

🍏PoM: Efficient Image and Video Generation with the Polynomial Mixer

David Picard¹, Nicolas Dufour^{1,2}

¹LIGM, École Nationale des Ponts et Chaussées, IP Paris, Univ Gustave Eiffel, CNRS, France

²LIX, École Polytechnique, IP Paris, CNRS, France

{david.picard, nicolas.dufour}@enpc.fr

Abstract

Diffusion models based on Multi-Head Attention (MHA) have become ubiquitous to generate high quality images and videos. However, encoding an image or a video as a sequence of patches results in costly attention patterns, as the requirements both in terms of memory and compute grow quadratically. To alleviate this problem, we propose a drop-in replacement for MHA called the Polynomial Mixer (PoM) that has the benefit of encoding the entire sequence into an explicit state. PoM has a linear complexity with respect to the number of tokens. This explicit state also allows us to generate frames in a sequential fashion, minimizing memory and compute requirement, while still being able to train in parallel. We show the Polynomial Mixer is a universal sequence-to-sequence approximator, just like regular MHA. We adapt several Diffusion Transformers (DiT) for generating images and videos with PoM replacing MHA, and we obtain high quality samples while using less computational resources. The code is available at <https://github.com/davidpicard/HoMM>.

1. Introduction

In a sudden change of pace, high quality image and video generation have evolved from a task seemingly impossible to achieve to a task almost solved by available commercial or open-source tools like Stable Diffusion 3 [19], Sora [7] or MovieGen [61]. At the heart of this success lies the Multi-head Attention (MHA) in the transformer architecture [72] that has excellent scaling properties [58, 82]. These so-called scaling laws [44] enable *brute-forcing* complex problems such as image and video generation by using very large models trained on gigantic data, at the expense of an ever increasing computational cost. The main focus of current research lies thus in scaling transformer-based approaches to larger models handling larger datasets.

The issue with transformers is that the computational cost increases quadratically with the sequence length due to the pairwise computation in MHA. This means that gener-

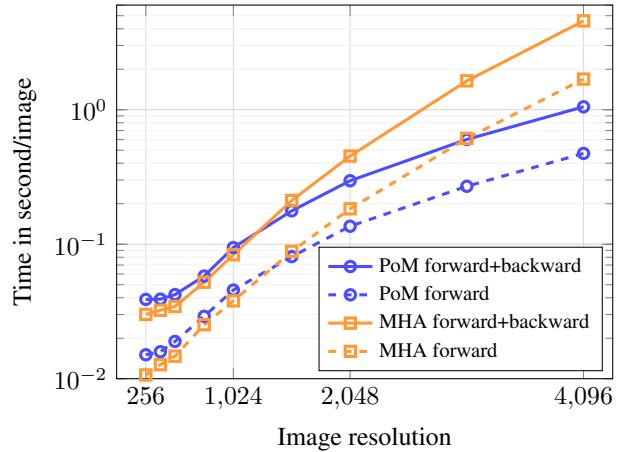


Figure 1. **Comparison between the speed of PoM and Multi-Head Attention (MHA) in the same DiT-XL/2 architecture for different image resolutions.** We use an H100 GPU and compute the average time on 100 synthetic training batches to perform the forward or forward+backward passes. We use synthetic data to remove the influence from data loading. Training with PoM is less costly than inference with MHA at higher resolutions.

ating an image at twice the spatial resolution (respectively a video at twice the resolution and double the duration) results in 4 times more patches and thus 16 times more computational cost (respectively 8 times more patches and thus 64 times more computational cost). Attempts at having transformers with sub-quadratic complexity [11, 47, 76] introduce the additional constraint of fixing the number of tokens, which prevents generating images or videos of different sizes. Alternatively, recurrent models such as State-Space Models (SSM) [26, 27] have been investigated for the task [38, 69, 79] since their complexity is linear with the sequence length [25]. However, they introduce an arbitrary causal raster scan of the sequence that does not fit the 2D geometry of images very well.

In this paper, we enable better scaling in large generative models by introducing a new building block called the Polynomial Mixer (PoM). PoM has a linear complexity

like SSMs while still enabling all pairwise information to be processed like in MHA, obtaining effectively the best of both worlds. From a theoretical standpoint, we prove PoM can be used as a drop-in replacement for attention. Doing so in the popular DiT architecture [56, 58] results in improved scaling such that at higher resolutions, it becomes less costly to train a model with PoM than to perform inference with a model using MHA, as shown on Figure 5.

To sum up, the contributions of this paper are the following:

- ✓ We introduce the Polynomial Mixer (PoM), a replacement for MHA that has a linear complexity with respect to the sequence length and without sacrificing generation quality;
- ✓ We prove that models equipped with PoM are universal sequence-to-sequence approximators;
- ✓ We train DiT-inspired image generative models and obtain results of similar quality while being much more compute efficient at higher resolutions;
- ✓ We train video generative models leveraging PoM with a constant processing cost per frame while not sacrificing on visual quality.

Our contribution is therefore primarily fundamental: We show that it is possible to train generative models with an alternative mechanism to MHA. We believe this direction will not only ground future research on high resolution images and very long videos generation, but also could benefit many areas of research (*e.g.*, large language models, vision-language models, etc).

2. Related Work

Diffusion Diffusion models [35, 57, 67] learn a neural operator that produces natural images from noise using a forward-reverse set of processes. The forward process consists in pushing the distribution of natural images forward to a known distribution, typically Gaussian, which can be done by adding increasing level of noise to the image. The reverse process does not have an explicit solution, but can be approximated by a neural network by regressing the local inverse of the forward process, *i.e.*, solving

$$\min_{\theta} \mathbb{E}_{t \sim \mathcal{U}(0,1)} [\|\varepsilon_t - f_{\theta}(x_t, t)\|^2], \quad (1)$$

$$\text{s.t. } x_t = \alpha_t x_0 + \gamma_t \varepsilon_t, \varepsilon_t \sim \mathcal{N}(0, 1). \quad (2)$$

Here, α_t and γ_t are chosen such that x_0 corresponds to a natural image whereas x_1 corresponds to pure Gaussian noise. A great amount of research has been put into finding better noise schedules (α_t and γ_t) [4, 31, 45], or improving the quantity that is regressed [51, 52, 64], keeping the general idea of learning to invert step by step the stochastic differential equation that transforms an image into noise.

For image and generation, most efforts have been poured into designing efficient architectures at the task. While

the original DDPM papers [35, 57] sample images in pixel space, making it unsuitable for large resolution, the most groundbreaking improvement was introduced by Stable Diffusion [63] with the addition of a variational auto-encoder (VAE) that allows the diffusion process to be performed in a lower dimensional latent space. Stable Diffusion uses a U-Net architecture complemented by attention layers [63, 65]. To benefit more from the scaling properties of transformers [44, 82], simpler approaches based solely on transformer layers has been proposed in DiT [58] and the subsequent flow-matching version SiT [56]. Most modern text-to-image generation models are now based on Transformer layers rather than the U-Net [9, 19, 20, 32], [12, 28], train efficient pixel space transformers models by leveraging multiscale training and SwinAttention. Similarly, RIN [10, 40] also proposes an approach using attention only, albeit in a Perceiver-IO [42] inspired architecture that uses cross-attention to perform most of the computation in a smaller latent space, and has been successfully extended to text-to-image [18]. In addition to architectures and sampling [2, 84, 86], the importance of training is also highlighted in recent works, from resampling the training data [24, 54] to RL [49, 74, 78] and model averaging [46].

In video generation [29, 36, 66, 73, 83], early attempts have focused on extending existing text-to-image models to benefit from their large scale pretraining [5, 21, 22, 34, 37, 48]. However, the drawback of such approaches is that they re-use the VAE of existing text-to-image models which does not encode temporal information, which is thus not compressed. As such, novel architectures using a 2D+t VAE such as CogVideoX [80], PyramidFlow [43] can benefit from a smaller latent space leading to less computational costs.

Fast alternative to attention Since the introduction of Transformers [72], many effort have been made to reduce the quadratic complexity of MHA [11, 47, 76]. Notably, methods like Reformer [47] use fast approximate neighbors to reduce the size of the attention matrix based on the assumption that most tokens will have zero attention. To go further, Linformer [76] proposes to compute an explicit low rank projection of the keys and the values to reduce the complexity of MHA for each query from the size of the sequence n to an arbitrary chosen number $k \ll n$. The main drawback of such approach is that n and k are fixed, which means that the model can no longer process sequences of varying length. With the advent of Large Language Models and their ability to process extremely long sequences [1, 17, 68], recent efforts have been put on more efficient implementations such as Flash-Attention [13, 14] or KV-cache [6, 55] which seem sufficient for text. However for visual content, the sequence length grows quadratically with the resolution, which, because MHA is also quadratic

in the number of tokens, leads to quartic computational and memory complexity.

Alternatively, some attempts have been made to just remove the Multi-Head Attention, such as in Mlp-Mixer [70] and Resmlp [71] that replace MHA with simple projection on the transpose tensor (*i.e.*, considering the sequence dimension as the features). These approaches have been shown to obtain competitive results, but similarly to Linformer, they imply a fix sequence length since this length is now an intrinsic dimension of the projection in the transpose direction. More recently, State-Space Models (SSM) [26, 27] have become the focus of recent work especially in language modeling [15, 23, 50, 88]. SSM are recurrent models, which is highly beneficial for language modeling because of the causal property of text. In that case, the complexity to generate the next token becomes constant. In visual content however, there is no such natural causality pattern in the spatial dimensions. Attempt to use such models for vision tasks have been successful [53, 59, 87], albeit at the cost of enforcing an arbitrary 1-dimensional scan order of the tokens that does not encode well the 2D nature of an image. In image generation using diffusion [38, 79], since the model has to be iterated, this results in a doubly sequential processing (space and iterations) that does not benefit from the parallel nature of processing images. For video however, the causal aspect is natural over the time dimension, and recurrent approaches may be more efficient.

