MASKED GENERATIVE NESTED TRANSFORMERS WITH DECODE TIME SCALING

Sahil Goyal[*,†] **Debapriya Tula**[*,‡,§] **Gagan Jain**[†] **Pradeep Shenoy**[†] **Prateek Jain**[†] **Sujoy Paul**[*,†]

ABSTRACT

Recent advances in visual generation have made significant strides in producing content of exceptional quality. However, most methods suffer from a fundamental problem - a bottleneck of inference computational efficiency. Most of these algorithms involve multiple passes over a transformer model to generate tokens or denoise inputs. However, the model size is kept consistent throughout all iterations, which makes it computationally expensive. In this work, we aim to address this issue primarily through two key ideas - (a) not all parts of the generation process need equal compute, and we design a decode time model scaling schedule to utilize compute effectively, and (b) we can cache and reuse some of the computation. Combining these two ideas leads to using smaller models to process more tokens while large models process fewer tokens. These different-sized models do not increase the parameter size, as they share parameters. We rigorously experiment with ImageNet256×256 , UCF101, and Kinetics600 to showcase the efficacy of the proposed method for image/video generation and frame prediction. Our experiments show that with almost 3× less compute than baseline, our model obtains competitive performance.

1 INTRODUCTION

The last decade has witnessed tremendous progress in image and video generation, under diverse paradigms - Generative Adversarial Networks (Brock, 2018; Sauer et al., 2022), denoising processes such as diffusion models (Ho et al., 2020; 2022b; Dhariwal & Nichol, 2021; Rombach et al., 2022; Gu et al., 2022), image generation via vector quantized tokenization (Razavi et al., 2019; Esser et al., 2021; Ge et al., 2022; Van Den Oord et al., 2017), and so on. In recent years, diffusion models and modeling visual tokens as language have been the de-facto processes used to generate high-quality images. While initially proposed with a CNN or U-Net based architectures (Rombach et al., 2022; Saharia et al., 2022), transformer models have become the norm now for these methods (Peebles & Xie, 2023; Yu et al., 2023a).

The recent advancements in visual generation can be categorized along two axes – (a) different types of denoising processes in the continuous latent space (Ho et al., 2020; Nichol & Dhariwal, 2021b), discrete space (Gu et al., 2022; Lou et al.) or masking in the discrete space (Yu et al., 2023a; Chang et al., 2022), continuous space (Li et al., 2024a) (b) modeling tokens either auto-regressively (Kondratyuk et al., 2024; Esser et al., 2021; Yu et al., 2021) with causal attention or parallel decoding with bi-directional attention (Gu et al., 2022; Yu et al., 2023a; Chang et al., 2022; Zheng et al., 2022). To achieve a high synthesis fidelity, both, denoising in diffusion models, and raster scan based auto-regressive token modeling require several iterations.

Recently, parallel decoding of discrete tokens have shown promise in generating high quality images with few iterations - MaskGIT (Chang et al., 2022), MAGVIT (Yu et al., 2023a), MUSE (Chang et al., 2023), MaskBIT (Weber et al., 2024), TiTok (Yu et al., 2024b). These models are trained with Masked Language Modeling (MLM) type losses, and the generation process involves unmasking a few confident tokens every decoding iteration, starting from all masked tokens. They can even surpass diffusion models, given a good visual tokenizer (Yu et al., 2023b; Weber et al., 2024).

[*]Equal contribution
[†]Google DeepMind
[‡]University of California, Los Angeles
[§]work done while at Google DeepMind

