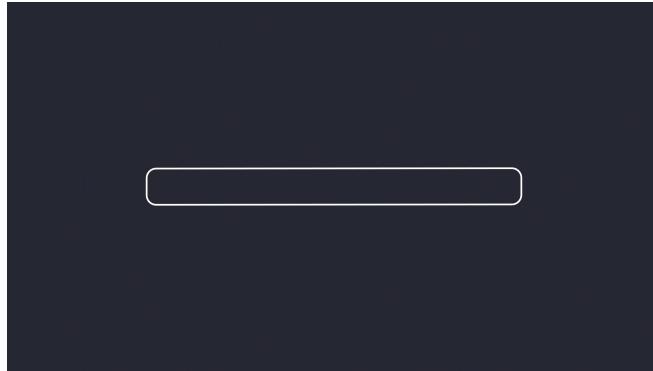


Fig. 1. Timeline of Diffusion Language Models. This figure highlights key milestones in the development of DLMs, categorized into three groups: continuous DLMs, discrete DLMs, and recent multimodal DLMs. We observe that while early research predominantly focused on continuous DLMs, discrete DLMs have gained increasing popularity in more recent years.



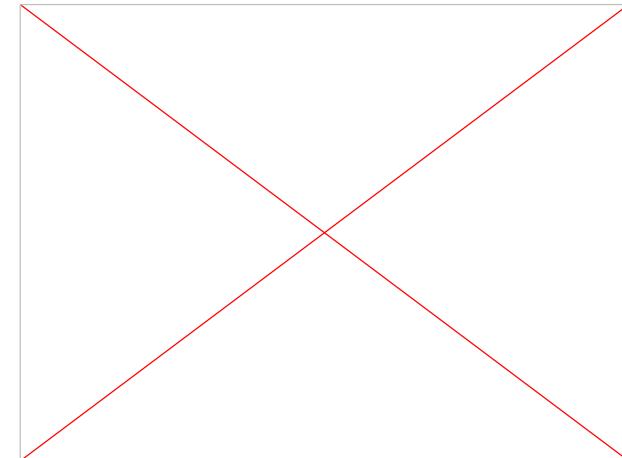
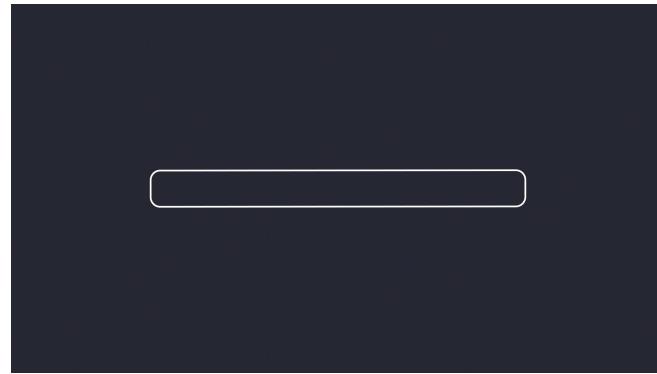
# They're fast in large part because of parallel decoding

- (i.e. because they can sample multiple tokens at a time)



# They're fast in large part because of parallel decoding

- (i.e. because they can sample multiple tokens at a time)
- (It's also pretty common now to do blockwise semi-autoregressive sampling)
  - Allows for KV-caching
- But some recent papers/blogs have questioned the limitations of parallel decoding and DLLMs in general



# Recap: Sampling from a dLLM

- Go from all-noise ([MASK]...[MASK]) at  $t = 1$  to noiseless at  $t = 0$

# Recap: Sampling from a dLLM

- Go from all-noise ([MASK]...[MASK]) at  $t = 1$  to noiseless at  $t = 0$
- At time  $t \in [0, 1]$  we will unmask token  $x_i^t$  at index  $i$  with probability  $\alpha_t$ .

## Recap: Sampling from a dLLM

- Go from all-noise ([MASK]...[MASK]) at  $t = 1$  to noiseless at  $t = 0$
- At time  $t \in [0, 1]$  we will unmask token  $x_i^t$  at index  $i$  with probability  $\alpha_t$ .
- For this token, we sample the new non-[MASK] token value using mask-prediction probabilities  $p_\theta(x_i^t) \in [0, 1]^{|\mathcal{V}|}$ .

# Recap: Sampling from a dLLM

- Go from all-noise ([MASK]...[MASK]) at  $t = 1$  to noiseless at  $t = 0$ .
- At time  $t \in [0, 1]$  we will unmask token  $x_i^t$  at index  $i$  with probability  $\alpha_t$ .
- For this token, we sample the new non-[MASK] token value using mask-prediction probabilities  $p_\theta(x_i^t) \in [0, 1]^{|\mathcal{V}|}$ .
- Many options are available for the schedule  $\alpha_t$ .

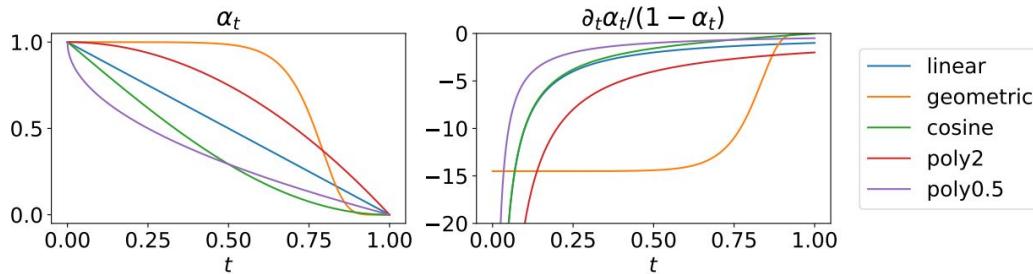


Figure 1: Masking schedules in the literature: (Left)  $\alpha_t$ ; (Right) weight of the cross-entropy loss w.r.t.  $t$ ; Equations for these schedules are given in Tab. 4 in Appendix.

# Recap: Sampling from a dLLM

- Go from all-noise ([MASK]...[MASK]) at  $t = 1$  to noiseless at  $t = 0$
- At time  $t \in [0, 1]$  we will unmask token  $x_i^t$  at index  $i$  with probability  $\alpha_t$ .
- For this token, we sample the new non-[MASK] token value using mask-prediction probabilities  $p_\theta(x_i^t) \in [0, 1]^{|\mathcal{V}|}$ .
- Many options are available for the schedule  $\alpha_t$ .

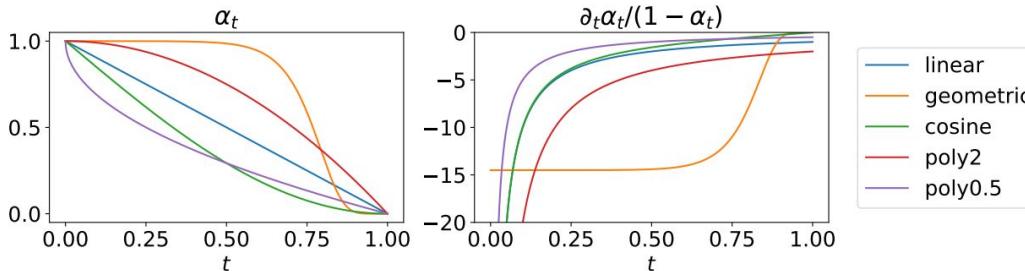


Figure 1: Masking schedules in the literature: (Left)  $\alpha_t$ ; (Right) weight of the cross-entropy loss w.r.t.  $t$ ; Equations for these schedules are given in Tab. 4 in Appendix.

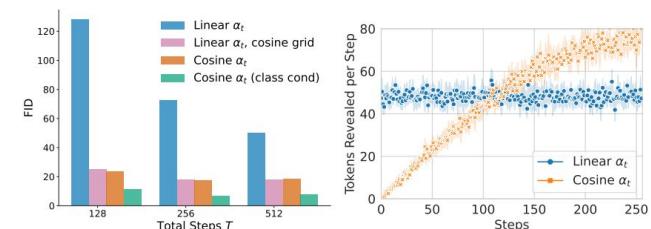


Figure 2: Left: FID evaluation for 50k samples randomly generated from MD4 on pixel-level modeling of ImageNet  $64 \times 64$  (numbers in Tab. 6). Right: Number of tokens revealed per generation step ( $T = 256$ ). Each image consists of  $64 \times 64 \times 3 = 12288$  tokens.

By default this is IID per token, but we can do cleverer stuff too

# Recap: Sampling from a dLLM

- Go from all-noise ([MASK]...[MASK]) at  $t = 1$  to noiseless at  $t = 0$
- At time  $t \in [0, 1]$  we will unmask token  $x_i^t$  at index  $i$  with probability  $\alpha_t$ .
- For this token, we sample the new non-[MASK] token value using mask-prediction probabilities  $p_\theta(x_i^t) \in [0, 1]^{|\mathcal{V}|}$ .
- Many options are available for the schedule  $\alpha_t$ .

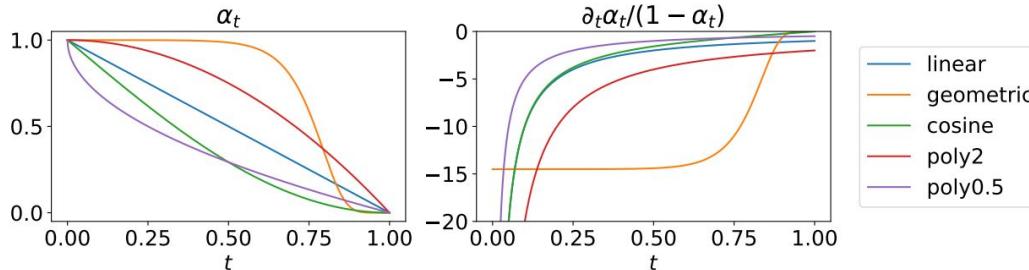


Figure 1: Masking schedules in the literature: (Left)  $\alpha_t$ ; (Right) weight of the cross-entropy loss w.r.t.  $t$ ; Equations for these schedules are given in Tab. 4 in Appendix.

