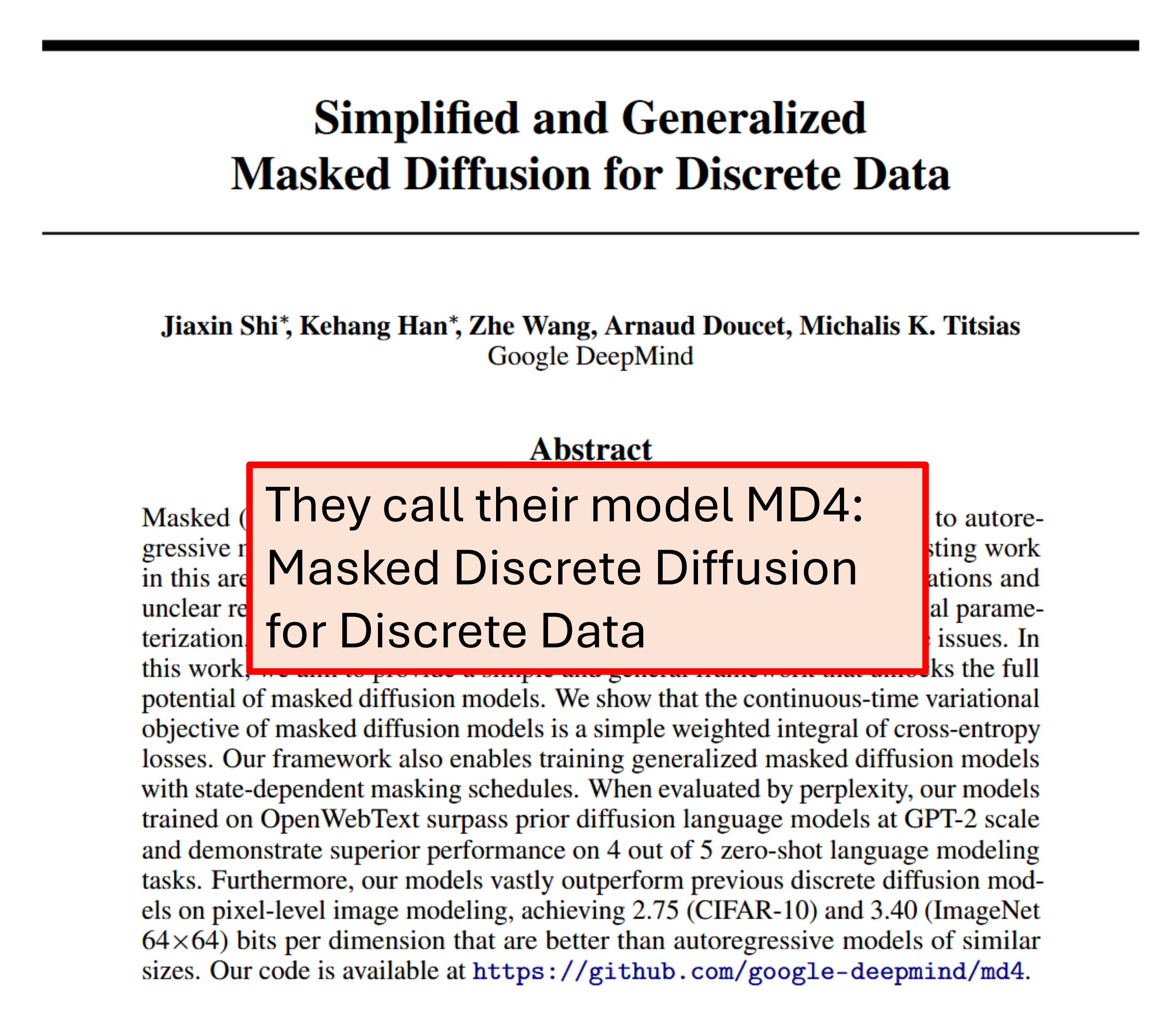


Simplified and Generalized Masked Diffusion for Discrete Data

Jiaxin Shi*, Kehang Han*, Zhe Wang, Arnaud Doucet, Michalis K. Titsias
Google DeepMind