3. Polynomial Mixer and Polymorpher

We define a *Polymorpher* block as a sequence-to-sequence function mapping $\mathbb{R}^{d \times n}$ to $\mathbb{R}^{d \times n}$, composed of two residual blocks, a *Polynomial Mixer* and a feed-forward block.

For a sequence $X \in \mathbb{R}^{d \times n}$, the Polynomial Mixer (PoM) shown on Figure 2 is defined as follows:

$$\text{PoM}(X) = W_o [\sigma(W_s X) \circ H(X) \mathbf{1}^\top], \quad (3)$$

$$H(X) = \left[h(W_1 X); \dots; \prod_{m=1}^k h(W_m X) \right] \mathbf{1}, \quad (4)$$

where k is the degree of the Polynomial Mixer, σ is the sigmoid function, h an activation function, \circ and \prod the element-wise (Hadamard) product, and $\mathbf{1}$ a vector of the appropriate dimension filled with ones. The notation $[\cdot ; \cdot]$ is for vertical concatenation. The matrices $W_o \in \mathbb{R}^{d \times kD}$, $W_s \in \mathbb{R}^{kD \times d}$ and $W_1, \dots, W_k \in \mathbb{R}^{D \times d}$ are the learnable parameters of the Polynomial Mixer.

The idea of the Polynomial Mixer is to that the sequence $X \in \mathbb{R}^{d \times n}$ is uniquely summarized into the representation $H(X) \in \mathbb{R}^{kD \times 1}$. Each element in X then gets to query $H(X)$ independently thanks to the map $S(W_s X) \in \mathbb{R}^{kD \times n}$. The queried information is then projected back into the original space with W_o .

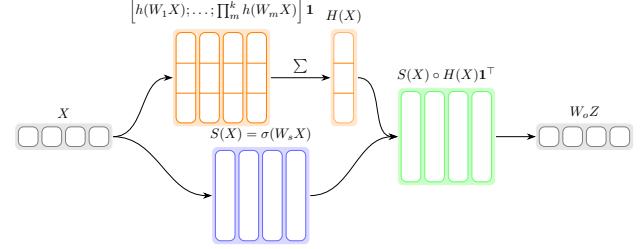


Figure 2. **Diagram for the Polynomial Mixer.** The input sequence is split into two paths. The top path expands each token using a polynomial before they are mixed (averaged)² into a single representation. The bottom path expands the tokens into gating coefficients. Both paths are recombined and projected back into the input dimension.

Contrarily to MHA that computes all pairwise exchanges of information between tokens in the sequence, the Polynomial Mixer follows a state-representation ($H(X)$) approach where all information is shared in a common memory location that all tokens can access. This state-representation is defined by mixing all tokens of the sequence after they are mapped to a high dimensional space by a learned polynomial, hence the name *Polynomial Mixer*, and a similar approach has been successfully used for learning image representation [41]. The main benefit is that the complexity of the approach is no longer quadratic but linear with the sequence length n .

Taking inspiration from transformers with MHA, we define a Polymorpher block P as alternating residual Polynomial Mixers with feed-forward networks as follows:

$$P(X) = X + \text{PoM}(X) + \text{FF}(X + \text{PoM}(X)), \quad (5)$$

with $\text{FF}(X)$ being a two-layer feed-forward network.

A Polymorpher is a drop-in replacement for any Transformer-based architecture as it performs the same role of sequence-to-sequence mapping. The main difference is in its parametrization: A Transformer is configured by the number of heads and their dimension in MHA, whereas the Polymorpher is configured by its degree k and the dimension D of each polynomial.

3.1. Polymorpher for causal sequences

A causal sequence can easily be modeled in PoM by adding a mask M that prevents summing future tokens into the blackboard. This corresponds to the following definition

$$\text{PoM}(X, M) = W_o [\sigma(W_s X) \circ H(X)], \quad (6)$$

$$H(X) = \left[h(W_1 X); \dots; \prod_{m=1}^k h(W_m X) \right] M^\top. \quad (7)$$

Now $H(X) \in \mathbb{R}^{kD \times n}$ and $M \in \{0, 1\}^{n \times n}$ is a binary matrix that defines which pairs of tokens are related. Just

like for MHA, a binary matrix defines an attention pattern that can be arbitrarily chosen.

In the special case of causal sequences, M is a lower triangular matrix. Moreover, one can express the mixing part of the Polynomial Mixer as an iterative process as follows:

$$H(X)_{:,i} = \sum_{j \leq i} \left[h(W_1 X); \dots; \prod_{m=1}^k h(W_m X) \right]_{:,j}, \quad (8)$$

$$= H(X)_{:,i-1} + \left[h(W_1 X); \dots; \prod_{m=1}^k h(W_m X) \right]_{:,i}. \quad (9)$$

In this formula, $H(X)_{:,i}$ is an explicit hidden state that is updated by adding the polynomial mapping of the next token. Such a configuration enables $\mathcal{O}(1)$ inference complexity in the auto-regressive setup, a property that is shared with recurrent networks, but not transformers. Like SSMs, Polymorphers have the best of both worlds, they can train on the whole sequence in parallel and do the inference in the recursive way.

In addition, Polymorphers can handle block causal sequences. Let M be a block causal matrix for some integer block size K :

$$M_{i,j} = 1 \text{ if } j \leq \lceil i/K \rceil K \text{ else } 0. \quad (10)$$

We can now rewrite H as

$$\begin{aligned} H(X)_{:,i} &= H(X)_{:,\lfloor i/K \rfloor K} \\ &+ \sum_{j=\lfloor i/K \rfloor K}^{\lceil i/K \rceil K} \left[h(W_1 X); \dots; \prod_{m=1}^k h(W_m X) \right]_{:,j}. \end{aligned} \quad (11)$$

In this configuration, we can sequentially process groups of tokens at a time during inference, which reduces the memory requirement. This is in particular practical for video sequences where it makes sense to have a causal mask in the temporal dimension that makes each frame depend on the previous ones, while keeping the ability of all the tokens (patches) of a frame to look at each others, since causality does not have much sense in the spatial dimension.

3.2. Theoretical analysis

We first show that PoM is equivariant, which means that permutations in the input sequence result in permuted outputs. This is a key property that made transformers popular and does not hold for other architectures like convolutions:

Proposition 1 (Permutation equivariance). *A Polynomial Mixer is permutation equivariant, i.e., let $X \in \mathbb{R}^{d \times n}$ be a set of vectors and P a column permutation matrix, then $\text{PoM}(XP) = \text{PoM}(X)P$.*

Proof. For a permutation P , we have

$$\text{PoM}(XP) = W_o [\sigma(W_s XP) \circ H(XP) \mathbf{1}^\top]. \quad (12)$$

Notice that $H(XP) = H(X)$ because the sum is permutation invariant, and $\sigma(W_s XP) = \sigma(W_s X)P$ because σ is an element-wise operation. Noticing that $H(X)\mathbf{1}^\top$ has all identical columns allows us to move P outside of the brackets to conclude the proof. \square

More importantly, we can also prove a universal approximation theorem for Polymorphers similar to what is well known for Transformers [81]. As the polynomial mixer is equivariant, it requires the use of positional encoding, which also underlines the similarity between PoM and MHA.

We use the following standard definition of distance between functions that map sequences to sequences. Given two functions f and $g : \mathbb{R}^{d_n} \rightarrow \mathbb{R}^{d_n}$ and an integer $A \leq p \leq \infty$, we define the distance d_p as:

$$d_p(f, g) = \left(\int \|f(X) - g(X)\|_p^p dX \right)^{1/p}. \quad (13)$$

The following theorem holds:

Theorem 2 (Universal approximation). *Let $1 \leq p \leq \infty$ and $\epsilon > 0$, then for any given $f \in \mathcal{F}$ the set of continuous functions that map a compact domain in $\mathbb{R}^{d \times n}$ to $\mathbb{R}^{d \times n}$, there exists a Polymorpher g with learned positional encoding such that $d_p(f, g) \leq \epsilon$.*

The proof follows exactly the same scheme as in [81], where most of the heavy lifting is done by the feed-forward networks. Their main argument is to show that MHA can map every token in the sequence to a unique value that depends on the entire sequence, and then the feed-forward blocks can map those unique values to the desired output. In our case, we just have to ensure that the Polynomial Mixer has the same properties as MHA, which is obtained using the following lemma:

Lemma 3 (Contextual mapping (informal)). *There exists $k > 0$ for which any Polynomial Mixer q of degree k is a contextual mappings on $\mathbb{R}^{d \times n}$, that is:*

- For any $X \in \mathbb{R}^{d \times n}$ with different entries, $q(X)$ has different entries.
- For any $X, X' \in \mathbb{R}^{d \times n}$ that differ at least by one element, then all entries of $q(L)$ and $q(L')$ are different.

The proof is deferred to the appendix and primarily uses the fact that a sufficiently high degree polynomial is uniquely defined by a sequence of point-wise evaluation. As noted in [81], having the contextual mapping property is not so common as it requires to summarize uniquely the context while preserving the identity of the current token.

With these results, we show that a Polymorpher is as potent as a Transformer for sequence modeling.