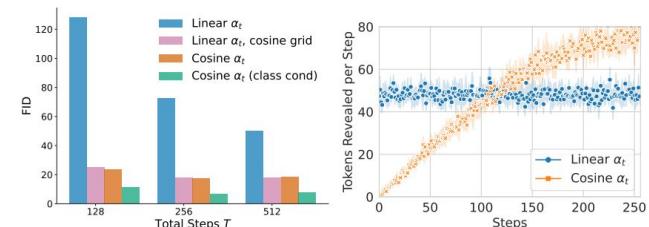
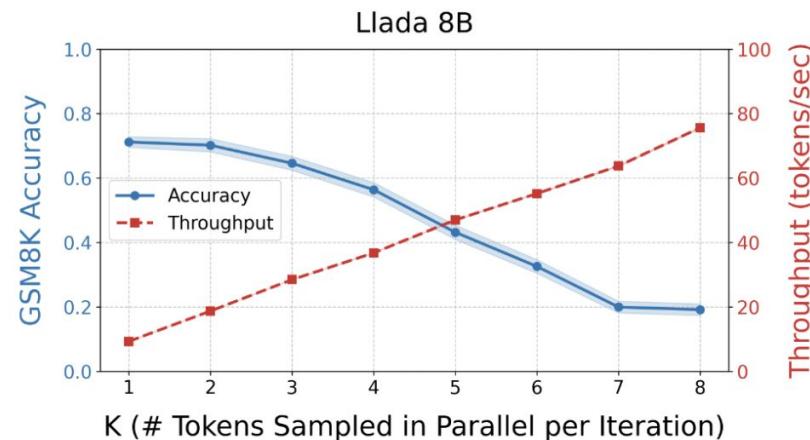
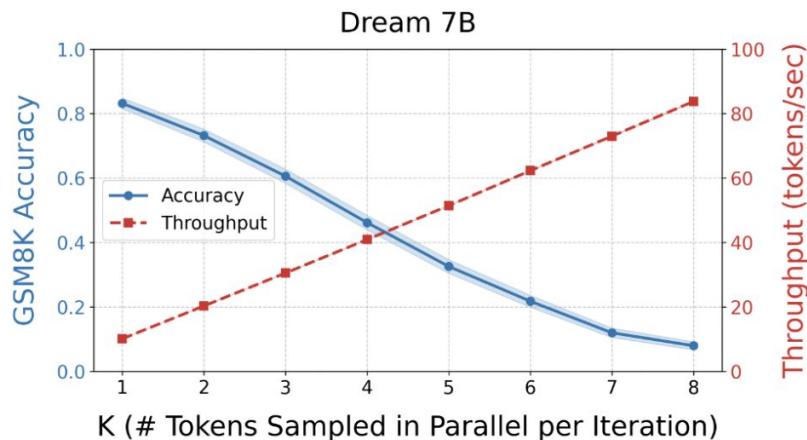


Figure 2: Left: FID evaluation for 50k samples randomly generated from MD4 on pixel-level modeling of ImageNet  $64 \times 64$  (numbers in Tab. 6). Right: Number of tokens revealed per generation step ( $T = 256$ ). Each image consists of  $64 \times 64 \times 3 = 12288$  tokens.

# But parallel decoding isn't always ideal...



# But parallel decoding isn't always ideal...

## PARALLELBENCH: UNDERSTANDING THE TRADE-OFFS OF PARALLEL DECODING IN DIFFUSION LLMs

**Wonjun Kang<sup>\*1,5</sup>**   **Kevin Galim<sup>\*1</sup>**   **Seunghyuk Oh<sup>\*1</sup>**   **Minjae Lee<sup>1</sup>**  
**Yuchen Zeng<sup>2,3</sup>**   **Shuibai Zhang<sup>2</sup>**   **Coleman Hooper<sup>4</sup>**   **Yuezhou Hu<sup>4</sup>**  
**Hyung Il Koo<sup>1</sup>**   **Nam Ik Cho<sup>5</sup>**   **Kangwook Lee<sup>2,6</sup>**

<sup>1</sup> FuriosaAI   <sup>2</sup> UW-Madison   <sup>3</sup> Microsoft Research   <sup>4</sup> UC Berkeley  
<sup>5</sup> Seoul National University   <sup>6</sup> KRAFTON AI

Project Page: <https://parallelbench.github.io>

# Example 1

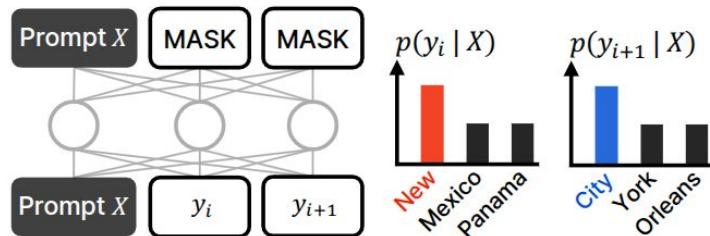
E.g. here, sampling  $y_i$  and  $y_{i+1}$  in parallel will always assign some positive probability to the incorrect answer “New City”

Issue

## Limitations of Parallel Decoding

Q. Pick a random city for travel: New York, New Orleans, Mexico City, or Panama City?

	$y_i$	$y_{i+1}$	Joint
One-by-One	$p(y_i X)$	$p(y_{i+1} X, y_i)$	$p(y_i, y_{i+1} X)$
Parallel	$p(y_i X)$	$p(y_{i+1} X)$	$p(y_i X) \cdot p(y_{i+1} X)$



A. New City

Parallel decoding ignores token dependencies

# Example 1

E.g. here, sampling  $y_i$  and  $y_{i+1}$  in parallel will always assign some positive probability to the incorrect answer “New City”

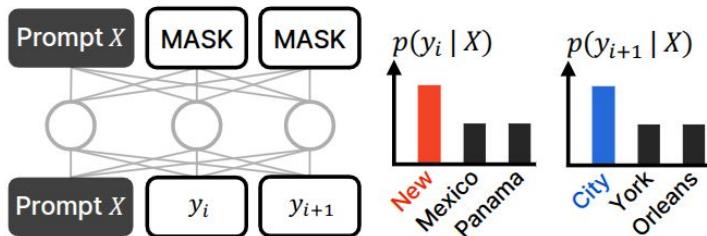
ARMs wouldn't have this problem: once “New” has been selected as  $y_i$ , then  $y_{i+1}$  will either be “York” or “Orleans”.

Issue

## Limitations of Parallel Decoding

Q. Pick a random city for travel: New York, New Orleans, Mexico City, or Panama City?

	$y_i$	$y_{i+1}$	Joint
One-by-One	$p(y_i X)$	$p(y_{i+1} X, y_i)$	$p(y_i, y_{i+1} X)$
Parallel	$p(y_i X)$	$p(y_{i+1} X)$	$p(y_i X) \cdot p(y_{i+1} X)$



A. New City

Parallel decoding ignores token dependencies

## Example 2

Similarly, sampling these three tokens in parallel leaves no way to ensure that the final output will have all three items “A”, “B”, and “C”.

\*Assuming that all tokens are unmasked in parallel

Q. Shuffle the following items: A, B, C.

$$\begin{array}{lll} \textcolor{red}{A} = 33\% & \textcolor{blue}{A} = 33\% & \textcolor{blue}{A} = 33\% \\ \textcolor{blue}{B} = 33\% & \textcolor{blue}{B} = 33\% & \textcolor{blue}{B} = 33\% \\ \textcolor{green}{C} = 33\% & \textcolor{green}{C} = 33\% & \textcolor{green}{C} = 33\% \end{array}$$

---

---

$$\text{Acc.}* = \frac{3}{3} \times \frac{2}{3} \times \frac{1}{3} = 22.2\%$$

Quality inevitably drops under parallel decoding

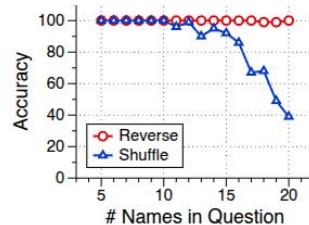
# dLLMs vs ARLLMs

This is (sort of) confirmed in real-world models

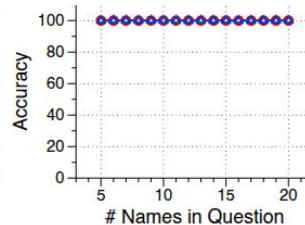
# dLLMs vs ARLLMs

This is (sort of) confirmed in real-world models

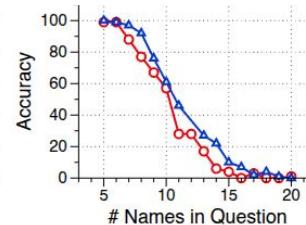
- Mercury (dLLM) maintains accuracy on ‘reverse’ task (as list length increases), but degrades with ‘shuffle’ task



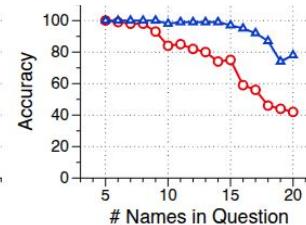
(a) Mercury



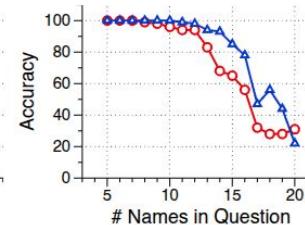
(b) Haiku 3.5



(c) Qwen2.5 3B



(d) Qwen3 4B



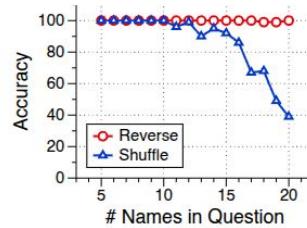
(e) Qwen2.5 7B

Figure 4: *Waiting Line* results on Mercury ([Inception Labs et al., 2025](#)) and autoregressive LLMs.

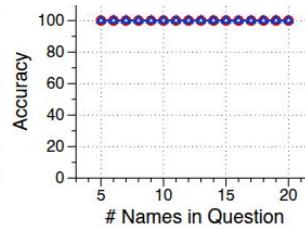
# dLLMs vs ARLLMs

This is (sort of) confirmed in real-world models

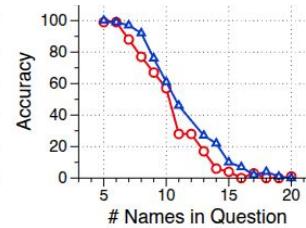
- Mercury (dLLM) maintains accuracy on ‘reverse’ task (as list length increases), but degrades with ‘shuffle’ task
- By contrast, ARM accuracy degrades on ‘reverse’ faster than on ‘shuffle’



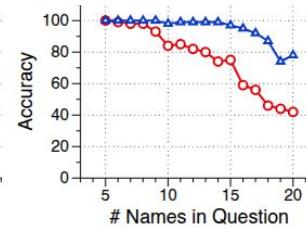
(a) Mercury



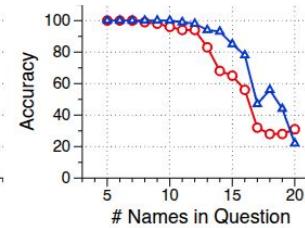
(b) Haiku 3.5



(c) Qwen2.5 3B



(d) Qwen3 4B



(e) Qwen2.5 7B

Figure 4: *Waiting Line* results on Mercury ([Inception Labs et al., 2025](#)) and autoregressive LLMs.