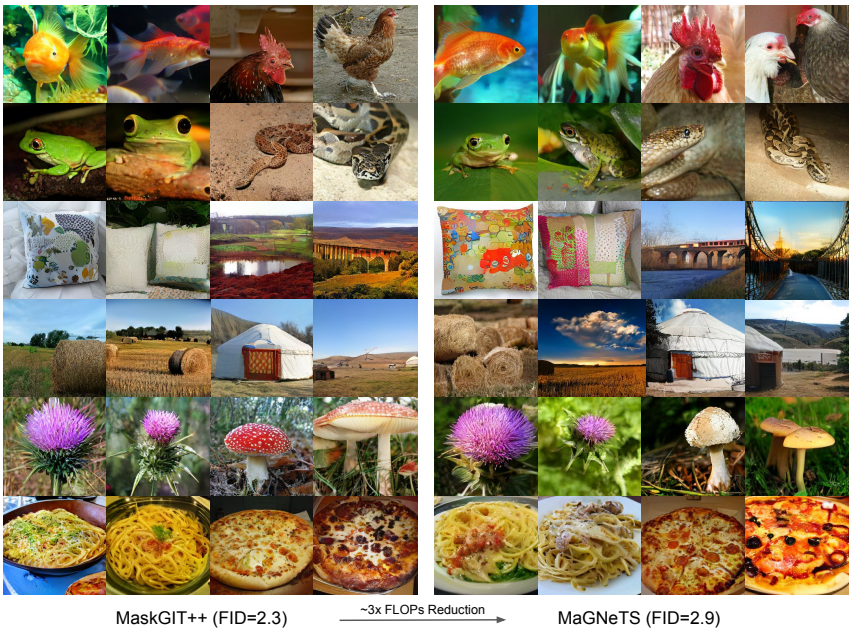


Figure 1: **Class-conditional image generation on ImageNet.**. Comparing MaskGIT++ and MaGNeTS

Although MaskGIT reduces decode complexity significantly, parallel decoding still includes several redundant computations. First, the need for same capacity model for all steps needs to be investigated. Second, unlike auto-regressive models, which cache its computation in all steps, parallel decoding models perform re-computation for all tokens. We empirically find that a smaller model can generate good-quality images but its performance saturates after a point with more decoding iterations. A bigger model can perform finer refinement and generate better-quality images.

Motivated by these observations, we present **Ma**sked **G**enerate **Ne**sted **T**ransformers with Decode Time **S**caling (MaGNeTS). We design a model size curriculum over the decoding process, which efficiently utilize compute. MaGNeTS gradually scales the model size up to the full model size over the decoding iterations instead of using a single large model throughout. Operating on discrete tokens, we cache key-value pairs of unmasked tokens and reuse them in later iterations. A combined effect of these two techniques leads to processing more tokens with smaller and fewer tokens with larger models. The heterogenous sized models share parameters, sequentially occupying larger and larger subspaces of the parameter space, as in MatFormer (Kudugunta et al., 2023). We build MaGNeTS on top of MaskGIT. We find that MaskGIT can be drastically improved using classifier-free guidance, specifically when trained with it. We call this MaskGIT++ and use this as the improved baseline, presenting all results on top of it.

On ImageNet, with $\sim$ 3$\times$ less compute, MaGNeTS generates images of similar quality as MaskGIT++ (see Figure 1). It is also comparable to state-of-the-art methods, which need significantly more compute. We also show MaGNeTS's efficacy on video datasets like UCF101 (Soomro et al., 2012) and Kinetics600 (Carreira et al., 2018). We summarize our main contributions below:

- We introduce model size scheduling over the generation process for inference efficiency.
- We show that like auto-regressive models, KV-caching can also be used in parallel decoding, which can effectively reuse computation when refreshed appropriately.
- We introduce nested modeling in image/video generation to exploit the above ideas effectively.
- Extensive experiments show that MaGNeTS offers 2.5 - 3.7$\times$ compute gains across tasks.

2 Related Work

Efficient Visual Generation. Image generation literature has seen significant improvements in the past years - Generative Adversarial Networks (Brock, 2018; Sauer et al., 2022), discrete token based models (Chang et al., 2022; Yu et al., 2023a), diffusion-based models (Kingma & Gao, 2023;

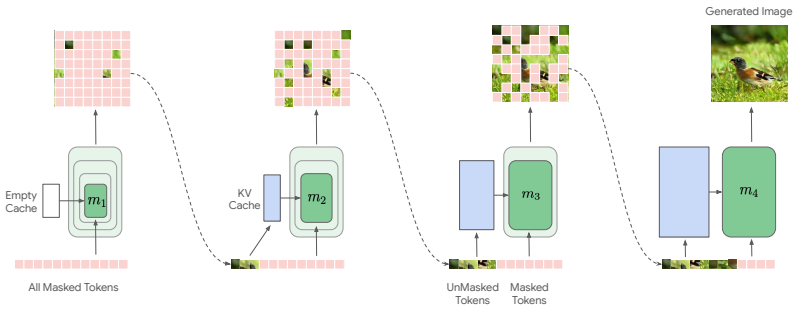


Figure 2: **MaGNeTS Decoding.** We start from the smallest nested model with an empty cache and gradually move to bigger models over the decode iters. We iterate with a model size for a few iterations, before moving onto the next model size. As we cache key-value for the unmasked tokens, the KV cache size also increases over time. We also refresh the cache when we switch models, hence its dimension increases over decode iters.