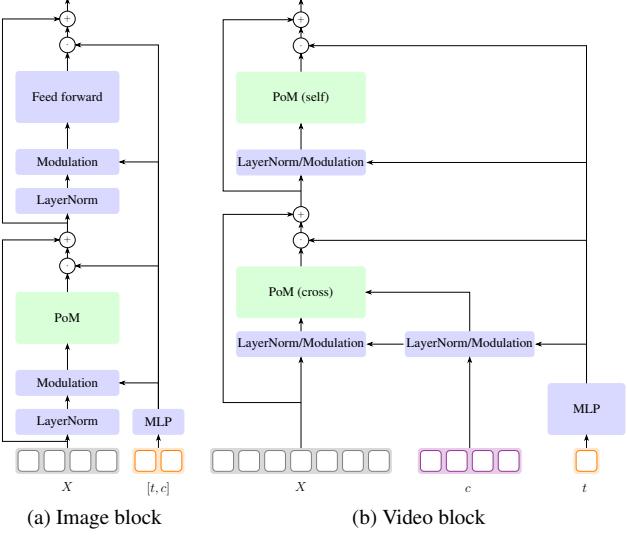


Figure 3. Building blocks for our diffusion models using PoM. For class-conditional image generation (a), we follow strictly DiT[58] in the AdaLN variant, replacing multi-head attention with PoM. For text to video generation (b), we follow a hybrid approach in which the encoded text tokens are incorporated into the video tokens using PoM instead of cross attention, while the time is used as a modulation. Modulation means component-wise scale and shift modification based on the coefficients predicted by the MLP (similarly to the AdaLN approach).

4. Diffusion with PoM

Armed with the definition of PoM and Polymorphers, we now design diffusion models taking inspiration from models based on MHA, and show that PoM can replace attention in practice. We follow the design choices of DiT [58] and propose a class-conditional image generation polymorpher as well as a text-to-video generation polymorpher.

4.1. Architecture design

Image generation For image generation, the class-conditional polymorpher is similar to the AdaLN variant of DiT. The image is encoded through the VAE of SD1.5 [63] and then features are aggregated into visual tokens X . We add a 2D cosine positional encoding to them before we feed them to the model. The class c and the time step t are embedded using an embedding matrix and a cosine embedding respectively before being summed together.

The model consists in several blocks that combine modulations, PoM and feed forward networks as shown on Figure 3a. In each block, the modulation consists in predicting from the condition $c + t$ a scale γ and a shift β that modify the input by

$$x \leftarrow \gamma(x - \beta). \quad (14)$$

Similarly to DiT, the MLP also predicts gates σ that can

shut down an entire block f thanks to

$$x \leftarrow x + (1 + s)f(x), \quad (15)$$

with the 1 in $1 + s$ being added so that there is a full residual connection when the MLP predicts $f(x) = 0$. For naming the architectures, we follow the same parametrization as in DiT. Namely, an $S/2$ model has a kernel size and stride of 2 for aggregating the VAE features into tokens, and 12 blocks of dimension 384. Similarly, an $XL/2$ model that has 28 blocks of dimension 1152. For the PoM operation inside each block, we use an polynomial of order 2 with an expansion factor of 2 unless specified otherwise. Pytorch code for the blocks is given in appendix.

Video generation For video generation from text, we extend the DiT architecture to handle text as a condition. We first encode video clips using the 3D VAE from CogXVideo [80] and then group the features into visual tokens using a kernel size of $2 \times 2 \times 2$ (with 2×2 for the spatial axes, and 2 for the temporal axis resulting in a downscaling factor of $16 \times 16 \times 8$). We add a 3D cosine positional encoding to the visual tokens before feeding them to the model. The text is encoded using T5 [62] embeddings and the time step is encoded using a cosine embedding.

The model consists in blocks using PoM to aggregate information between the text condition and the visual tokens as shown on Figure 3b. More precisely, a first PoM operation is used in a cross fashion, similar to cross-attention, to aggregate information from the text tokens into the visual tokens. Then, a second PoM operation is used to aggregate information among the visual tokens themselves, similar to what self-attention would do. Finally, a feed forward module processes the visual tokens only. The time step embedding is used in an MLP to predict the coefficients of modulations and gates at each of the operations.

We train a single model of size $XL/2$ that consists in 20 layers of dimension 1152 resulting in 1.1B parameters.

4.2. Training setup

For class-conditional image generation, we train on ImageNet. We rescale each image to 256 pixels on their smallest size and then take a crop of size 256×256 . We use both the original images and horizontally flipped version for a total of 2.4M images. We train a model f_θ either using the diffusion loss:

$$\mathcal{L}_D = E_{t \sim \mathcal{U}[0,1]} \|\varepsilon_t - f_\theta(x_t, c, t)\|^2, \quad (16)$$

or the flow matching loss:

$$\mathcal{L}_{FM} = E_{t \sim \mathcal{U}[0,1]} \|v_t - f_\theta(x_t, c, t)\|^2, \quad (17)$$

with $v_t = \varepsilon_t - x_0$. For each experimental result, we mention which loss is used, but the models are trained similarly

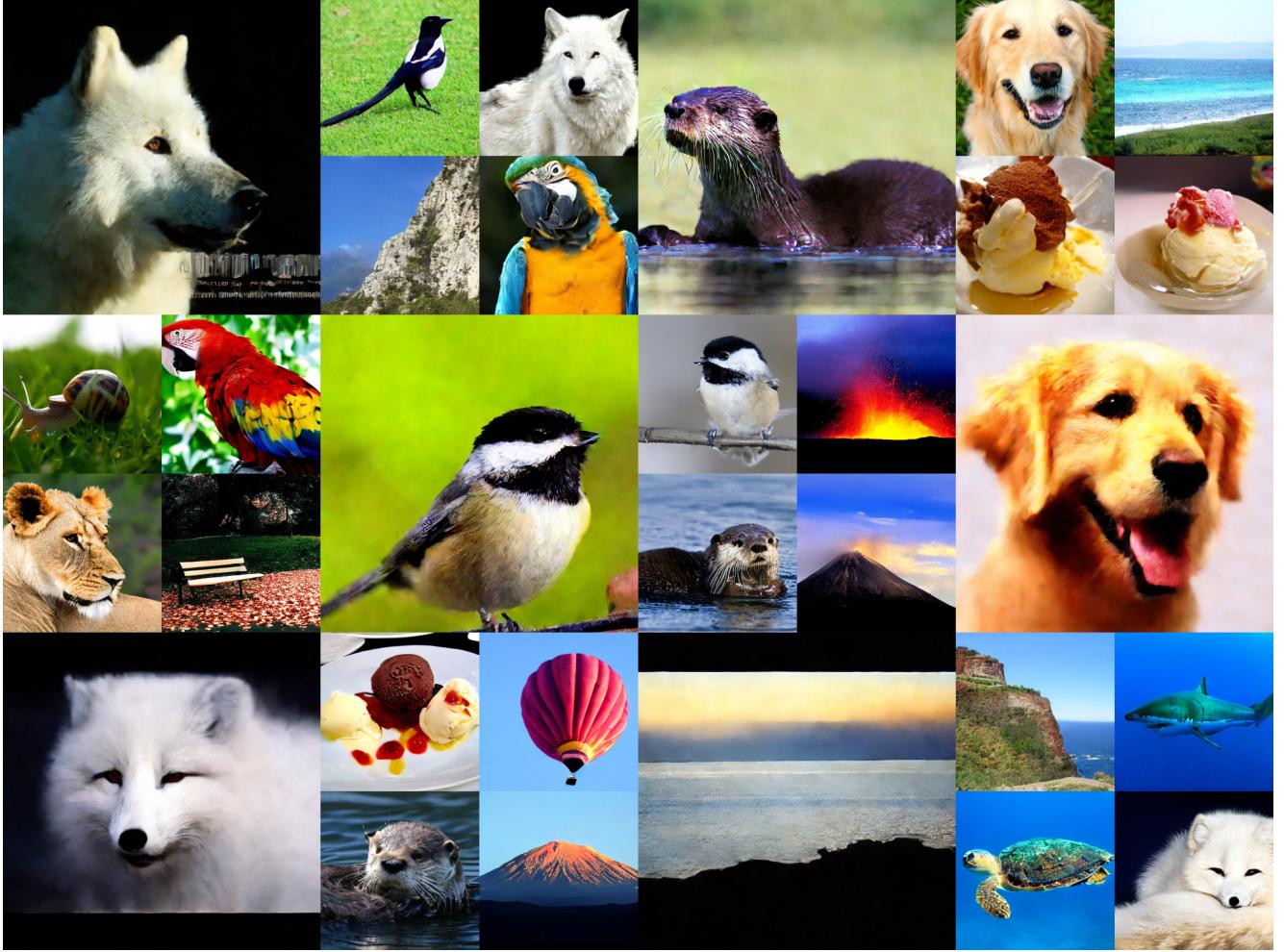


Figure 4. **Qualitative results on class-conditional generation.** We show images sampled with the model DiPoM-XL/2 trained with the flow-matching loss \mathcal{L}_{FM} at several resolutions for different classes. We use classifier-free guidance with $\omega = 4s/s_0$ with s the scale of the image and s_0 the reference scale (256).

without requiring change in training hyper-parameters. We use AdamW with a constant learning rate of 10^{-4} followed by a short cooldown with square root decay [30].

For video, we used WebVid-2M [3] that we rescale to 240×384 at 16 fps. We keep only the first 5 seconds, corresponding to 80 frames. This results in a total of 2.5M clips. We train using the flow matching loss \mathcal{L}_{FM} . We also use AdamW with a constant learning rate of 10^{-4} followed by a short cooldown with square root decay.