# The authors come up with ParallelBench with tasks demanding differing levels of parallelism

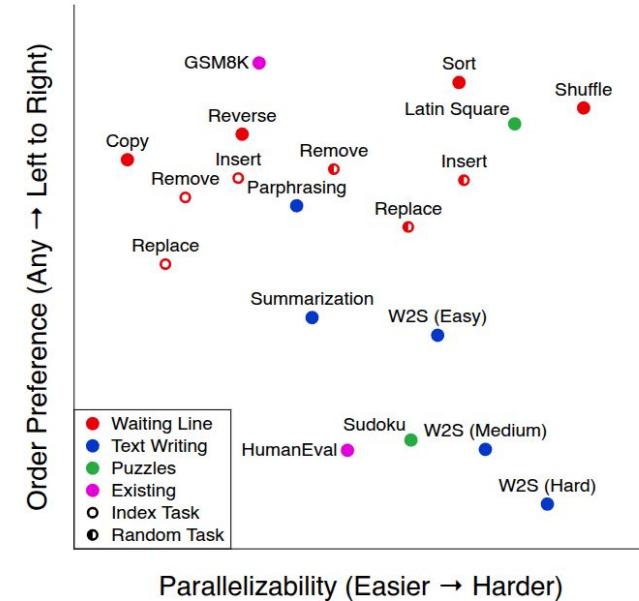
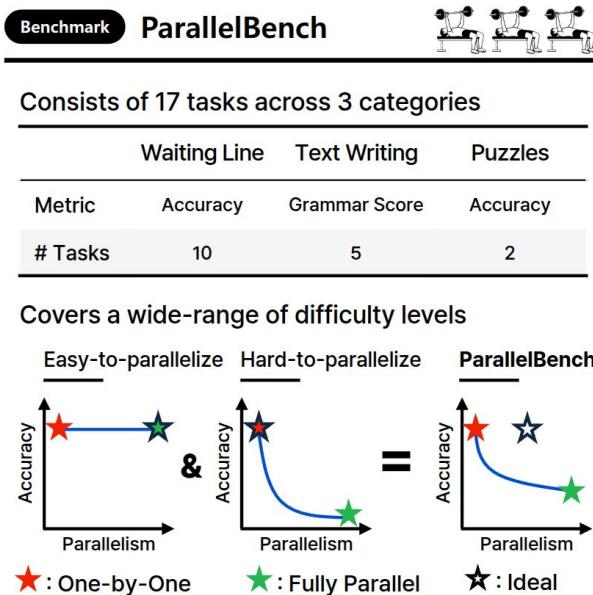
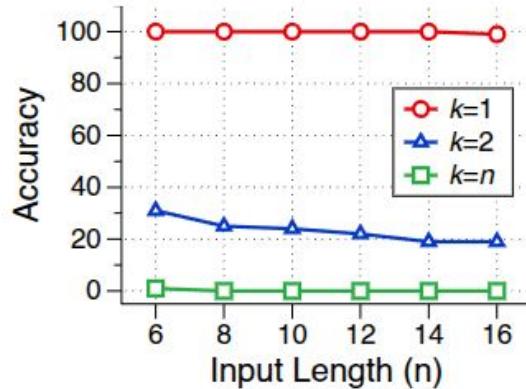
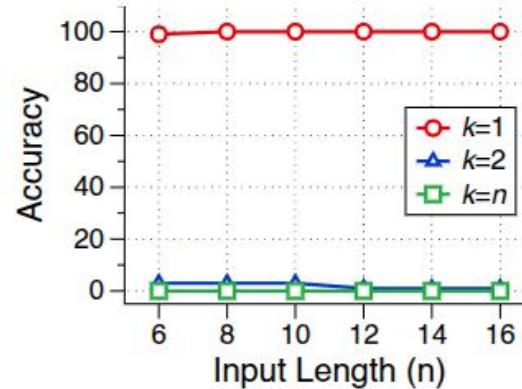


Figure 6: Broad coverage of PARALLELBENCH.

# E.g. Shuffle a list of length $n$



(a) Shuffle ( $\tau = 1$ )

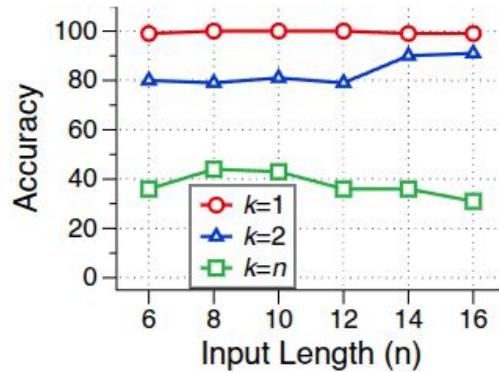


(b) Shuffle ( $\tau = 0$ )

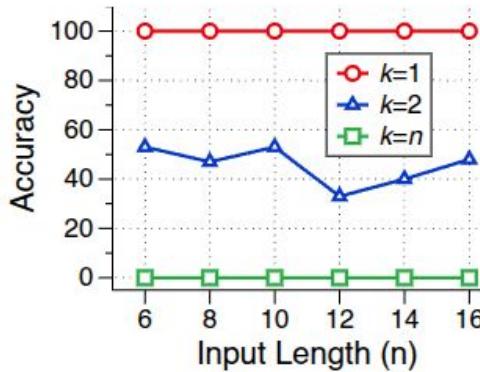
Tau = temperature

K = number of tokens to unmask at each step

E.g. Replace a specific item in a list of length  $n$



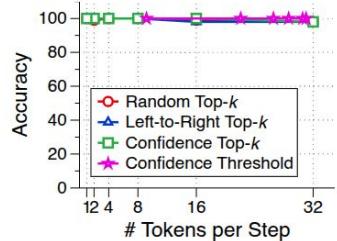
(c) Replace ( $\tau = 1$ )



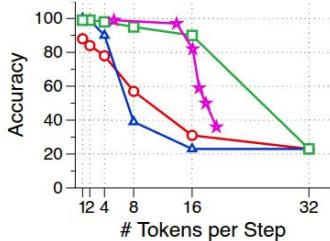
(d) Replace ( $\tau = 0$ )

Tau = temperature

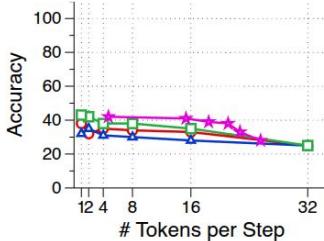
K = number of tokens to unmask at each step



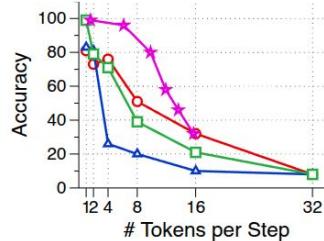
(a) Copy



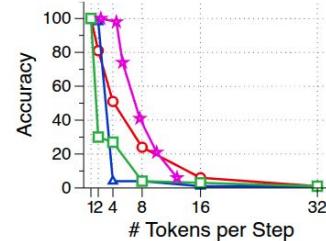
(b) Reverse



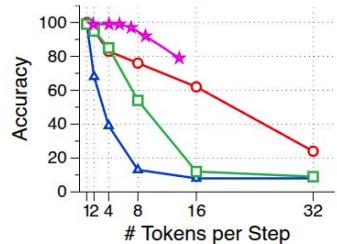
(c) Replace Index



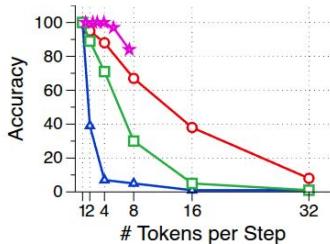
(d) Replace Random



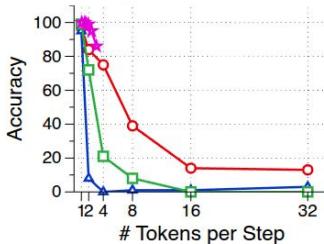
(e) Shuffle



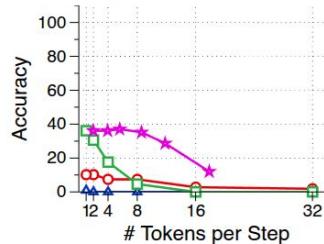
(f) Paraphrasing



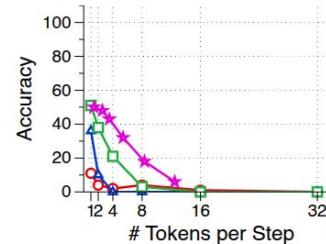
(g) W2S (easy)



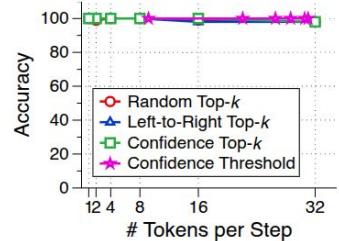
(h) W2S (hard)



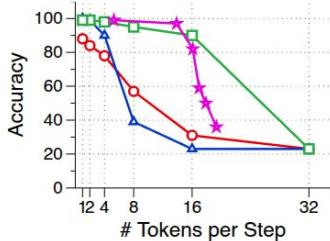
(i) Sudoku



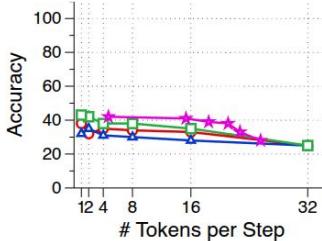
(j) Latin Square



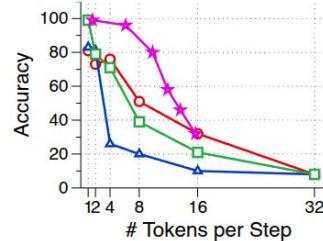
(a) Copy



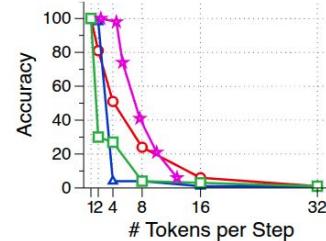
(b) Reverse



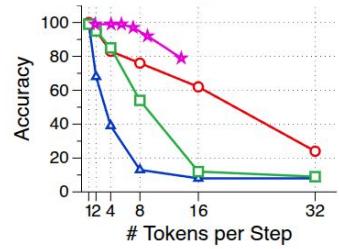
(c) Replace Index



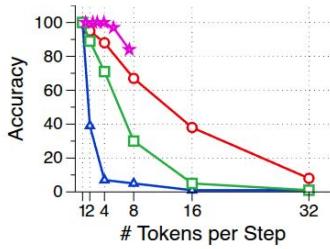
(d) Replace Random



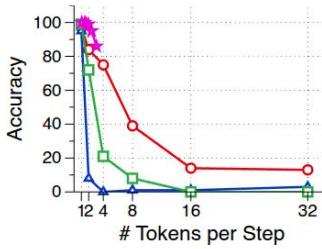
(e) Shuffle



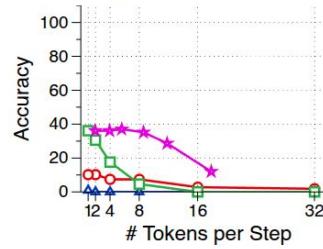
(f) Paraphrasing



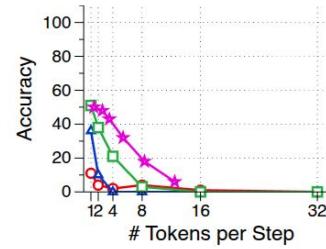
(g) W2S (easy)