Abstract

Masked (to autore-
gressive r sting work
in this are ations and
unclear re al parame-
terization issues. In
this work, ks the full
potential of masked diffusion models. We show that the continuous-time variational objective of masked diffusion models is a simple weighted integral of cross-entropy losses. Our framework also enables training generalized masked diffusion models with state-dependent masking schedules. When evaluated by perplexity, our models trained on OpenWebText surpass prior diffusion language models at GPT-2 scale and demonstrate superior performance on 4 out of 5 zero-shot language modeling tasks. Furthermore, our models vastly outperform previous discrete diffusion models on pixel-level image modeling, achieving 2.75 (CIFAR-10) and 3.40 (ImageNet 64×64) bits per dimension that are better than autoregressive models of similar sizes. Our code is available at https://github.com/google-deepmind/md4.

They call their model MD4: Masked Discrete Diffusion for Discrete Data

Paper: https://arxiv.org/pdf/2406.04329

Code: https://github.com/google-deepmind/md4

Motivation: Why do we need Discrete Diffusion?

AI currently handles two fundamentally different data types

Standard (Gaussian) Diffusion	**The Discrete Challenge**
Used for: Continuous data (Images, Audio)	**Targeting:** Discrete data (Text, Code, DNA sequences)
Mechanism: Add Gaussian noise.	**Constraint:** Data is categorical.
Why it works there: An image pixel is continuous data. A "slightly noisy pixel" is still a valid concept.	**The Problem:** There is no such thing as "half a word" or "Word A + 0.1 * Word B".
The Failure Mode on Text: Adding Gaussian noise to the token embedding for "Cat" creates a nonsense vector that does not map to *any* word in the vocabulary dictionary.	**The Goal:** We want the benefits of diffusion (iterative refinement, holistic/non-causal generation) but mathematically adapted specifically for a discrete vocabulary.

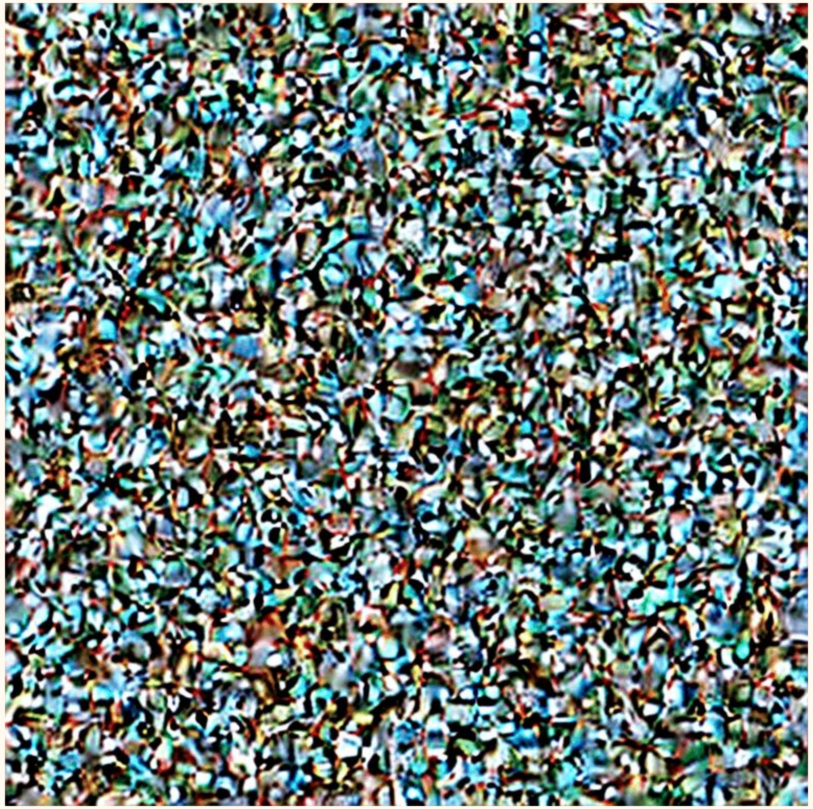
Solution: Instead of noise, we add masks!

What does discrete diffusion look like?