5. Experiments

We first show result on class-conditional image generation and then of text-to-video generation.

5.1. Class-conditional image generation

Quantitative results We compare the results of our XL/2 model trained with the diffusion loss to the state of the art on Table 1. We compute the Fréchet Inception Distance (FID), the Inception Score (IS), precision (P) and recall (R) using the code from ADM [16] on 50k generated images. The table is split between methods on masked encoding (Mask-GIT [8]), diffusion models based on SSM and diffusion models based on attention. Results are extracted from the corresponding papers. Our images are generated with 250 steps of the DDIM sampler for the model trained with the diffusion loss \mathcal{L}_{D} , and 125 steps of Heun sampler for the model trained with the flow-matching loss \mathcal{L}_{FM} , with classifier free guidance (CFG, $\omega = 0.7$ in both cases).

Using the evaluation code and reference set from ADM [16], we obtain an FID of 2.46, which is slightly

Model	Sample config	#train	FID↓	IS↑	Precision↑	Recall↑
Mask-GIT [8]			6.18	182.1	0.80	0.51
DIFFUSSM-XL [†] [79]	250 steps DDPM	660M	2.28	259.1	0.86	0.56
DiM-H [†] [69]	25 steps DPM++	480M	2.21	-	-	-
ADM-G [16]	250 steps DDIM	500M	4.59	186.7	0.83	0.53
LDM-4-G [63]	250 steps DDIM	215M	3.60	247.7	0.87	0.48
RIN [40]	1000 steps DDPM	600M	3.42	182.0	-	-
DiT-XL/2 [†] [58]	250 steps DDPM	1.8B	2.27	278.2	0.83	0.57
SiT-XL/2 [†] [56]	125 steps Heun	1.8B	2.15	254.9	0.81	0.60
DiPoM-XL/2 \mathcal{L}_D (ours)	250 steps DDIM	950M	2.46	240.6	0.78	0.60
DiPoM-XL/2 \mathcal{L}_{FM} (ours)	125 steps Heun	950M	3.70	255.2	0.79	0.56

Table 1. **Quantitative results on ImageNet 256 × 256 class-conditional generation.** #train denotes the number of training images seen during train (i.e., batch size × number of training steps). [†] denotes methods evaluated against the Imagenet training set instead of the usual ADM evaluation archive. We color in blue (respectively in red) DiT and PoM architectures of equivalent size that are trained with the same diffusion loss \mathcal{L}_D (respectively flow-matching loss \mathcal{L}_{FM}), and we bold the best values between the two, even though the results are not evaluated against the same reference set.

Degree	Expand	FID↓	IS↑	Precision	Recall
1	12	90.1	15.1	0.27	0.36
2	6	87.0	15.8	0.29	0.37
3	4	86.1	16.0	0.29	0.38
4	3	88.8	15.5	0.28	0.36
6	2	90.7	15.0	0.28	0.36

Table 2. **Comparison of different degrees of Polynomial Mixer** at a constant memory budget with a S/2 model on 10k images from Imagenet. Having a degree ≥ 2 is necessary to get good performances, but there is a trade-off between the degree and the expansion factor.

above that of the comparable DiT architecture, but notice that our model was trained for only half of the number steps of DiT. In addition, we found FID to be very unreliable as a metric, as it is highly varying with the reference set. For example, using the validation set of ImageNet, we obtained 3.45 FID¹. We obtain a slightly lower IS compared to DiT, but this could be improved by using a higher CFG. Indeed, we show in appendix that the trade-off between FID and IS can reach as high as 300 IS at the cost of a much higher FID. We obtain a precision/recall trade-off comparable to DiT, slightly lower on precision but also higher on recall.

Overall, the results obtained using PoM are on par with the literature, showing that PoM can be used as a drop-in replacement for multi-head attention in a neural architecture, without requiring either architectural changes or training hyper-parameter tuning.

Qualitative results We further fine-tune the model on higher resolution data for a small number of steps to ob-

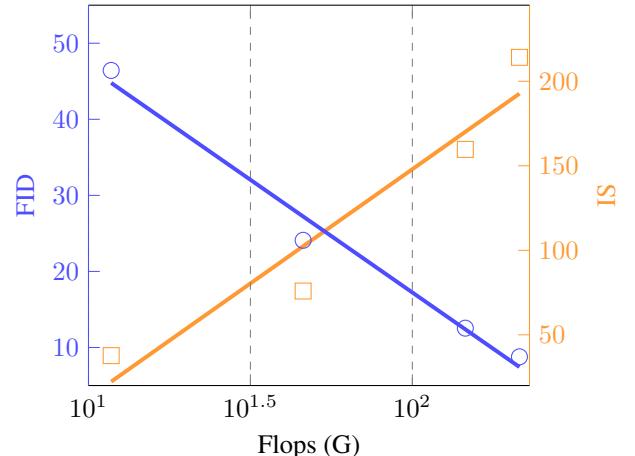


Figure 5. **Scaling laws for a DiT-like architecture with attention replaced by PoM.** FIDs and Inception Scores (IS) are computed on 10k samples with classifier free guidance ($\omega = 1$), and shown with a linear regression in log space. Performances scale with the computation budget, similarly to transformers.

tain a collection of models able to sample images up to 1024 × 1024 resolution (which is the maximum resolution we found reasonable to upscale to on ImageNet). We show selected samples at these higher resolution on Figure 4. At higher resolution, some classes are collapsed due to the lack of available data.

Ablation study We study the impact of the degree of the polynomial on Table 2. To enable a fair comparison, we consider a set of S/2 models that have the same dimension for $H(X)$ and compute different trade-offs between the degree of the polynomials and their dimension. As we can see, having at least second order polynomials is crucial to

¹It was shown in [60] that randomness affects significantly the results.

Models	Subject Consistency	Background Consistency	Temporal Flickering	Motion Smoothness	Dynamic Degree	Aesthetic Quality	Imaging Quality	Object Class
LaVie [77]	91.4%	97.5%	98.3%	96.4%	49.7%	54.9%	61.9%	91.8%
ModeScope [75]	89.9%	95.3%	98.3%	95.8%	66.4%	52.1%	58.6%	82.3%
VideoCrafter [33]	86.2%	92.9%	97.6%	91.8%	89.7%	44.4%	57.2%	87.3%
CogVideo [37]	92.2%	96.2%	97.6%	96.5%	42.2%	38.2%	41.0%	73.4%
V-DiPoM-XL/2 <i>no-mask</i>	90.6%	96.6%	99.7%	97.3%	31.7%	28.6%	47.1%	29.3%
V-DiPoM-XL/2 <i>b-causal</i>	80.4%	92.2%	98.1%	97.4%	37.5%	30.5%	47.9%	30.0%
Models	Multiple Objects	Human Action	Color	Spatial Relationship	Scene	Appearance Style	Temporal Style	Overall Consistency
LaVie [77]	33.3%	96.8%	86.4%	34.1%	52.7%	23.6%	25.9%	26.4%
ModeScope [75]	39/0%	92.4%	81.7%	33.7%	39.3%	23.4%	25.4%	25.8%
VideoCrafter [33]	25.9%	93.0%	78.8%	36.7%	43.4%	21.6%	25.4%	25.2%
CogVideo [37]	18.1%	78.2%	79.6%	18.2%	28.2%	22.0%	7.8%	7.70%
V-DiPoM-XL/2 <i>no-mask</i>	1.9%	21.6%	76.3%	7.6%	2.8%	21.4%	13.4%	15.1%
V-DiPoM-XL/2 <i>b-causal</i>	3.3%	31.0%	69.5%	10.4%	3.0%	21.2%	17.2%	17.3%

Table 3. **Quantitative results on VBench [39].** We compare the same architecture of a V-DiPoM-XL/2 trained with the flow-matching loss \mathcal{L}_{FM} using either no mask (denoted *no-mask*) or block-causal masking (denoted *b-causal*). We report results from the literature taken from [39] to provide some calibration, but noting that the comparison is not fair as these models are trained on much larger and richer datasets than ours, leading to much richer vocabulary and better semantic understanding.

obtain the best performance. This is consistent with the intuition that $H(X)$ has to contain sufficient statistics about the sequence and that using only the mean is not sufficient for that purpose.

We also study scaling laws for PoM by training models at different scales (S/2, B/2, L/2 and XL/2 following the DiT naming scheme), has shown on Figure 5. PoM enjoys exponential decrease of the FID with respect to the sampling computing complexity as shown by a linear regression on the logarithmic plot. This is similar to what was observed for transformers in DiT [58].

5.2. Text to video generation

We evaluate our model generating videos of 5 seconds at 16 fps and 240p resolution on VBench [39] and show the results in Table 3. Note that contrarily to ImageNet, video generation is not as well standardized and models differ dramatically in terms of size, complexity and training dataset. Notably, most text-to-video generation models are trained on a mix of images and videos to get more diverse captions. In our case, we want to study the impact of enforcing temporal causality in the generation process and as such we limit our train set to WebVid-2M [3] only. Due to this smaller training set, we observed that our models are limited to a smaller vocabulary of objects, motion and styles.

We compare the standard architecture (denoted *no-mask*) with the use of a block-causal mask as detailed in section 3.1 (denoted as *b-causal*). As we can see, the impact of using a block causal mask is negative on some tasks like *subject consistency*, *background consistency* and *color*. This can be explain by the model struggling to follow the prompt for the first frames in the *block causal* case, which

penalizes consistency, whereas the *no-mask* case can leverage information from later frames to improve consistency. Interestingly, using a *block causal* mask improves temporal tasks like *dynamic degree*, *human action* and *temporal style*, which shows the importance of modeling properly the temporal aspect for these tasks.