(h) W2S (hard)

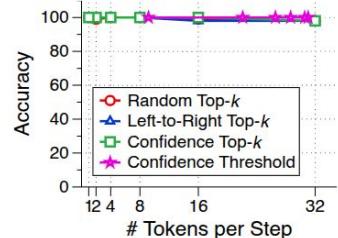


(i) Sudoku

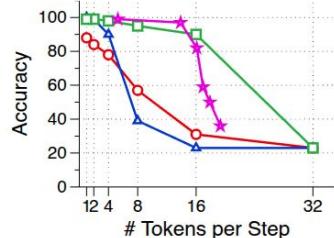


(j) Latin Square

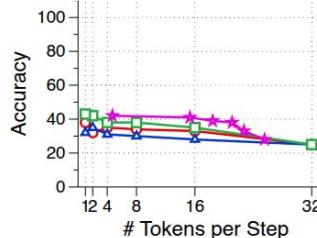
- **Random top-k:** unmask some random k tokens at each step



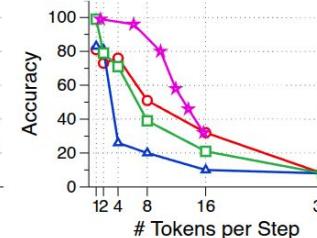
(a) Copy



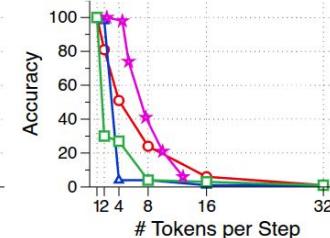
(b) Reverse



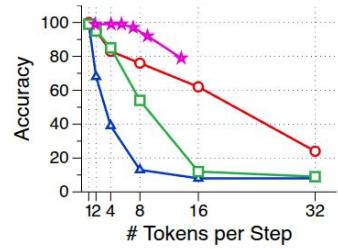
(c) Replace Index



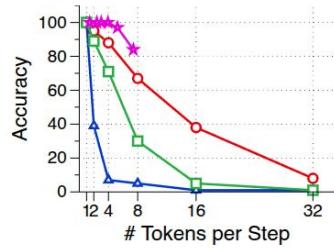
(d) Replace Random



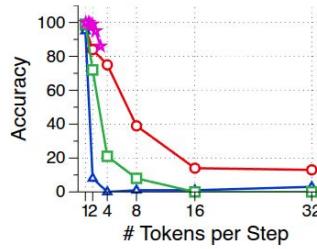
(e) Shuffle



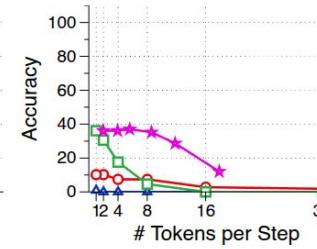
(f) Paraphrasing



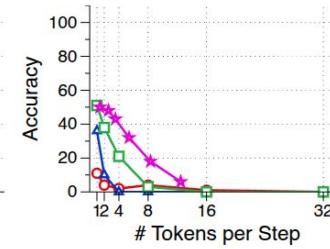
(g) W2S (easy)



(h) W2S (hard)

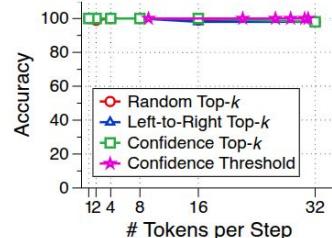


(i) Sudoku

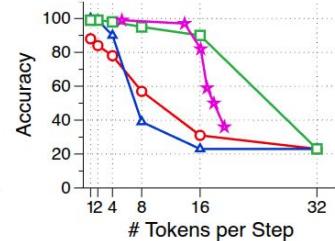


(j) Latin Square

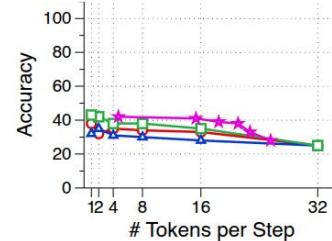
- **Random top- $k$ :** unmask some random  $k$  tokens at each step
- **L2R top- $k$ :** unmask the next  $k$  tokens



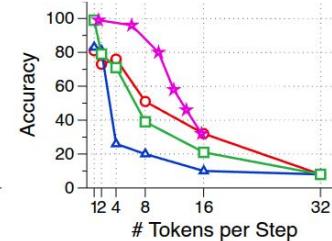
(a) Copy



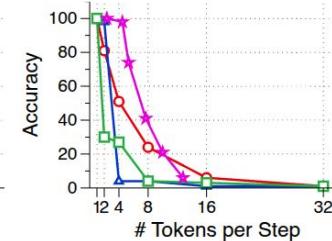
(b) Reverse



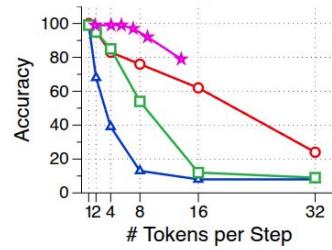
(c) Replace Index



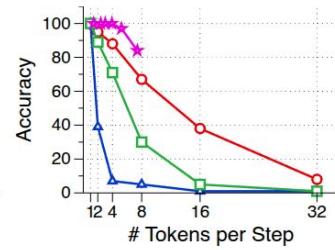
(d) Replace Random



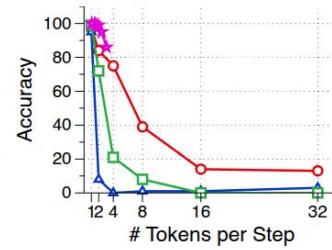
(e) Shuffle



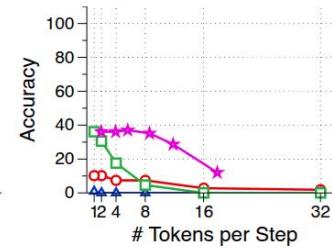
(f) Paraphrasing



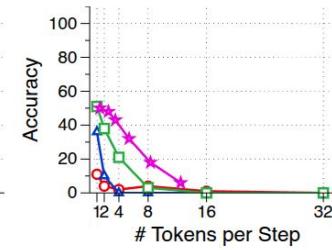
(g) W2S (easy)



(h) W2S (hard)

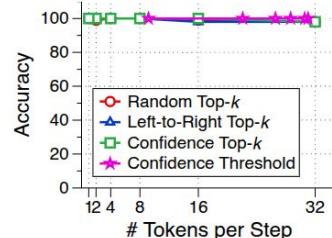


(i) Sudoku

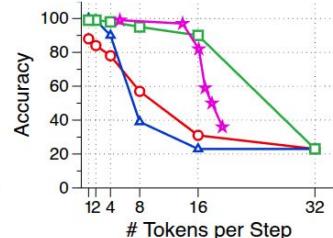


(j) Latin Square

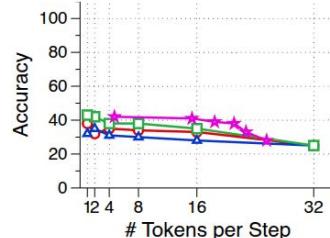
- **Random top-k:** unmask some random k tokens at each step
- **L2R top-k:** unmask the next k tokens
- **Confidence top-k:** unmask the K tokens with highest  $\max_v p_\theta(x_i^t)^{(v)}$



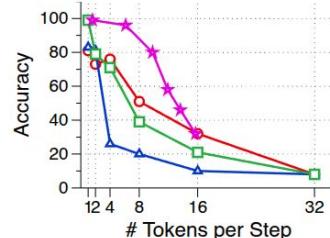
(a) Copy



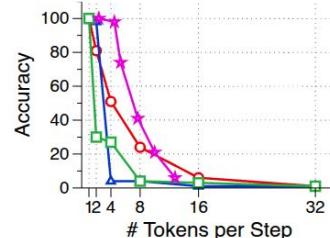
(b) Reverse



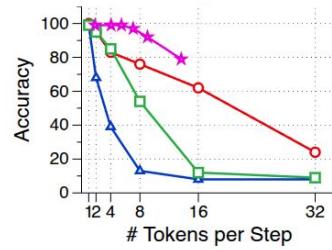
(c) Replace Index



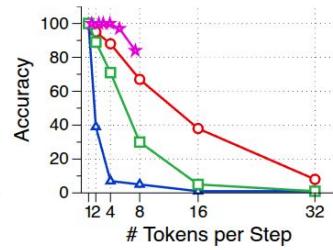
(d) Replace Random



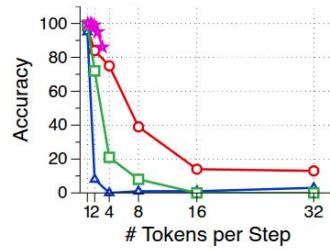
(e) Shuffle



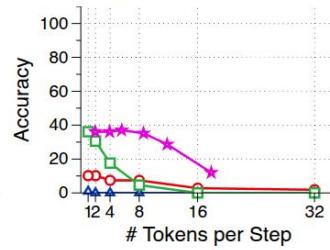
(f) Paraphrasing



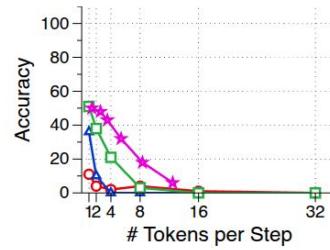
(g) W2S (easy)



(h) W2S (hard)