Starting from a fully masked state (Discrete)

https://github.com/google-deepmind/md4

Starting from noise (continuous)

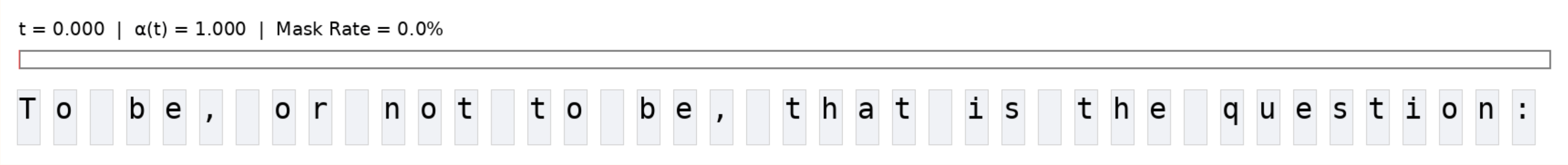


https://github.com/alen-smajic/Stable-Diffusion-Latent-Space-Explorer

The simplified masked Forward Process (1/2)

- **$t = 0$ (No noise), $t = 1$ (All masks), x_0 (original data)**
- **The General Framework (D3PM*):** Transition Matrix Q_t representing the $P(i \rightarrow j)$ of a token transitioning from state i to state j.
- **The Problem:** Calculating a full Q_t needs K^2entries. For large vocabulary sizes (e.g. $K = 50\mathrm{k}$) this becomes prohibitive.

- **The Simplification - Absorbing State**.
- **The Rule:** A token has only two possible states:
  - **Stay x_0:** It remains the original token.
  - **Become [MASK]:** It is corrupted into a special mask token.
- **Absorbing Property:** Once a token becomes [MASK], it *stays* [MASK] for the rest of the forward process.



t = 0.000 | α(t) = 1.000 | Mask Rate = 0.0%

To be, or not to be, that is the question:

*D3PM paper: https://arxiv.org/pdf/2107.03006

The simplified masked Forward Process (2/2)

- $\boldsymbol{t = 0}$ **(No noise),** $\boldsymbol{t = 1}$ **(All masks),** $\boldsymbol{x_0}$ **(original data)**

The Marginal Distribution simplifies into a direct formula for the state at time t:

$$q(x_t|x_0) = \alpha_t \cdot \delta_{x_0} + (1 - \alpha_t) \cdot \delta_{\text{Mask}}$$

Interpretation:

- At any time t, the token is either the original x_0 (with prob α_t) or the [MASK] (with prob $1 - \alpha_t$).
- α_t is a schedule that we can choose. $t = 1 \ \rightarrow \ \alpha_t = 0$

The final Forward Process: This easily gives us now the formula for any two times $\boldsymbol{0 \leq s \leq t \leq 1}$**:**

$$q(x_t|x_s) = \frac{\alpha_t}{\alpha_s} \cdot \delta_{x_s} + (1 - \frac{\alpha_t}{\alpha_s}) \cdot \delta_{\text{Mask}}$$

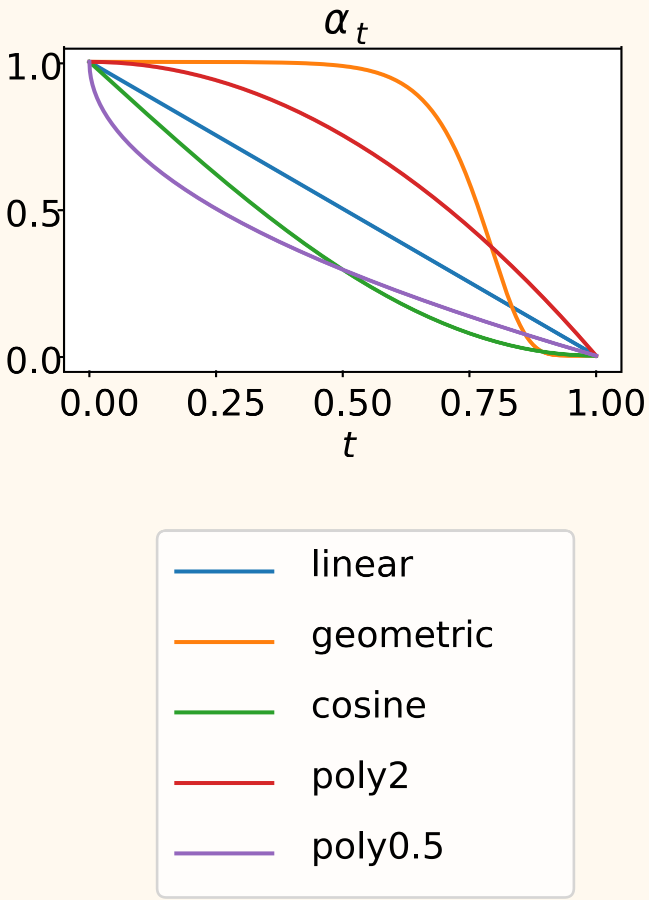


Image from: https://github.com/google-deepmind/md4

The Reverse Process (Posterior)