6. Discussion

In this paper, we presented PoM, the Polynomial Mixer, a building block for neural networks that aims at replacing attention. PoM has a complexity linear with the sequence length and we prove it is a universal sequence-to-sequence approximator. To demonstrate the effectiveness of PoM, we train image and video generation models with it in lieu of Multi-Head Attention. These diffusion models obtain competitive results while being able to generate higher resolution images faster than with attention.

PoM is very interesting for high-definition video of long duration. However, the extreme cost of training such model makes this endeavor clearly out of the scope of a research paper. Another area where PoM could shine is LLMs and more particularly multimodal LLMs. Indeed, LLMs are causal, which means the generation of text could greatly benefit from the $\mathcal{O}(1)$ complexity of PoM for causal sequence. In addition, recent works [85] show that next token prediction and diffusion objectives can be merged in a single model. In that case the ability of PoM to seamlessly adapt from causal to block-causal masking scheme greatly reduces the complexity of such mixed training objective. As for high definition video, the extreme cost of training such large models also renders this endeavor out of the scope of a research paper.

7. Acknowledgment

This work was granted access to the HPC resources of IDRIS under the allocation 2024-AD011013085R2 made by GENCI. The authors would like to thank Vincent Lepetit, G  l Varol, Loic Landrieu and Dimitris Samaras for their insightful comments and suggestions.

References

- [1] Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023) [2](#)
- [2] Bai, X., Melas-Kyriazi, L.: Fixed point diffusion models. In: CVPR (2024) [2](#)
- [3] Bain, M., Nagrani, A., Varol, G., Zisserman, A.: Frozen in time: A joint video and image encoder for end-to-end retrieval. In: ICCV (2021) [6](#), [7](#)
- [4] Balaji, Y., Nah, S., Huang, X., Vahdat, A., Song, J., Zhang, Q., Kreis, K., Aittala, M., Aila, T., Laine, S., Catanzaro, B., Karras, T., Liu, M.Y.: ediff-i: Text-to-image diffusion models with ensemble of expert denoisers. arXiv preprint arXiv:2211.01324 (2022) [2](#)
- [5] Blattmann, A., Rombach, R., Ling, H., Dockhorn, T., Kim, S.W., Fidler, S., Kreis, K.: Align Your Latents: High-Resolution Video Synthesis with Latent Diffusion Models . In: CVPR (2023) [2](#)
- [6] Brandon, W., Mishra, M., Nrusimha, A., Panda, R., Kelly, J.R.: Reducing transformer key-value cache size with cross-layer attention. arXiv preprint arXiv:2405.12981 (2024) [2](#)
- [7] Brooks, T., Peebles, B., Holmes, C., DePue, W., Guo, Y., Jing, L., Schnurr, D., Taylor, J., Luhman, T., Luhman, E., Ng, C., Wang, R., Ramesh, A.: Video generation models as world simulators (2024), <https://openai.com/research/video-generation-models-as-world-simulators> [1](#)
- [8] Chang, H., Zhang, H., Jiang, L., Liu, C., Freeman, W.T.: Maskgit: Masked generative image transformer. In: CVPR (2022) [6](#), [7](#)
- [9] Chen, J., Ge, C., Xie, E., Wu, Y., Yao, L., Ren, X., Wang, Z., Luo, P., Lu, H., Li, Z.: Pixart-\sigma: Weak-to-strong training of diffusion transformer for 4k text-to-image generation. In: ECCV (2024) [2](#)
- [10] Chen, T., Li, L.: Fit: Far-reaching interleaved transformers. arXiv (2023) [2](#)
- [11] Child, R., Gray, S., Radford, A., Sutskever, I.: Generating long sequences with sparse transformers. arXiv preprint arXiv:1904.10509 (2019) [1](#), [2](#)
- [12] Crowson, K., Baumann, S.A., Birch, A., Abraham, T.M., Kaplan, D.Z., Shippole, E.: Scalable high-resolution pixel-space image synthesis with hourglass diffusion transformers. In: ICML (2024) [2](#)
- [13] Dao, T.: Flashattention-2: Faster attention with better parallelism and work partitioning. arXiv preprint arXiv:2307.08691 (2023) [2](#)
- [14] Dao, T., Fu, D., Ermon, S., Rudra, A., R  , C.: Flashattention: Fast and memory-efficient exact attention with io-awareness. In: NeurIPS (2022) [2](#)
- [15] Dao, T., Gu, A.: Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. In: Int. Conf. Mach. Learn. (2024) [3](#)
- [16] Dhariwal, P., Nichol, A.: Diffusion models beat gans on image synthesis. In: NeurIPS (2021) [6](#), [7](#)
- [17] Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al.: The llama 3 herd of models. arXiv preprint arXiv:2407.21783 (2024) [2](#)
- [18] Dufour, N., Besnier, V., Kalogeiton, V., Picard, D.: Don't drop your samples! coherence-aware training benefits conditional diffusion. In: CVPR (2024) [2](#)
- [19] Esser, P., Kulal, S., Blattmann, A., Entezari, R., M  ller, J., Saini, H., Levi, Y., Lorenz, D., Sauer, A., Boesel, F., et al.: Scaling rectified flow transformers for high-resolution image synthesis. In: Int. Conf. Mach. Learn. (2024) [1](#), [2](#)
- [20] Gao, P., Zhuo, L., Lin, Z., Liu, C., Chen, J., Du, R., Xie, E., Luo, X., Qiu, L., Zhang, Y., et al.: Lumina-t2x: Transforming text into any modality, resolution, and duration via flow-based large diffusion transformers. arXiv preprint arXiv:2405.05945 (2024) [2](#)
- [21] Ge, S., Nah, S., Liu, G., Poon, T., Tao, A., Catanzaro, B., Jacobs, D., Huang, J.B., Liu, M.Y., Balaji, Y.: Preserve your own correlation: A noise prior for video diffusion models. In: ICCV (2023) [2](#)
- [22] Girdhar, R., Singh, M., Brown, A., Duval, Q., Azadi, S., Rambhatla, S.S., Shah, A., Yin, X., Parikh, D., Misra, I.: Factorizing text-to-video generation by explicit image conditioning. In: ECCV (2024) [2](#)
- [23] Glorioso, P., Anthony, Q., Tokpanov, Y., Whittington, J., Pi-lault, J., Ibrahim, A., Millidge, B.: Zamba: A compact 7b ssm hybrid model. arXiv preprint arXiv:2405.16712 (2024) [3](#)
- [24] Gokaslan, A., Cooper, A.F., Collins, J., Seguin, L., Jacobson, A., Patel, M., Frankle, J., Stephenson, C., Kuleshov, V.: Commoncanvas: Open diffusion models trained on creative-commons images. In: CVPR (2024) [2](#)
- [25] Gu, A., Dao, T.: Mamba: Linear-time sequence modeling with selective state spaces. In: ICLR (2024) [1](#)
- [26] Gu, A., Goel, K., Re, C.: Efficiently modeling long sequences with structured state spaces. In: ICLR (2021) [1](#), [3](#)
- [27] Gu, A., Johnson, I., Goel, K., Saab, K., Dao, T., Rudra, A., R  , C.: Combining recurrent, convolutional, and continuous-time models with linear state space layers. In: NeurIPS (2021) [1](#), [3](#)
- [28] Gu, J., Zhai, S., Zhang, Y., Susskind, J.M., Jaitly, N.: Matryoshka diffusion models. In: ICLR (2023) [2](#)
- [29] Gupta, A., Yu, L., Sohn, K., Gu, X., Hahn, M., Li, F.F., Essa, I., Jiang, L., Lezama, J.: Photorealistic video generation with diffusion models. In: ECCV (2025) [2](#)
- [30] H  gele, A., Bakouch, E., Kosson, A., Allal, L.B., Werra, L.V., Jaggi, M.: Scaling Laws and Compute-Optimal Training Beyond Fixed Training Durations. In: NeurIPS (2024), <http://arxiv.org/abs/2405.18392> [6](#)