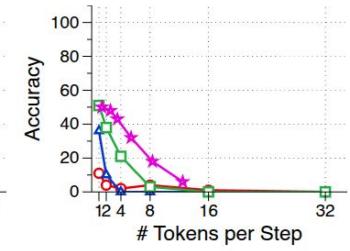
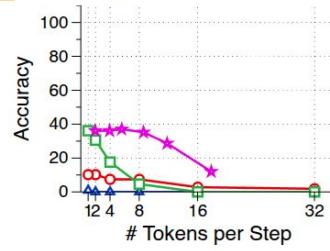
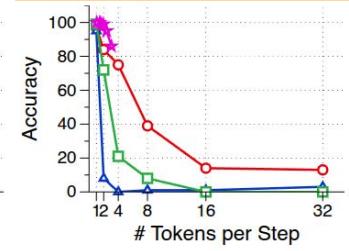
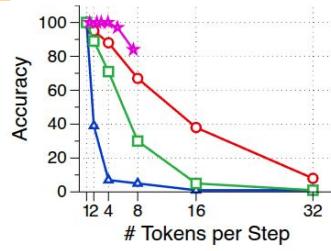
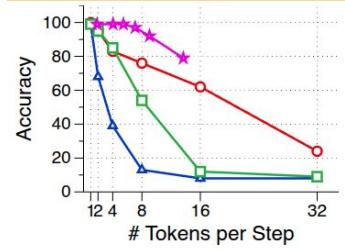
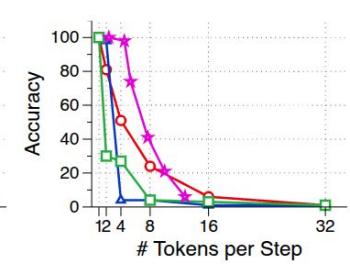
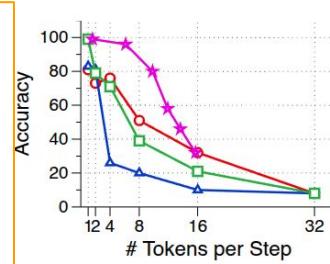
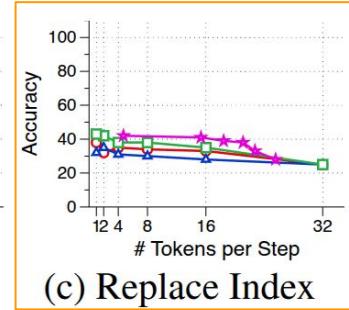
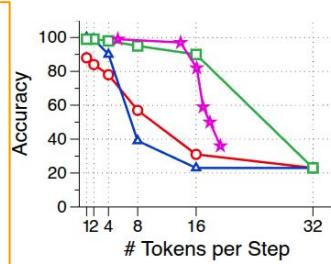
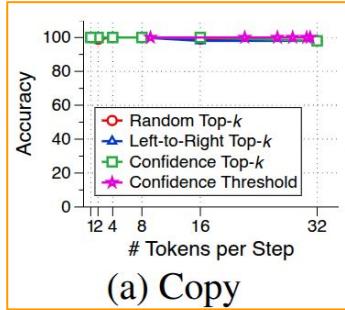


(i) Sudoku



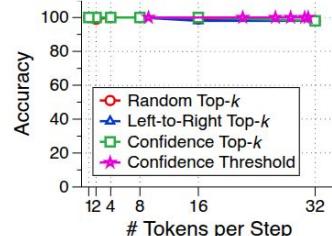
(j) Latin Square

- **Random top-k:** unmask some random k tokens at each step
- **L2R top-k:** unmask the next k tokens
- **Confidence top-k:** unmask the K tokens with highest  $\max_v p_\theta(x_i^t)^{(v)}$
- **Confidence Threshold:** unmask all tokens where  $\max_v p_\theta(x_i^t)^{(v)} > \gamma$

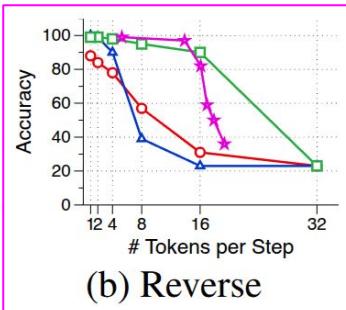


- **Random top-k:** unmask some random k tokens at each step
- **L2R top-k:** unmask the next k tokens
- **Confidence top-k:** unmask the K tokens with highest  $\max_v p_\theta(x_i^t)^{(v)}$
- **Confidence Threshold:** unmask all tokens where  $\max_v p_\theta(x_i^t)^{(v)} > \gamma$

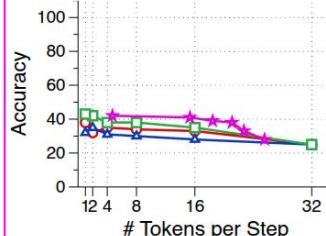
In all except (a) and (c), increased parallelism decreases performance



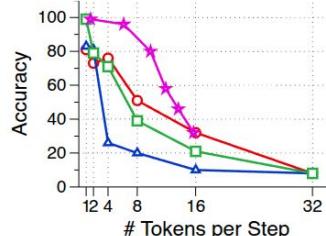
(a) Copy



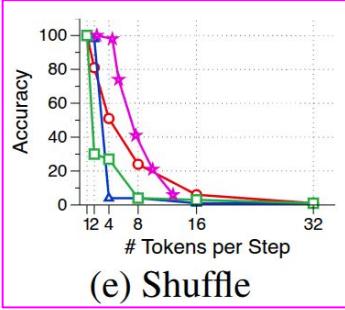
(b) Reverse



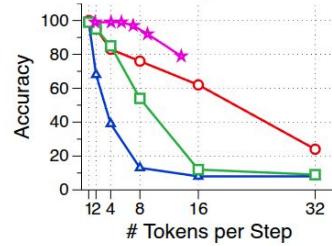
(c) Replace Index



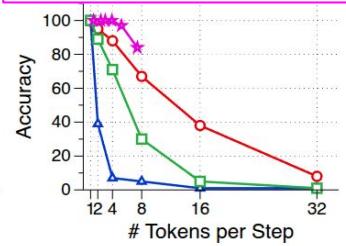
(d) Replace Random



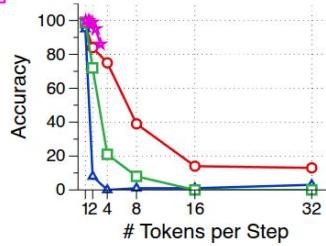
(e) Shuffle



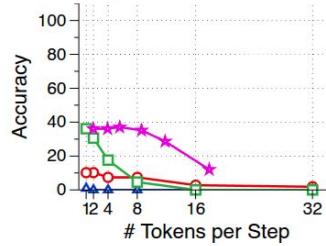
(f) Paraphrasing



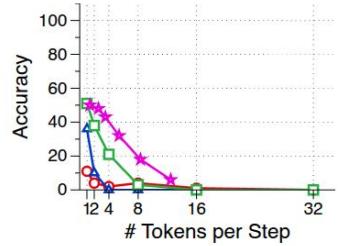
(g) W2S (easy)



(h) W2S (hard)



(i) Sudoku



(j) Latin Square

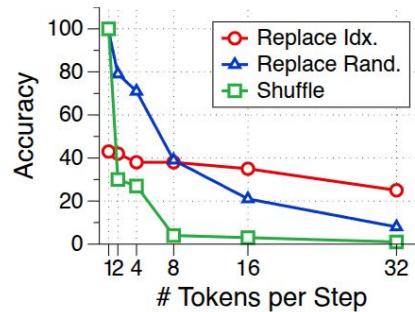
- **Random top-k:** unmask some random k tokens at each step
- **L2R top-k:** unmask the next k tokens
- **Confidence top-k:** unmask the K tokens with highest  $\max_v p_\theta(x_i^t)^{(v)}$
- **Confidence Threshold:** unmask all tokens where  $\max_v p_\theta(x_i^t)^{(v)} > \gamma$

In all except (a) and (c), increased parallelism decreases performance

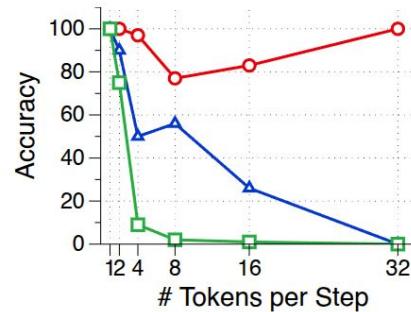
**Takeaway.** Static parallel decoding (e.g., top-k) can suffer severe quality degradation, and adaptive decoding strategies (e.g., threshold) still have significant room for improvement.

# How about using training the underlying model further?

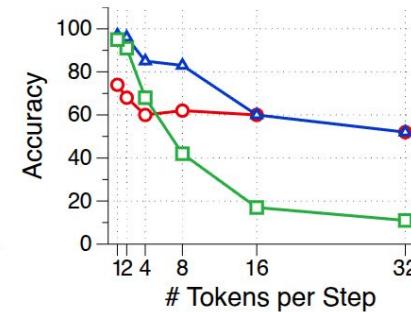
Using (static) random top-k:



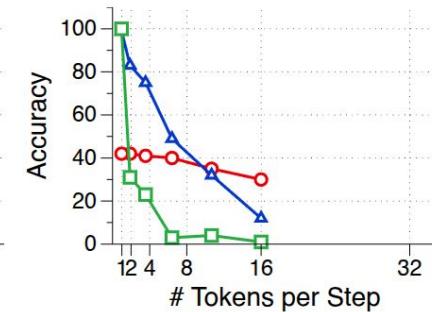
(a) Pretrained



(b) Fine-tuning



(c) Chain-of-Thought



(d) ReMDM

Side note: I'm confused by Replace Idx. in (b) getting better with increased parallelism...

# Summary of this paper

**Takeaway.** Static parallel decoding (e.g., top-k) can suffer severe quality degradation, and adaptive decoding strategies (e.g., threshold) still have significant room for improvement.

# Summary of this paper

**Takeaway.** Static parallel decoding (e.g., top-k) can suffer severe quality degradation, and adaptive decoding strategies (e.g., threshold) still have significant room for improvement.

We want *adaptive unmasking* strategies:

# Summary of this paper

**Takeaway.** Static parallel decoding (e.g., top-k) can suffer severe quality degradation, and adaptive decoding strategies (e.g., threshold) still have significant room for improvement.

We want *adaptive unmasking* strategies:

- Exploit parallel decoding when it's possible

# Summary of this paper

**Takeaway.** Static parallel decoding (e.g., top-k) can suffer severe quality degradation, and adaptive decoding strategies (e.g., threshold) still have significant room for improvement.

We want *adaptive unmasking* strategies:

- Exploit parallel decoding when it's possible
- When it's not possible, fall back on one-token-at-a-time decoding (~any order autoregressive model)

# Summary of this paper

**Takeaway.** Static parallel decoding (e.g., top-k) can suffer severe quality degradation, and adaptive decoding strategies (e.g., threshold) still have significant room for improvement.

We want *adaptive unmasking* strategies:

- Exploit parallel decoding when it's possible
- When it's not possible, fall back on one-token-at-a-time decoding (~any order autoregressive model)

... however, are we sure dLLMs can learn any-order AR generation effectively?

# Understanding the Limitations of Diffusion LLMs through a Probabilistic Perspective

Author: Cunxiao Du, Xinyu Yang, Min Lin, Chao Du and the team



Cunxiao Du

@ducx\_du

Follow ...

Diffusion LLMs (DLLM) can do “any-order” generation, in principle, more flexible than left-to-right (L2R) LLM.

Our main finding is uncomfortable:

➡ In real language, this flexibility backfires: DLLMs become worse probabilistic models than the L2R / R2L AR LMs.

This thread is about why “any order” turns into a curse.

(Work with Xinyu Yang [@Xinyu2ML](#), Min Lin [@mavenlin](#), Chao Du [@duchao0726](#) and the team.)