- **$t = 0$ (No noise), $t = 1$ (All masks), x_0 (original data), $0 \leq s \leq t \leq 1$**

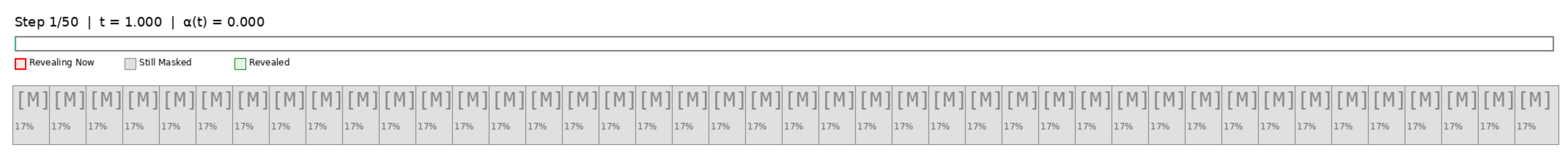
We look at the posterior $q(x_s | x_t, x_0)$.

“Absorbing” in the forward process -> deterministic in the reverse process:

1. If x_t is unmasked, then $x_s = x_t$. (Masks cannot appear out of nowhere in the backward.)
2. If $x_t$is [MASK]: We have a chance to reveal the token x_0.

Probability for a token to get revealed in this specific step $t \to s$:

$$P(\text{reveal}) = \frac{\alpha_s - \alpha_t}{1 - \alpha_t}$$



Step 1/50 | t = 1.000 | α(t) = 0.000

🟥 Revealing Now ▨ Still Masked 🟩 Revealed

[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]
17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%

The model & The Loss

Enter the Neural Network ($\boldsymbol{\mu_\theta}$):

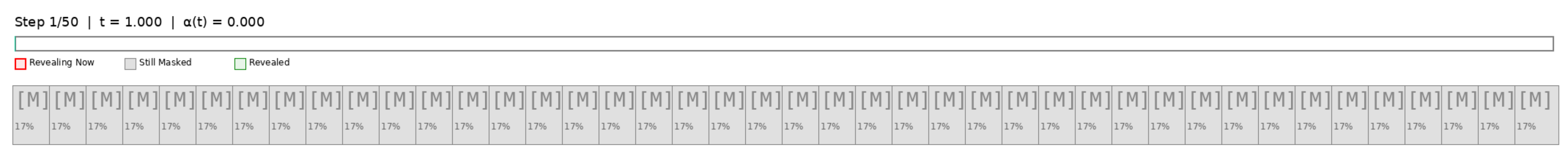
- During generation, we obviously don't know $\boldsymbol{x_0}$.
- Predicted categorical probabilities for the original token $\boldsymbol{\mu_\theta(x_t, t) \approx P(x_0)}$.
- **The Reverse Transition $\boldsymbol{q_\theta(x_s | x_t, \hat{x}_0)}$**: We take the formula for the posterior (previous slide) and substitute the true $\boldsymbol{x_0}$ with our model‘s prediction $\boldsymbol{\hat{x}_0}$

Final Objective:

$$\boxed{\mathcal{L} = \int_0^1 w(t) \cdot \mathrm{CE}(x_0, \mu_\theta(x_t, t))dt}$$

Weighting: $\boldsymbol{w(t) = \frac{\alpha\prime(t)}{1-\alpha(t)}}$. The more tokens are masked, the less we weight the error.

Training: We sample a **random $\boldsymbol{t}$**, mask the input accordingly, and compute the loss.



Step 1/50 | t = 1.000 | α(t) = 0.000

□ Revealing Now   □ Still Masked   □ Revealed

[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]	[M]
17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%

Why Cross Entropy?

The key idea: Binary rule - *Stay Original* or *Become Mask*.