- [31] Hang, T., Gu, S.: Improved noise schedule for diffusion training. arXiv preprint arXiv:2407.03297 (2024) 2
- [32] Hatamizadeh, A., Song, J., Liu, G., Kautz, J., Vahdat, A.: Diffit: Diffusion vision transformers for image generation. In: ECCV (2024) 2
- [33] He, Y., Yang, T., Zhang, Y., Shan, Y., Chen, Q.: Latent video diffusion models for high-fidelity long video generation. arXiv preprint arXiv:2211.13221 (2022) 8
- [34] Ho, J., Chan, W., Saharia, C., Whang, J., Gao, R., Gritsenko, A., Kingma, D.P., Poole, B., Norouzi, M., Fleet, D.J., et al.: Imagen video: High definition video generation with diffusion models. arXiv preprint arXiv:2210.02303 (2022) 2
- [35] Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. In: NeurIPS (2020) 2
- [36] Ho, J., Salimans, T., Gritsenko, A., Chan, W., Norouzi, M., Fleet, D.J.: Video diffusion models. In: NeurIPS (2022) 2
- [37] Hong, W., Ding, M., Zheng, W., Liu, X., Tang, J.: Cogvideo: Large-scale pretraining for text-to-video generation via transformers. In: ICLR (2023) 2, 8
- [38] Hu, V.T., Baumann, S.A., Gui, M., Grebenkova, O., Ma, P., Fischer, J., Ommer, B.: Zigma: A dit-style zigzag mamba diffusion model. In: ECCV (2024) 1, 3
- [39] Huang, Z., He, Y., Yu, J., Zhang, F., Si, C., Jiang, Y., Zhang, Y., Wu, T., Jin, Q., Chanpaisit, N., et al.: Vbench: Comprehensive benchmark suite for video generative models. In: CVPR (2024) 8
- [40] Jabri, A., Fleet, D.J., Chen, T.: Scalable adaptive computation for iterative generation. In: Int. Conf. Mach. Learn. (2023) 2, 7
- [41] Jacob, P., Picard, D., Histace, A., Klein, E.: Metric learning with horde: High-order regularizer for deep embeddings. In: ICCV (2019) 3
- [42] Jaegle, A., Borgeaud, S., Alayrac, J.B., Doersch, C., Ionescu, C., Ding, D., Koppula, S., Zoran, D., Brock, A., Shelhamer, E., et al.: Perceiver io: A general architecture for structured inputs & outputs. In: ICLR (2022) 2
- [43] Jin, Y., Sun, Z., Li, N., Xu, K., Jiang, H., Zhuang, N., Huang, Q., Song, Y., Mu, Y., Lin, Z.: Pyramidal flow matching for efficient video generative modeling. arXiv preprint arXiv:2410.05954 (2024) 2
- [44] Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., Amodei, D.: Scaling laws for neural language models. arXiv preprint arXiv:2001.08361 (2020) 1, 2
- [45] Karras, T., Aittala, M., Lehtinen, J., Hellsten, J., Aila, T., Laine, S.: Analyzing and improving the training dynamics of diffusion models. In: CVPR (2024) 2
- [46] Karras, T., Aittala, M., Lehtinen, J., Hellsten, J., Aila, T., Laine, S.: Analyzing and improving the training dynamics of diffusion models. In: CVPR (2024) 2
- [47] Kitaev, N., Kaiser, L., Levskaya, A.: Reformer: The efficient transformer. In: ICLR (2020) 1, 2
- [48] Kwon, M., Oh, S.W., Zhou, Y., Liu, D., Lee, J.Y., Cai, H., Liu, B., Liu, F., Uh, Y.: Harivo: Harnessing text-to-image models for video generation. In: ECCV (2024) 2
- [49] Lee, S.H., Li, Y., Ke, J., Yoo, I., Zhang, H., Yu, J., Wang, Q., Deng, F., Entis, G., He, J., et al.: Parrot: Pareto-optimal multi-reward reinforcement learning framework for text-to-image generation. In: ECCV (2025) 2
- [50] Lieber, O., Lenz, B., Bata, H., Cohen, G., Osin, J., Dalmedios, I., Safahi, E., Meirom, S., Belinkov, Y., Shalev-Shwartz, S., et al.: Jamba: A hybrid transformer-mamba language model. arXiv preprint arXiv:2403.19887 (2024) 3
- [51] Lipman, Y., Chen, R.T., Ben-Hamu, H., Nickel, M., Le, M.: Flow matching for generative modeling. In: ICLR (2022) 2
- [52] Liu, X., Gong, C., et al.: Flow straight and fast: Learning to generate and transfer data with rectified flow. In: ICLR (2023) 2
- [53] Liu, Y., Tian, Y., Zhao, Y., Yu, H., Xie, L., Wang, Y., Ye, Q., Liu, Y.: Vmamba: Visual state space model (2024) 3
- [54] Liu, Y., Zhang, Y., Jaakkola, T., Chang, S.: Correcting diffusion generation through resampling. In: CVPR (2024) 2
- [55] Luohe, S., Hongyi, Z., Yao, Y., Zuchao, L., Hai, Z.: Keep the cost down: A review on methods to optimize llm's kv-cache consumption. arXiv preprint arXiv:2407.18003 (2024) 2
- [56] Ma, N., Goldstein, M., Albergo, M.S., Boffi, N.M., Vandeneijnden, E., Xie, S.: Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. In: ECCV (2024) 2, 7
- [57] Nichol, A.Q., Dhariwal, P.: Improved denoising diffusion probabilistic models. In: Int. Conf. Mach. Learn. (2021) 2
- [58] Peebles, W., Xie, S.: Scalable diffusion models with transformers. In: ICCV (2023) 1, 2, 5, 7, 8
- [59] Pei, X., Huang, T., Xu, C.: Efficientvmamba: Atrous selective scan for light weight visual mamba (2024) 3
- [60] Picard, D.: Torch.manual_seed(3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision (2023) 7
- [61] Polyak, A., Zohar, A., Brown, A., Tjandra, A., Sinha, A., Lee, A., Vyas, A., Shi, B., Ma, C.Y., Chuang, C.Y., et al.: Movie gen: A cast of media foundation models. arXiv preprint arXiv:2410.13720 (2024) 1
- [62] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer. JMLR (2020) 5
- [63] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: CVPR (2022) 2, 5, 7
- [64] Shi, Y., De Bortoli, V., Campbell, A., Doucet, A.: Diffusion schrödinger bridge matching. In: NeurIPS (2024) 2
- [65] Si, C., Huang, Z., Jiang, Y., Liu, Z.: Freeu: Free lunch in diffusion u-net. In: CVPR (2024) 2
- [66] Singer, U., Polyak, A., Hayes, T., Yin, X., An, J., Zhang, S., Hu, Q., Yang, H., Ashual, O., Gafni, O., et al.: Make-a-video: Text-to-video generation without text-video data. In: ICLR (2023) 2
- [67] Song, Y., Sohl-Dickstein, J., Kingma, D.P., Kumar, A., Ermon, S., Poole, B.: Score-based generative modeling through stochastic differential equations. In: International Conference on Learning Representations (2021) 2
- [68] Team, G., Anil, R., Borgeaud, S., Alayrac, J.B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A.M., Hauth, A., Millican, K., et al.: Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805 (2023) 2

- [69] Teng, Y., Wu, Y., Shi, H., Ning, X., Dai, G., Wang, Y., Li, Z., Liu, X.: Dim: Diffusion mamba for efficient high-resolution image synthesis. arXiv preprint arXiv:2405.14224 (2024) 1, 7
- [70] Tolstikhin, I.O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., et al.: Mlp-mixer: An all-mlp architecture for vision. In: NeurIPS (2021) 3
- [71] Touvron, H., Bojanowski, P., Caron, M., Cord, M., El-Nouby, A., Grave, E., Izacard, G., Joulin, A., Synnaeve, G., Verbeek, J., et al.: Resmlp: Feedforward networks for image classification with data-efficient training. IEEE TPAMI (2022) 3
- [72] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is all you need. In: NeurIPS (2017) 1, 2
- [73] Villegas, R., Babaeizadeh, M., Kindermans, P.J., Moraldo, H., Zhang, H., Saffar, M.T., Castro, S., Kunze, J., Erhan, D.: Phenaki: Variable length video generation from open domain textual descriptions. In: ICLR (2022) 2
- [74] Wallace, B., Dang, M., Rafailov, R., Zhou, L., Lou, A., Purushwalkam, S., Ermon, S., Xiong, C., Joty, S., Naik, N.: Diffusion model alignment using direct preference optimization. In: CVPR (2024) 2
- [75] Wang, J., Yuan, H., Chen, D., Zhang, Y., Wang, X., Zhang, S.: Modelscope text-to-video technical report (2023) 8
- [76] Wang, S., Li, B.Z., Khabsa, M., Fang, H., Ma, H.: Linformer: Self-attention with linear complexity. arXiv preprint arXiv:2006.04768 (2020) 1, 2
- [77] Wang, Y., Chen, X., Ma, X., Zhou, S., Huang, Z., Wang, Y., Yang, C., He, Y., Yu, J., Yang, P., et al.: Lavie: High-quality video generation with cascaded latent diffusion models. arXiv preprint arXiv:2309.15103 (2023) 8
- [78] Wei, F., Zeng, W., Li, Z., Yin, D., Duan, L., Li, W.: Powerful and flexible: Personalized text-to-image generation via reinforcement learning. In: ECCV (2024) 2
- [79] Yan, J.N., Gu, J., Rush, A.M.: Diffusion models without attention. In: CVPR. pp. 8239–8249 (2024) 1, 3, 7
- [80] Yang, Z., Teng, J., Zheng, W., Ding, M., Huang, S., Xu, J., Yang, Y., Hong, W., Zhang, X., Feng, G., et al.: Cogvideox: Text-to-video diffusion models with an expert transformer. arXiv preprint arXiv:2408.06072 (2024) 2, 5
- [81] Yun, C., Bhojanapalli, S., Rawat, A.S., Reddi, S., Kumar, S.: Are transformers universal approximators of sequence-to-sequence functions? In: ICLR (2020) 4
- [82] Zhai, X., Kolesnikov, A., Houlsby, N., Beyer, L.: Scaling vision transformers. In: CVPR (2022) 1, 2
- [83] Zhao, H., Lu, T., Gu, J., Zhang, X., Zheng, Q., Wu, Z., Xu, H., Jiang, Y.G.: Magdiff: Multi-alignment diffusion for high-fidelity video generation and editing. In: ECCV (2024) 2
- [84] Zhao, Y., Xu, Y., Xiao, Z., Jia, H., Hou, T.: Mobilediffusion: Instant text-to-image generation on mobile devices. In: ECCV (2024) 2
- [85] Zhou, C., Yu, L., Babu, A., Tirumala, K., Yasunaga, M., Shamis, L., Kahn, J., Ma, X., Zettlemoyer, L., Levy, O.: Transfusion: Predict the next token and diffuse images with one multi-modal model (2024) 8
- [86] Zhou, Z., Chen, D., Wang, C., Chen, C.: Fast ode-based sampling for diffusion models in around 5 steps. In: CVPR (2024) 2
- [87] Zhu, L., Liao, B., Zhang, Q., Wang, X., Liu, W., Wang, X.: Vision mamba: Efficient visual representation learning with bidirectional state space model. arXiv preprint arXiv:2401.09417 (2024) 3
- [88] Zuo, J., Velikanov, M., Rhaiem, D.E., Chahed, I., Belkada, Y., Kunsch, G., Hacid, H.: Falcon mamba: The first competitive attention-free 7b language model. arXiv preprint arXiv:2410.05355 (2024) 3

A. PoM pytorch code

In this section, we provide code in Pytorch for the main parts of the Polynomial Mixer as well as our diffusion blocks.