# Limitations of the any-order generation view of dLLMs

- From this point on, let's forget about parallel decoding and assume our dLLM only samples one token at a time

# Limitations of the any-order generation view of dLLMs

- From this point on, let's forget about parallel decoding and assume our dLLM only samples one token at a time
  - This isn't a reasonable assumption to make to represent *all* dLLM behaviour

# Limitations of the any-order generation view of dLLMs

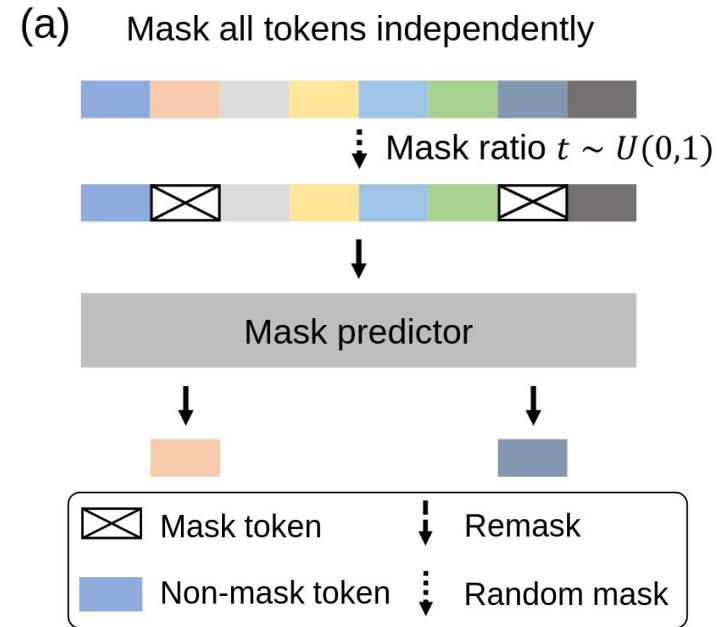
- From this point on, let's forget about parallel decoding and assume our dLLM only samples one token at a time
  - This isn't a reasonable assumption to make to represent *all* dLLM behaviour
  - But the ability to accurately decode one-token-at-a-time is a useful behaviour we want our dLLM to have

# Limitations of the any-order generation view of dLLMs

- From this point on, let's forget about parallel decoding and assume our dLLM only samples one token at a time
  - This isn't a reasonable assumption to make to represent *all* dLLM behaviour
  - But the ability to accurately decode one-token-at-a-time is a useful behaviour we want our dLLM to have
  - Parallelism is just the 'icing on the cake'

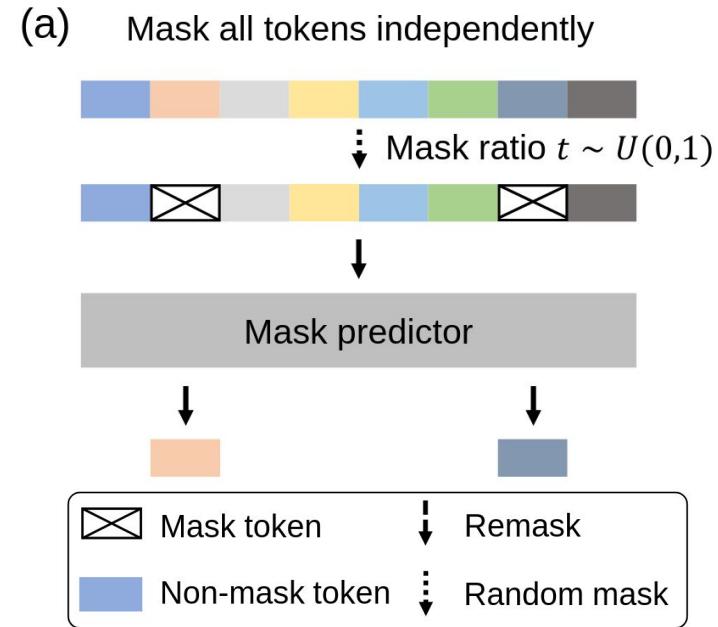
# Recap: Training a dLLM

- forward process: gradually mask tokens (independently) until fully masked at  $t = 1$ :
  - At time  $t \in [0, 1]$  each token is masked with prob  $t$ .
  - (Once a token is masked it stays masked for  $t' > t$ )



# Recap: Training a dLLM

- forward process: gradually mask tokens (independently) until fully masked at  $t = 1$ :
  - At time  $t \in [0, 1]$  each token is masked with prob  $t$ .
  - (Once a token is masked it stays masked for  $t' > t$ )
- reverse process: predict masked tokens as  $t$  moves from 0 to 1:

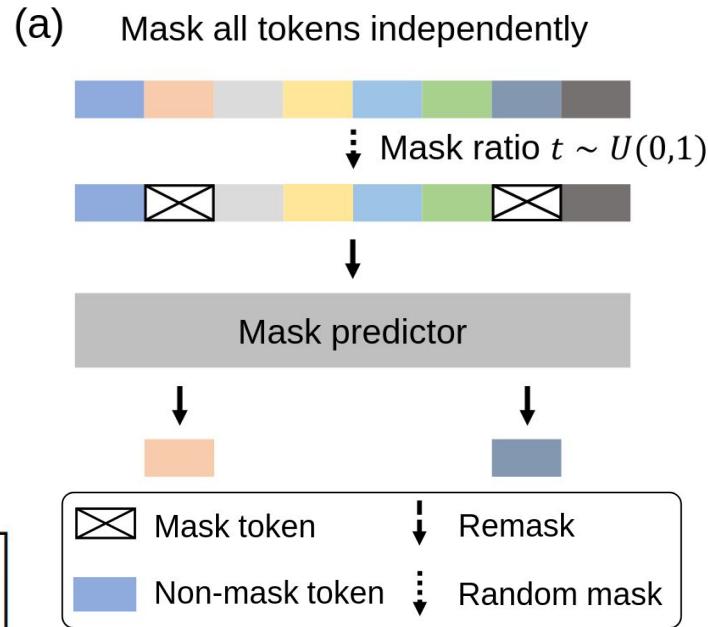


# Recap: Training a dLLM

- forward process: gradually mask tokens (independently) until fully masked at  $t = 1$ :
    - At time  $t \in [0, 1]$  each token is masked with prob  $t$ .
    - (Once a token is masked it stays masked for  $t' > t$ )
  - reverse process: predict masked tokens as  $t$  moves from 0 to 1:
    - Use a cross-entropy loss only on the masked tokens

$$\mathcal{L}(\theta) \triangleq -\mathbb{E}_{t,x_0,x_t} \left[ \frac{1}{t} \sum_{i=1}^L \mathbf{1}[x_t^i = \mathbf{M}] \log p_\theta(x_0^i | x_t) \right]$$

where  $x_0 \sim \mathcal{D}_{\text{train}}$ ,  $t \sim \text{Uniform}[0, 1]$ , and  $x_t$  is sampled from the forward process.



# Any-Order Factorisation

With any permutation  $o : [L] \rightarrow [L]$  over token indices, we can factorise our dLLM into an autoregressive formulation

$$p(x) = \prod_i p(x_{o(i)} | x_{o(<i)})$$

# Any-Order Factorisation

With any permutation  $o : [L] \rightarrow [L]$  over token indices, we can factorise our dLLM into an autoregressive formulation

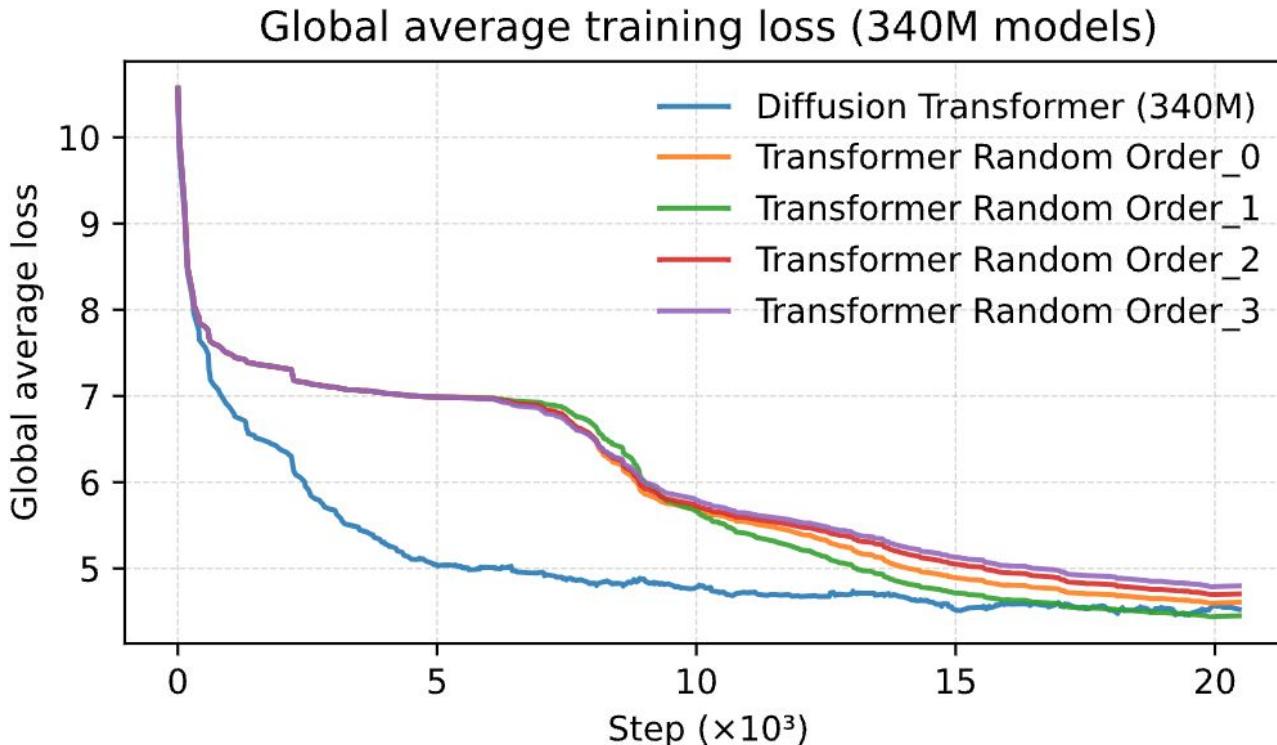
$$p(x) = \prod_i p(x_{o(i)} | x_{o(<i)})$$