ELBO: minimize KL-Div between the true reverse step $\boldsymbol{q(x_{t-1}|x_t, x_0)}$ and the model's approximated step $\boldsymbol{p_\theta(x_{t-1}|x_t)}$

In masked diffusion, the "true" reverse step is effectively deterministic: if a token unmasks, it **must** reveal the specific ground truth token $\boldsymbol{x_0}$.

The model is trained to predict the clean data $\boldsymbol{\hat{x}_0}$ directly, rather than predicting the slightly less noisy state $\boldsymbol{x_{t-1}}$, aka “classification”.

The Simplification: The KL divergence between a deterministic distribution (the one-hot ground truth $\boldsymbol{x_0}$) and a predicted probability distribution is, by definition, Cross Entropy.

$$\boldsymbol{D_{\mathrm{KL}}(\mathrm{OneHot}(x_0)||p_\theta(x)) = -\sum \mathbf{1}_{x=x_0} \log\big(p_\theta(x)\big) = \mathrm{CrossEntropy}}$$

Summary

Forward Process: $$\boxed{q(x_t|x_s) = \frac{\alpha_t}{\alpha_s} \cdot \delta_{x_s} + (1 - \frac{\alpha_t}{\alpha_s}) \cdot \delta_{Mask}}$$

Reverse Process: $$\boxed{q(x_s|x_t, x_0) \approx q_\theta(x_s|x_t, \hat{x}_0) \ = q_\theta(x_s|x_t, \mu_\theta(x_t, t))}$$

$$\boxed{P(\text{reveal}) = \frac{\alpha_s - \alpha_t}{1 - \alpha_t}}$$

α_t is a masking schedule chosen by us

From 1D to Multidimensional:

For e.g. text “multidimensional" refers to the sequence of tokens

We assume the forward / reverse process is independent for each token.

-> Simply vectorize the equations.

Note: The tokens are mathematically independent during noising, but the Neural Network mixes their information (via Self-Attention) during denoising.

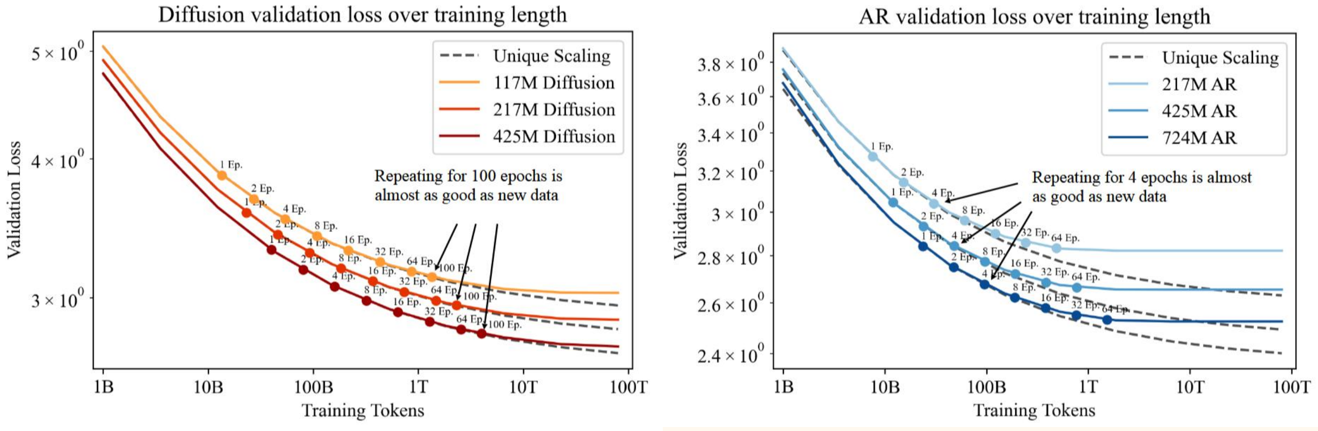
State dependent masking schedules (GenMD4)

- Once a token is unmasked, it is frozen -> Commitment risk!
- Standard MD4 unmasks all tokens at the same speed