We found that writing dedicated functions for specific degrees led to faster runtime due to the ability of the PyTorch's compiler to optimize them. We show below implementation for degrees 2, 3 and 4.

```

1 @torch.compile
2 def po2(x: torch.Tensor):
3     h1, h2 = gelu(x).chunk(2, dim=-1)
4     h2 = h2 * h1
5     return torch.cat([h1, h2], dim=-1)
6
7 @torch.compile
8 def po3(x: torch.Tensor):
9     h1, h2, h3 = gelu(x).chunk(3, dim=-1)
10    h2 = h2 * h1
11    h3 = h3 * h2
12    return torch.cat([h1, h2, h3], dim=-1)
13
14 @torch.compile
15 def po4(x: torch.Tensor):
16     h1, h2, h3, h4 = gelu(x).chunk(4, dim=-1)
17     h2 = h2 * h1
18     h3 = h3 * h2
19     h4 = h4 * h3
20     return torch.cat([h1, h2, h3, h4], dim=-1)

```

Listing 1. Pytorch code for order specific PoM functions.

Next, we show the function that computes both the polynomial and the mixing depending on the degree and the presence of a mask.

```

1 def high_order_aggregation_(x: torch.Tensor, k:
2     int, mask=None):
3     if k == 2:
4         h = po2(x)
5     elif k == 3:
6         h = po3(x)
7     elif k == 4:
8         h = po4(x)
9     else:
10        h = list(gelu(x).chunk(k, dim=-1))
11        for i in range(1, k):
12            h[i] = h[i] * h[i-1]
13        h = torch.cat(h, dim=-1)
14    if mask is None:
15        h = h.mean(dim=1, keepdims=True)
16    else:
17        if mask.dim() == 2:
18            h = mask_mixer(h, mask.to(h.device))
19        elif mask.dim() == 3:
20            h = full_mask_mixer(h, mask.to(h.
device))
21        else:
22            raise Exception('unsupported dim for
mask (should be 2,3 or None)')
23    return h

```

Listing 2. Pytorch code for the complete polynomial and mixing part.

In the case the mask is 3 dimensional (batch, queries, context), we have a dedicated function that performs the partial sums. Note that this implementation is not optimized and that more speedup could be gained with a compiled mask.

```

1 def full_mask_mixer(h, mask):
2     mask = mask.type(h.dtype)
3     h = torch.einsum('bnd, bmn -> bmd', h, mask)
4         # b batch, n context tokens, m query tokens,
4         d dim
5     h = h / (1.e-7 + mask.sum(dim=2, keepdims=
True))
6     return h

```

Listing 3. Pytorch code for the mixer part with full mask.

The selection operation is very simple and consists in an element-wise product. The whole PoM operation is just the computation of $H(X)$ followed by the selection.

```

1 @torch.compile
2 def high_order_selection_(x: torch.Tensor, h:
3     torch.Tensor):
4     return F.sigmoid(x) * h
5
6 def pom(xq: torch.Tensor, xc: torch.Tensor, k:
7     int, mask=None):
8     h = high_order_aggregation_(xc, k, mask)
9     o = high_order_selection_(xq, h)
10    return o

```

Listing 4. Pytorch code for the selection part and the whole PoM function.

In the PoM module, we add the projections $W_{1\dots m}$, W_s and W_o for each part of the PoM operation.

```

1 class PoM(nn.Module):
2     def __init__(self, dim, order, order_expand,
3      bias=True):
4         super().__init__()
5         self.dim = dim
6         self.order = order
7         self.order_expand = order_expand
8         self.ho_proj = nn.Linear(dim, order*
order_expand*dim, bias=bias)
9         self.se_proj = nn.Linear(dim, order*
order_expand*dim, bias=bias)
10        self.ag_proj = nn.Linear(order*
order_expand*dim, dim, bias=bias)
11        self.hom = hom
12
13    def forward(self, xq, xc=None, mask=None):
14        if xc is None:
15            xc = xq # self attention
16
17        s = self.se_proj(xq)
18        h = self.ho_proj(xc)
19        sh = self.hom(s, h, self.order, mask)
20
21        # output projection
22        return self.ag_proj(sh)

```

Listing 5. Pytorch module for PoM.

For image diffusion, the base building block is simply a PoM module followed by an MLP, with residual connections and AdaLN modulations.

```

1 def modulation(x, scale, bias):
2     return x * (1+scale) + bias
3
4 class DiPBlock(nn.Module):
5     def __init__(self, dim: int, order: int,
6                  order_expand: int, ffw_expand: int):
7         super().__init__()
8         self.dim = dim
9         self.order = order
10        self.order_expand = order_expand
11        self.ffw_expand = ffw_expand
12
13        self.mha_ln = nn.LayerNorm(dim,
14                                   elementwise_affine=False, eps=1e-6)
15        self.pom = PoM(dim, order=order,
16                      order_expand=order_expand, bias=True)
17        self.ffw_ln = nn.LayerNorm(dim,
18                                   elementwise_affine=False, eps=1e-6)
19        self.ffw = nn.Sequential(nn.Linear(dim,
20                               ffw_expand * dim, bias=True),
21                               nn.GELU(),
22                               nn.Linear(
23                                   ffw_expand * dim, dim, bias=True))
24        self.cond_mlp = nn.Sequential(
25            nn.SiLU(),
26            nn.Linear(dim, 4
27 * dim, bias=True))
28        self.gate_mlp = nn.Sequential(
29            nn.SiLU(),
30            nn.Linear(dim, 2
31 * dim, bias=True))
32
33    def forward(self, x, c):
34        s1, b1, s2, b2 = self.cond_mlp(c).chunk(
35            4, -1)
36        g1, g2 = self.gate_mlp(c).chunk(2, -1)
37
38        # mha
39        x_ln = modulation(self.mha_ln(x), s1, b1)
40        x = x + self.pom(x_ln) * (1 + g1)
41
42        # ffw
43        x_ln = modulation(self.ffw_ln(x), s2, b2)
44        x = x + self.ffw(x_ln)*(1+g2)
45
46    return x

```

Listing 6. Pytorch module for the image diffusion block.

For text-to-video, we add a second PoM module that gathers information from the text.

```

1 class TextVideoDiPBlock(nn.Module):
2     def __init__(self, dim: int, order: int,
3                  order_expand: int, ffw_expand: int):
4         super().__init__()
5         self.dim = dim
6         self.order = order
7         self.order_expand = order_expand
8         self.ffw_expand = ffw_expand
9
10        self.mha_ln = nn.LayerNorm(dim,
11                                   elementwise_affine=False, eps=1e-6)

```

```

10        self.x_mha_ln = nn.LayerNorm(dim,
11                                     elementwise_affine=False, eps=1e-6)
12        self.c_mha_ln = nn.LayerNorm(dim,
13                                     elementwise_affine=False, eps=1e-6)
14        self.pom = PoM(dim, order=order,
15                      order_expand=order_expand, bias=True)
16        self.c_pom = PoM(dim, order=order,
17                      order_expand=order_expand, bias=True)
18        self.ffw_ln = nn.LayerNorm(dim,
19                                   elementwise_affine=False, eps=1e-6)
20        self.ffw = nn.Sequential(nn.Linear(dim,
21                               ffw_expand * dim, bias=True),
22                               nn.GELU(),
23                               nn.Linear(
24                                   ffw_expand * dim, dim, bias=True))
25        self.cond_mlp = nn.Sequential(
26            nn.SiLU(),
27            nn.Linear(dim, 8
28 * dim, bias=True))
29        self.gate_mlp = nn.Sequential(
30            nn.SiLU(),
31            nn.Linear(dim, 3
32 * dim, bias=True))
33
34    def forward(self, x, t, c, mask,
35                temporal_mask=None):
36        sx, bx, sc, bc, s1, b1, s2, b2 = self.
37        cond_mlp(t).chunk(8, -1)
38        gc, g1, g2 = self.gate_mlp(t).chunk(3,
39                                         -1)
40
41        # ca
42        x_ln = modulation(self.x_mha_ln(x), sx,
43                           bx)
44        c_ln = modulation(self.c_mha_ln(c), sc,
45                           bc)
46        x = x + self.c_pom(x_ln, c_ln, mask) * (1
47 + gc)
48
49        # sa
50        x_ln = modulation(self.mha_ln(x), s1, b1)
51        x = x + self.pom(x_ln, mask=temporal_mask
52) * (1 + g1)
53
54        # ffw
55        x_ln = modulation(self.ffw_ln(x), s2, b2)
56        x = x + self.ffw(x_ln)*(1+g2)
57
58    return x

```

Listing 7. Pytorch module for the video diffusion block.