Where

$x_{o(<i)}$  are the visible/unmasked tokens

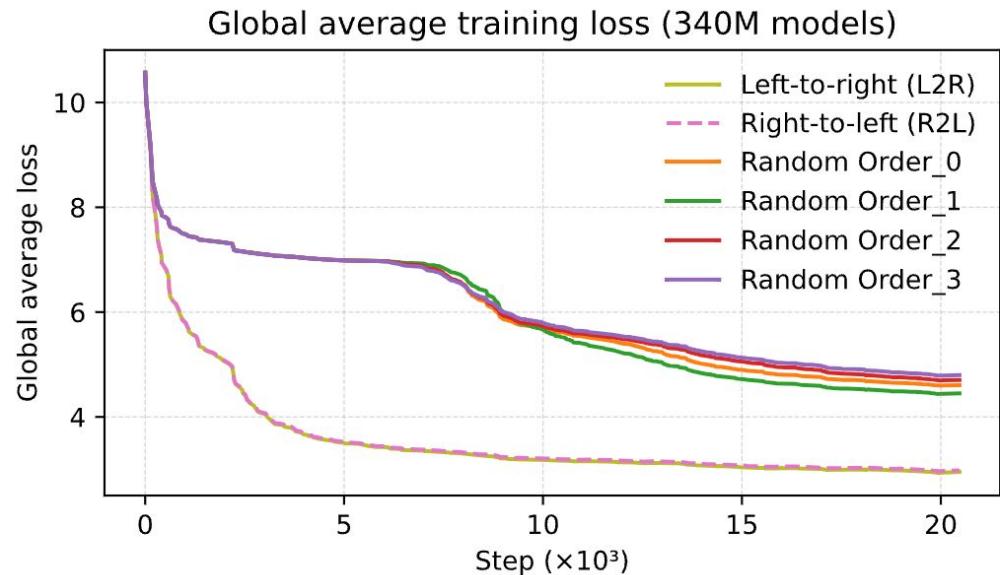
$x_{o(i)}$  is the next masked token to be predicted

# Random-order Transformers work about as well as a dLLM (at the end of training)



# But is ‘random ordered’ actually desirable?

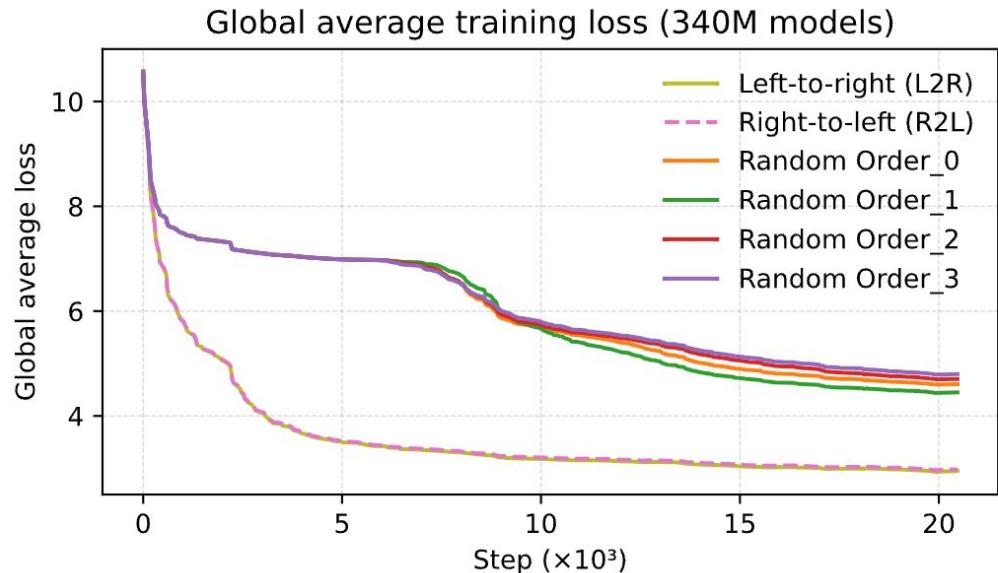
(perhaps unsurprisingly) no.



# But is ‘random ordered’ actually desirable?

(perhaps unsurprisingly) no.

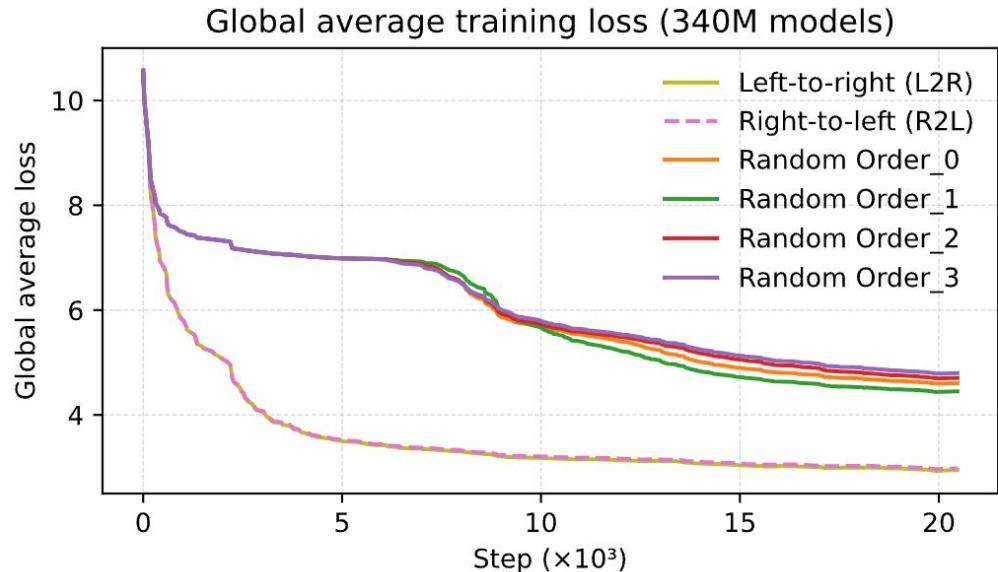
- L2R or R2L are easier to model



# But is ‘random ordered’ actually desirable?

(perhaps unsurprisingly) no.

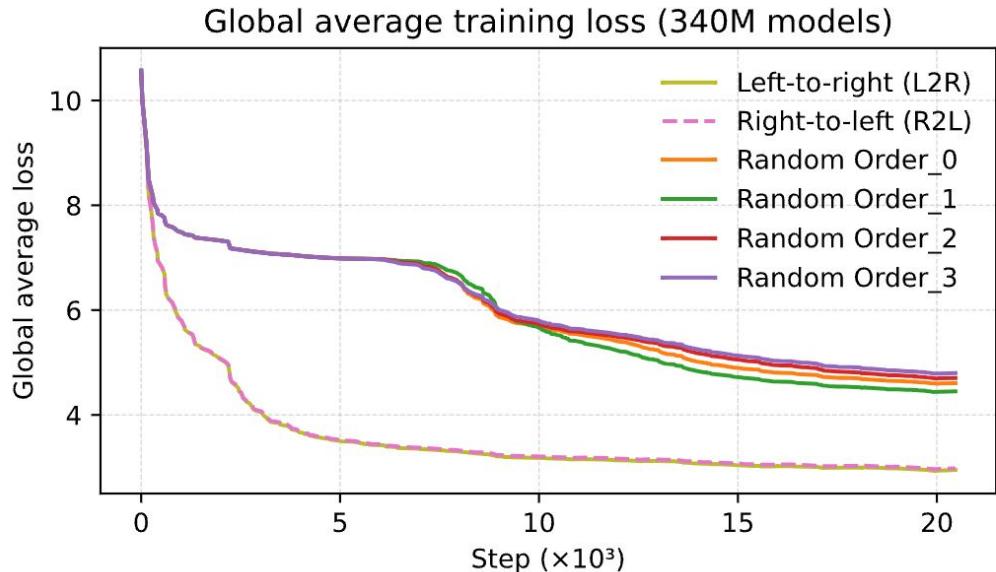
- L2R or R2L are easier to model
- Since a dLLM optimises all orders uniformly they:
  - Don’t naturally concentrate capacity on more favourable orderings (L2R, R2L)



# But is ‘random ordered’ actually desirable?

(perhaps unsurprisingly) no.

- L2R or R2L are easier to model
- Since a dLLM optimises all orders uniformly they:
  - Don’t naturally concentrate capacity on more favourable orderings (L2R, R2L)
  - Obtain a significantly looser approximation to the underlying data distribution



# Why is L2R or R2L better than a random order?

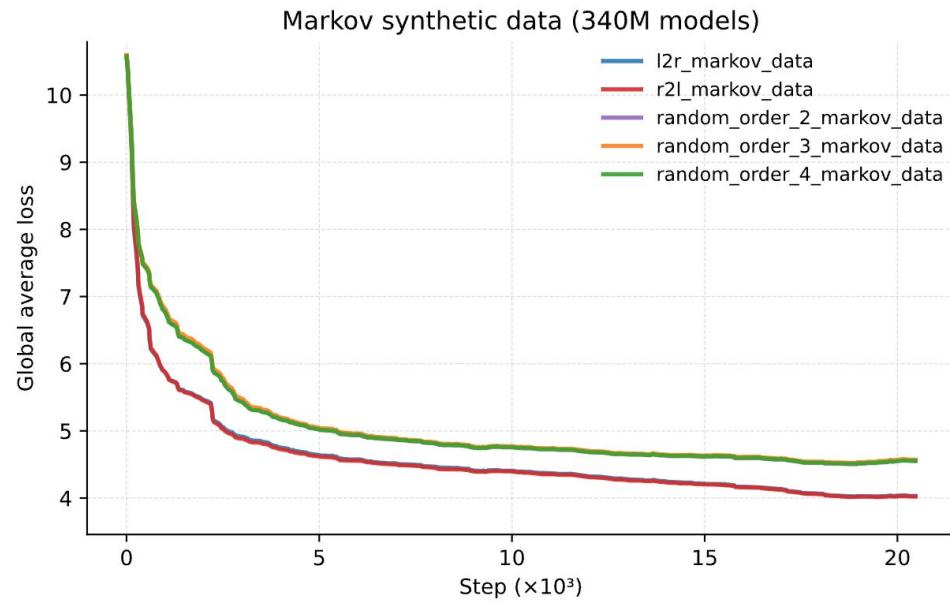
“Most data in natural language approximately satisfies a Markov assumption”

$$p(x_n | x_{1:n-1}) \approx p(x_n | x_{n-k:n-1})$$

# Why is L2R or R2L better than a random order?

“Most data in natural language approximately satisfies a Markov assumption”

$$p(x_n | x_{1:n-1}) \approx p(x_n | x_{n-k:n-1})$$

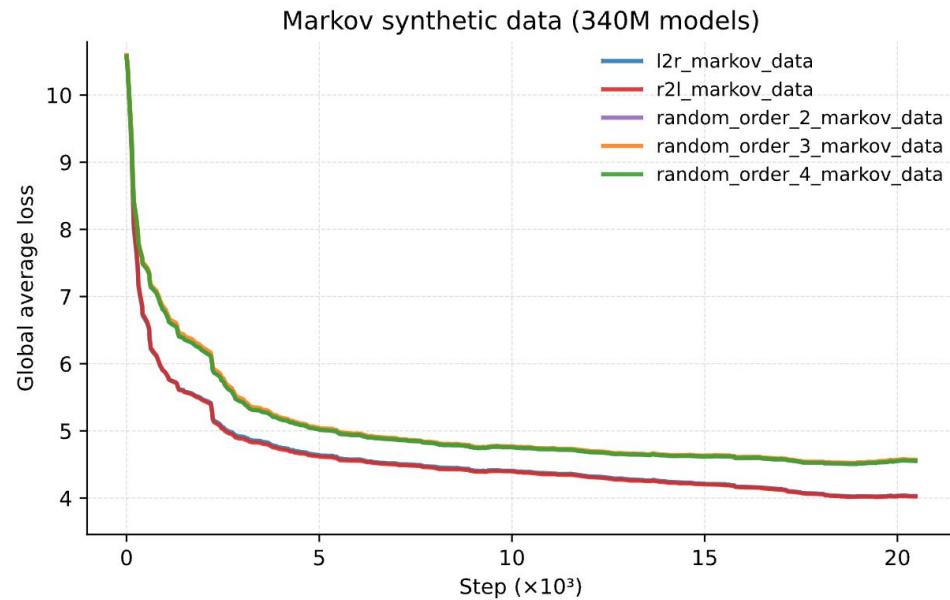


# Why is L2R or R2L better than a random order?

“Most data in natural language approximately satisfies a Markov assumption”

$$p(x_n | x_{1:n-1}) \approx p(x_n | x_{n-k:n-1})$$

So random ordering/permutations increase the required Markov neighbourhood size  $k$