Aspect	**Scalar Masking (MD4)**	**Vector Masking (GenMD4)**
Core idea	One global noise schedule	One schedule per vocabulary item
α **definition**	$\alpha(t)$	$\alpha_i(t)$
Crucial formula	$\alpha(t) = (1 - 2\epsilon)f(t) + \varepsilon$	$\alpha_i(t) = 1 - (1 - \epsilon)t^{w_i}$
Learnable params	Model Params	Model Params + Vocab size
Schedule shape	Fixed (cosine / linear / poly)	Fixed family (poly), **learned speed**
Reduces to MD4	-	If all w_i were equal

Masked Diffusion vs Autoregressive Models

- Can be better or worse compared to AR
- Recent paper showed that:
  1. Masked diffusion models are more data efficient
  2. But, need longer training to achieve the same loss
- Data constrained? -> diffusion
- Compute constrained? -> AR

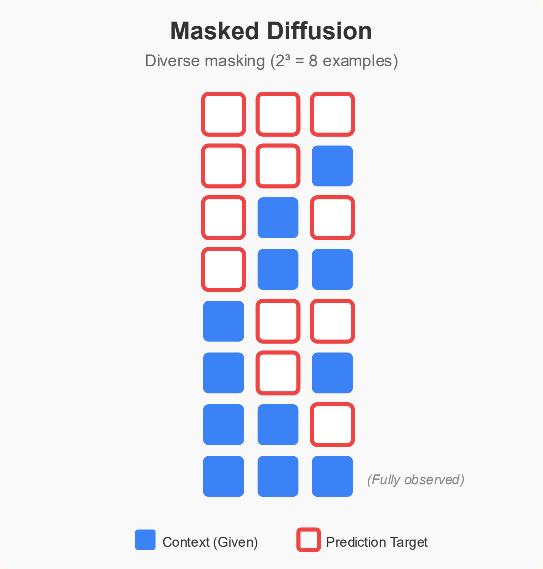


https://arxiv.org/pdf/2507.15857v1

+ Diffusion "sees and manipulates" across the whole sequence.

+ Thus, particularly good for tasks where this is useful (e.g., coding)

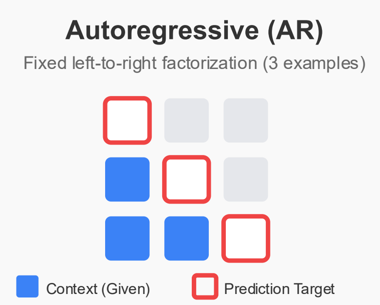
+ Faster for inference. You can pick the n steps.



Masked Diffusion

Diverse masking (2^3 = 8 examples)

🟦 Context (Given) 🟥 Prediction Target



Autoregressive (AR)

Fixed left-to-right factorization (3 examples)

🟦 Context (Given) 🟥 Prediction Target

Inception Labs

TDLM - „Tiny Discrete Language Model“

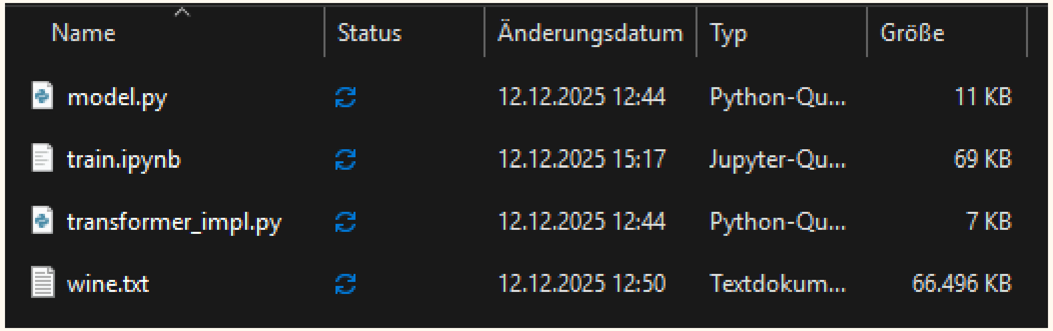
My project: A tiny language model

- Original code in Jax (github.com/google-deepmind/md4)
- Reimplemented in PyTorch from scratch