Hoogeboom et al., 2023), and more recently hybrid models (Peebles & Xie, 2023; Yu et al., 2024c), but they often guzzle computing power. Researchers tackle this bottleneck of computational costs with efficient model architectures and smarter sampling strategies.

In diffusion model literature, there have been some work to reduce the number of sampling steps, by treating the sampling process like ordinary differential equations (Song et al., 2022; Lu et al., 2022; Liu et al., 2022), incorporating additional training process (Kong & Ping, 2021; Nichol & Dhariwal, 2021a; Salimans & Ho, 2022; Song et al., 2023), sampling step distillation (Salimans & Ho, 2022; Song et al., 2023; Berthelot et al., 2023; Meng et al., 2023; Feng et al., 2024), sampling and training formulation modifications (Esser et al., 2024; Song et al., 2023), and more. Recently, there has been growing interest in understanding how each step in the diffusion sampling process contributes (Choi et al., 2022; Park et al., 2023; Lee et al., 2024). These approaches analyze sampling steps leveraging distance metrics such as LPIPS, Fourier analysis, and spectral density analysis. Building on these explorations researchers have designed methods based on optimal sampling steps (Watson et al., 2022; Lee et al., 2024), weighted training loss (Choi et al., 2022), and step-specific models (Li et al., 2023; Yang et al., 2024; Lee et al., 2023). These step-specific models use compute expensive evolutionary search algorithms, directly optimizing the quality metric, FID (Heusel et al., 2017). Concurrently, researchers are actively addressing the inherent architectural costs of diffusion models, particularly the transformer attention mechanism (Yuan et al., 2024; Yan et al., 2024).

Certain works also focus on building better tokenizers. Rombach et al. (2022) took diffusion models from pixel to compressed latent space for efficient and scalable generation. Yu et al. (2023b); Weber et al. (2024) explore vector quantizers in the tokenization process to improve generation quality. Tian et al. (2024) explore multi-scale tokenizer, while Yu et al. (2024b) look at 1D tokenizers to reduce the number of compressed tokens. Instead of sampling or tokenization optimization, we tackle an orthogonal problem of efficient compute allocation over the multi-step generation process. This makes our approach usable with a variety of tokenizers, architectures and sampling schemes.

Nested Models. Rippel et al. (2014) introduced nested dropout to learn ordered representations that improve retrieval speed and adaptive data compression. Matryoshka Learning (Kusupati et al., 2022) introduces the concept of nestedness in embedding space, making them flexible. MatFormer (Kudugunta et al., 2023) applies the same concept to the MLP hidden layer in each transformer block. Recent methods like (Cai et al., 2024a; Hu et al., 2024) explore the idea of nested models in multimodal large language models. MoNE (Jain et al., 2024) and Flextron (Cai et al., 2024b) learn to route tokens to variable sized nested models leading to inference compute efficiency. In this work, we show how different stages of a multi-step task like image generation, can be efficiently handled by nested models instead of relying on the full model at every step.

3 PRELIMINARIES

Parallel Decoding for Image Generation. Masked Generative Image Transformer (MaskGIT) (Chang et al., 2022) introduces a novel approach to image generation that significantly differs from

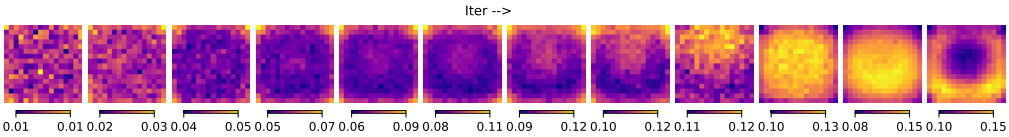


Figure 3: **Unmasked Token Density** visualization in each decoding iteration averaged over 50k generated samples on ImageNet. Yellow represents higher density. Each pixel represent a token from 16×16 latent token space. (See Appendix A for category-wise token density).