B. Condition adherence

In this section we study the trade-off between image quality as measured with FID and condition adherence as measured with Inception Score (IS) by varying the weight ω of the classifier-free guidance (CFG). We show the results for a model of size L2 trained with the diffusion loss \mathcal{L}_D on Gigure 6. Inference is performed with 250 steps of DDIM sampling. As we can see, the model is perfectly able to balance FID and IS, leading to a typical 'U' curve where CFG improves both FID and IS at first, but then improvements of

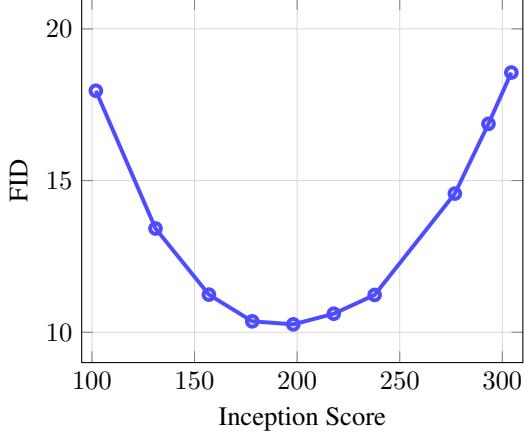


Figure 6. **Image quality versus condition adherence trade-off.** FID/IS curve for the L2 model with 250 DDIM sampling steps. Values are computed on 10k images against the validation set of ImageNet.

IS comes at the cost of FID. This is typical of mode collapse with the model generating low diversity but high quality images, similarly to what is observed with attention-based models.

C. Proof of Lemma 3

We first need to show that sets with different entries are mapped to different vectors. We first separate PoM into its two components:

$$s(X) = \sigma(W_s X) \quad (18)$$

$$H_k(X) = \left[h(W_1 X); \dots; \prod_m h(W_m X) \right] \mathbf{1} \mathbf{1}^\top \quad (19)$$

$$\text{PoM}(X) = W_o(s(X) \circ H_k(X)) \quad (20)$$

Assuming $\ker(W_o) = \{0\}$, and noting that $H_k(X)$ is the same for every column, we just have to show that $s(X)$ has different columns. This is easily achieved by having $\ker(W_s) = \{0\}$ since σ is injective and the composition of injective functions is itself injective.

Second, we have to show that sets that differ by at least one element are mapped to all different entries. To simplify notations, we will consider the special case where all matrices are the identity or an identity block positioned such as to perform submatrix selection. All the matrices can thus be removed from the formula. A similar argument can be made for matrices that are full rank as they preserve injectivity. We will also consider linear activations everywhere, which can be made as close as one wishes by partitioning the image of the activation function and performing piecewise linear approximation.

With this simplified version of PoM, we have to show that for 2 sets X, X' differing by at least one element (i.e.,

$\exists x' \in X', \forall x \in X, x \neq x'$), then there exist k such that

$$\forall x \in X, x' \in X', x \sum_{x_i \in X} x_i^k \neq x' \sum_{x_i \in X} x_i^k. \quad (21)$$

Consider the functions $P(t)$ and $P'(t)$ defined as follows:

$$P(t) = \sum_{x_i \in X} x_i^t \quad (22)$$

$$P'(t) = \sum_{x'_i \in X'} x'_i^t \quad (23)$$

Since X and X' differ by at least one element, there exists at least one $x_i \in X$ such that $x_i \neq x'_i, \forall x'_i \in X'$. This implies that the functions $P(t)$ and $P'(t)$ are not identical since are sums of exponentials with different bases.

Since $P(t)$ and $P'(t)$ are different functions, there must exist some k for which $P(k) \neq P'(k)$. In other words, there exists a k such that:

$$\sum_{x_i \in X} x_i^k \neq \sum_{x'_i \in X'} x'_i^k \quad (24)$$

For this k , let us denote $S_k = \sum_{x_i \in X} x_i^k$. We need to show that $xS_k \neq x'S'_k$ for all $x \in X$ and $x' \in X'$. Assume for the sake of contradiction that there exist $x \in X$ and $x' \in X'$ such that $xS_k = x'S'_k$. This implies:

$$x \sum_{x_i \in X} x_i^k = x' \sum_{x'_i \in X'} x'_i^k \quad (25)$$

Rearranging, we get:

$$\frac{x}{x'} = \frac{\sum_{x'_i \in X'} x'_i^k}{\sum_{x_i \in X} x_i^k} \quad (26)$$

Since $S_k \neq S'_k$, the right-hand side is not equal to 1. However, for this equality to hold for all $x \in X$ and $x' \in X'$, the ratio x/x' would need to be constant for all pairs (x, x') , which is not possible given that X and X' differ by at least one element.

Therefore, there exists a k such that $xS_k \neq x'S'_k$ for all $x \in X$ and $x' \in X'$.

D. Uncurated examples

In the following pages, we show randomly selected samples with obtained after 250 steps of DDIM sampling with the XL/2 model trained with the diffusion loss \mathcal{L}_D .



Figure 7. Uncurated 256^2 images for the class *magpie* (18).



Figure 8. Uncurated 256^2 images for the class *loggerhead*, *loggerhead turtle*, *Caretta caretta* (33).



Figure 9. Uncurated 256^2 images for the class *macaw* (88).



Figure 10. Uncurated 256^2 images for the class *otter* (360).



Figure 11. Uncurated 256^2 images for the class *balloon* (417).



Figure 12. Uncurated 256^2 images for the class *ice cream, icecream* (928).

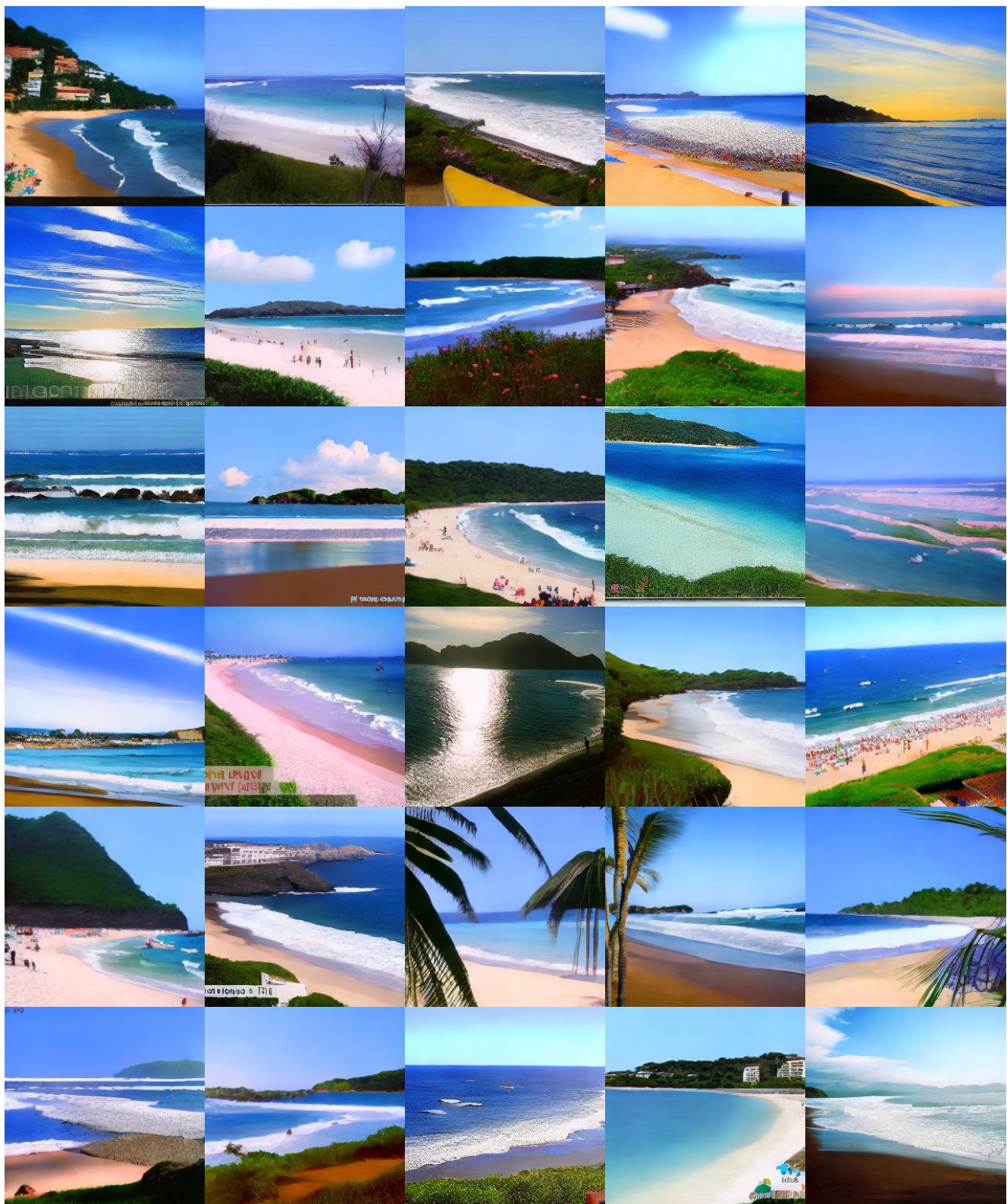


Figure 13. Uncurated 256^2 images for the class *seashore, coast, seacoast, sea-coast* (978).



Figure 14. Uncurated 256^2 images for the class *volcano* (980).