250 lines ->

160 lines ->

150 lines ->



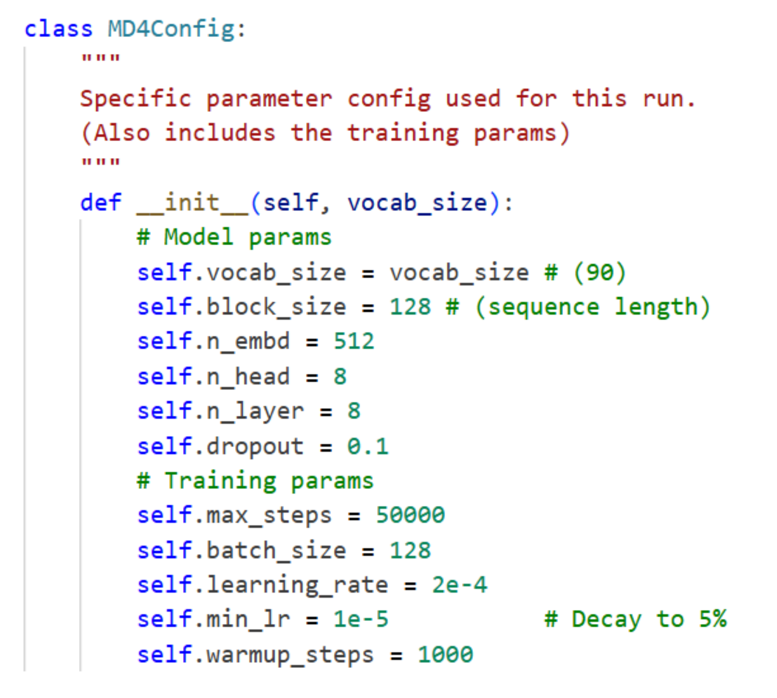
Name	Status	Änderungsdatum	Typ	Größe
model.py	⟳	12.12.2025 12:44	Python-Qu...	11 KB
train.ipynb	⟳	12.12.2025 15:17	Jupyter-Qu...	69 KB
transformer_impl.py	⟳	12.12.2025 12:44	Python-Qu...	7 KB
wine.txt	⟳	12.12.2025 12:50	Textdokum...	66.496 KB

Model:

- 40M parameter transformer model
- Character level (vocab size is 90)

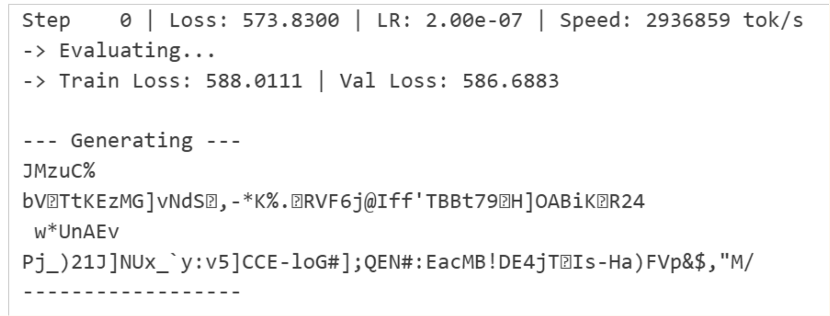
Training:

- Dataset: Wine review, 60M tokens (https://labelyourdata.com/datasets/wine-review-dataset)
- Trained for 800M tokens on (12 epochs)
- Training finished in 1h

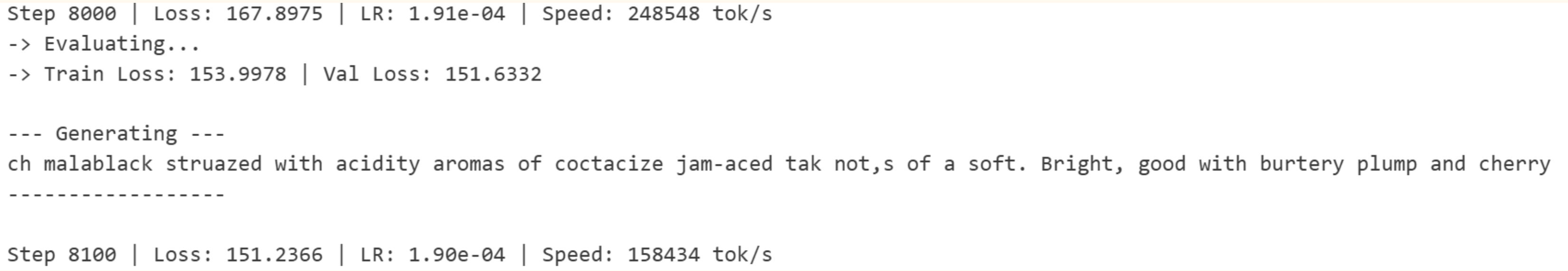


```python
class MD4Config:
    """
    Specific parameter config used for this run.
    (Also includes the training params)
    """
    def __init__(self, vocab_size):
        # Model params
        self.vocab_size = vocab_size # (90)
        self.block_size = 128 # (sequence length)
        self.n_embd = 512
        self.n_head = 8
        self.n_layer = 8
        self.dropout = 0.1
        # Training params
        self.max_steps = 50000
        self.batch_size = 128
        self.learning_rate = 2e-4
        self.min_lr = 1e-5          # Decay to 5%
        self.warmup_steps = 1000
```