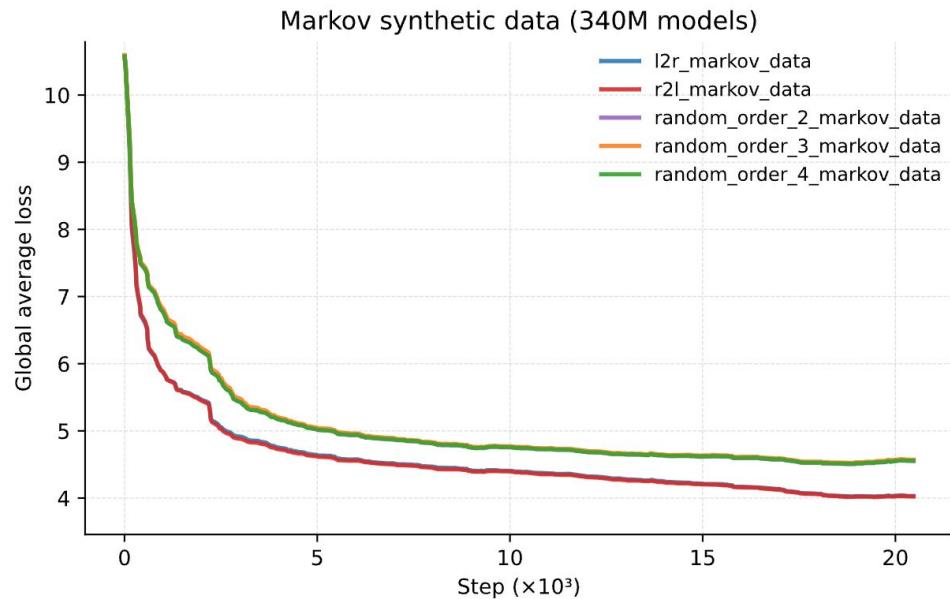
# Why is L2R or R2L better than a random order?

“Most data in natural language approximately satisfies a Markov assumption”

$$p(x_n | x_{1:n-1}) \approx p(x_n | x_{n-k:n-1})$$

So random ordering/permutations increase the required Markov neighbourhood size  $k$

Although full  $L \times L$  attention *could* solve this, it's still harder to learn the optimal solution for all permutations at once



# Can we fix the ELBO?

- dLLMs optimise the loss with a sum-log objective

$$\begin{aligned}\mathcal{L}_{\text{diff}} &= \mathbb{E}_{q_{\text{mask}}} [-\log p_{\theta}(x_{\text{mask}} | x_{\text{non-mask}})] \\ &= -\frac{1}{|\mathcal{O}|} \sum_{o \in \mathcal{O}} \log p_{\theta}^{(o)}(x)\end{aligned}$$

# Can we fix the ELBO?

- dLLMs optimise the loss with a sum-log objective

$$\begin{aligned}\mathcal{L}_{\text{diff}} &= \mathbb{E}_{q_{\text{mask}}} [-\log p_{\theta}(x_{\text{mask}} | x_{\text{non-mask}})] \\ &= -\frac{1}{|\mathcal{O}|} \sum_{o \in \mathcal{O}} \log p_{\theta}^{(o)}(x)\end{aligned}$$

- But maybe instead we should optimise a log-sum objective

$$\mathcal{L}_{\text{correct}} = -\log \sum_{o \in \mathcal{O}} p(o) p_{\theta}^{(o)}(x)$$

# Can we fix the ELBO?

- dLLMs optimise the loss with a sum-log objective

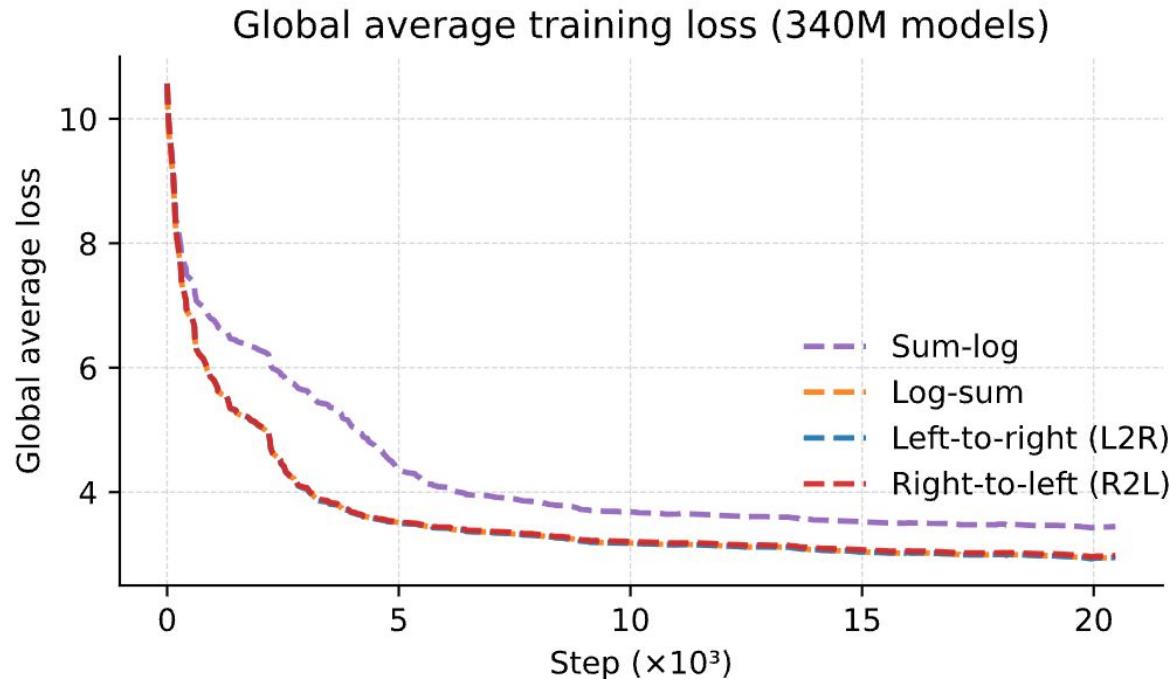
$$\begin{aligned}\mathcal{L}_{\text{diff}} &= \mathbb{E}_{q_{\text{mask}}} [-\log p_{\theta}(x_{\text{mask}} | x_{\text{non-mask}})] \\ &= -\frac{1}{|\mathcal{O}|} \sum_{o \in \mathcal{O}} \log p_{\theta}^{(o)}(x)\end{aligned}$$

- But maybe instead we should optimise a log-sum objective

$$\mathcal{L}_{\text{correct}} = -\log \sum_{o \in \mathcal{O}} p(o) p_{\theta}^{(o)}(x)$$

- This will get dominated by the best generation order, rather than requiring all orders to generate well

With four permutations  $\mathcal{O} = \{\text{L2R}, \text{R2L}, o_{\text{random}}^{(1)}, o_{\text{random}}^{(2)}\}$  we see that the log-sum loss does recover the optimal loss of L2R & R2L (albeit with 4x the computation)



**4. Maybe diffusion is not bad, masked diffusion LLMs are bad**

## Conclusion

## 4. Maybe diffusion is not bad, masked diffusion LLMs are bad

# Conclusion

- I don't think this is a death knell for dLLMs

## 4. Maybe diffusion is not bad, masked diffusion LLMs are bad

# Conclusion

- I don't think this is a death knell for dLLMs
- Even if L2R gives us a better loss than some random permutation, that doesn't mean L2R is the optimal ordering

## 4. Maybe diffusion is not bad, masked diffusion LLMs are bad

# Conclusion

- I don't think this is a death knell for dLLMs
- Even if L2R gives us a better loss than some random permutation, that doesn't mean L2R is the optimal ordering
  - Especially for a lot of the ParallelBench tasks

## 4. Maybe diffusion is not bad, masked diffusion LLMs are bad

# Conclusion

- I don't think this is a death knell for dLLMs
- Even if L2R gives us a better loss than some random permutation, that doesn't mean L2R is the optimal ordering
  - Especially for a lot of the ParallelBench tasks
- Anyway, there's a tradeoff between performance and speed

## 4. Maybe diffusion is not bad, masked diffusion LLMs are bad

# Conclusion

- I don't think this is a death knell for dLLMs
- Even if L2R gives us a better loss than some random permutation, that doesn't mean L2R is the optimal ordering
  - Especially for a lot of the ParallelBench tasks
- Anyway, there's a tradeoff between performance and speed
- Plus, dLLMs are basically only a year old at this point

## 4. Maybe diffusion is not bad, masked diffusion LLMs are bad

# Conclusion

- I don't think this is a death knell for dLLMs
- Even if L2R gives us a better loss than some random permutation, that doesn't mean L2R is the optimal ordering
  - Especially for a lot of the ParallelBench tasks
- Anyway, there's a tradeoff between performance and speed
- Plus, dLLMs are basically only a year old at this point
  - There are lots of fancy things we've yet to try

## 4. Maybe diffusion is not bad, masked diffusion LLMs are bad

# Conclusion

- I don't think this is a death knell for dLLMs
- Even if L2R gives us a better loss than some random permutation, that doesn't mean L2R is the optimal ordering
  - Especially for a lot of the ParallelBench tasks
- Anyway, there's a tradeoff between performance and speed
- Plus, dLLMs are basically only a year old at this point
  - There are lots of fancy things we've yet to try
  - Edit-diffusion models seem promising imo

### Edit Flows: Flow Matching with Edit Operations

Marton Havasi<sup>1</sup>, Brian Karrer<sup>1</sup>, Itai Gat<sup>1</sup>, Ricky T.Q. Chen<sup>1</sup>

<sup>1</sup>FAIR at Meta