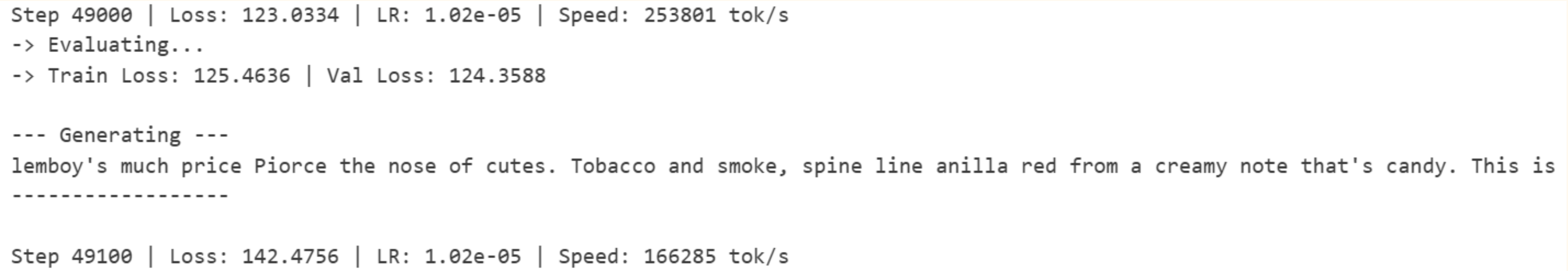
TDLM - Training Progression



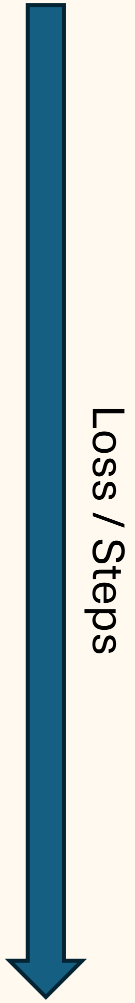
```
Step    0 | Loss: 573.8300 | LR: 2.00e-07 | Speed: 2936859 tok/s
-> Evaluating...
-> Train Loss: 588.0111 | Val Loss: 586.6883

--- Generating ---
JMzuC%
bV⍰TtKEzMG]vNdS⍰,-*K%.⍰RVF6j@Iff'TBBt79⍰H]OABiK⍰R24
 w*UnAEv
Pj_)21J]NUx_`y:v5]CCE-loG#];QEN#:EacMB!DE4jT⍰Is-Ha)FVp&$,"M/
------------------
```



```
Step 8000 | Loss: 167.8975 | LR: 1.91e-04 | Speed: 248548 tok/s
-> Evaluating...
-> Train Loss: 153.9978 | Val Loss: 151.6332

--- Generating ---
ch malablack struazed with acidity aromas of coctacize jam-aced tak not,s of a soft. Bright, good with burtery plump and cherry
------------------

Step 8100 | Loss: 151.2366 | LR: 1.90e-04 | Speed: 158434 tok/s
```



```
Step 49000 | Loss: 123.0334 | LR: 1.02e-05 | Speed: 253801 tok/s
-> Evaluating...
-> Train Loss: 125.4636 | Val Loss: 124.3588

--- Generating ---
lemboy's much price Piorce the nose of cutes. Tobacco and smoke, spine line anilla red from a creamy note that's candy. This is
------------------

Step 49100 | Loss: 142.4756 | LR: 1.02e-05 | Speed: 166285 tok/s
```



Loss / Steps

Now it knows fancy words ☺