traditional autoregressive models. In autoregressive decoding, images are generated sequentially, one pixel/token at a time, following a raster scan order (Esser et al., 2021; Kondratyuk et al., 2024; Yu et al., 2024a; Li et al., 2024b). This sequential approach is computationally inefficient, as each token is conditioned only on the previously generated tokens, leading to a bottleneck in processing time. MaskGIT generates all tokens of an image simultaneously and then iteratively refines them. This method enables significant acceleration in the decoding process. The tokens are discrete and obtained using Vector Quantized (VQ) autoencoders, learned with self-reconstruction and photo-realism losses (Yu et al., 2023a). The iterative parallel decoding process is represented as:

$$\mathbf{X}_k \leftarrow \text{Mask} \circ \text{Sample}(M(\mathbf{X}_{k-1}, c), k) \tag{1}$$

where $\mathbf{X} \in \mathbb{Z}_{\geq 0}^{N}$, are the input tokens, N is the number of tokens, $k \in [1, K]$ denote the iteration number, with K being the total number of iterations, $\mathbf{X}_0$ is either completely masked for full generation, and partially masked for conditional generation tasks like frame prediction, c is the category of image/video under generation. The Sample function utilizes logits predicted by the model $M(.)$, introduces certain randomness, and sorts them by confidence, unmasking only top-k tokens while masking the rest. We also follow the above process (Chang et al., 2022; Yu et al., 2023a).

Nested Models. The core of our algorithm for inference-efficient decoding relies on variable-sized nested models for efficient parameter-sharing. We use MatFormer's (Kudugunta et al., 2023) modeling approach to extract multiple nested models, from a single model, without increasing the total parameter count. Given a full transformer model M, MatFormer defines nested models $\{m_1, \ldots, m_C\}$, such that $m_1 \subset m_2 \cdots \subset m_C = M$. Each m_i has fewer parameters and reduced compute. The core idea of extracting nested models is that in a transformer block, a reduced computation using a parameter subspace can be performed via a sliced matrix multiplication. Assuming a parameter matrix $\mathbf{W} \in \mathbb{R}^{d' \times d}$ and feature vector $\mathbf{x} \in \mathbb{R}^{d}$, then the computation $\mathbf{y} = \mathbf{W}\mathbf{x}$ can be partially obtained by computing $\mathbf{y}_{[:\frac{d'}{p}]} = \mathbf{W}_{[:\frac{d'}{p},:]}\mathbf{x}$, if $\mathbf{y}$ is desired to be partial and $\mathbf{y} = \mathbf{W}_{[:,:\frac{d}{p}]}\mathbf{x}_{[:\frac{d}{p}]}$, if input $\mathbf{x}$ is partial. Nested models can be obtained via partial computations throughout the network.

While MatFormer (Kudugunta et al., 2023) obtained sub-models with partial computation only in the MLP layer, we also do it in the Self-Attention layer, specifically in obtaining the $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ features. These features are of dimension $n_h \times \frac{d_h}{p}$, where n_h, d_h and p are number of attention heads, head feature dimension, and model downscaling factor respectively. We choose four downscaled models $C = 4$, with $p \in \{1, 2, 4, 8\}$ in this work. After attention computation, we obtain p-times downscaled features, that are projected back to the full model dimension d using partial computation, as the input features are partial. The same strategy is applied to the MLP layer. This process gives us models with close to linear reduction in parameter count and inference compute with downscaling factor p.

4 METHOD

Given the preliminaries, here we introduce the core algorithm. We first discuss the idea of scheduling models of different sizes over decode iters of MaskGIT. Then, we discuss the process of caching key-value in parallel decoding, followed by how to refresh them to improve performance. We finally discuss the nested model training method. See Figure 2 for a visual overview of our method.

Decode Time Model Schedule. In iterative parallel decoding (Chang et al., 2022; Yu et al., 2023a), the same-sized model is used for all steps, starting with all tokens being masked. However, we hypothesize that certain stages of the generation process might be easier than others. For example,

in the initial steps, the model only needs to capture coarse global structures, which is achieved efficiently using smaller models. In the later steps, the model must refine finer details, which requires larger models. This hypothesis is bolstered with Figure 3, which shows that the generation process starts unmasking tokens from the background and shifts to the middle of the image in the later iterations (more categorical examples in Appendix Figure 8).

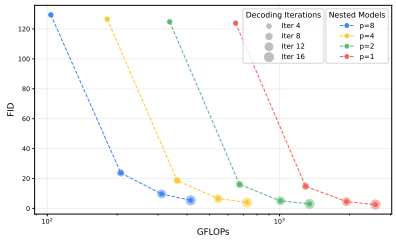


Figure 4: **Nested Models at different decoding iterations**. Different values of the downscaling factor p correspond to the nested models. The diameter of the blobs indicates #iterations.

Our hypothesis is further motivated by Figure 4, which presents the generation quality (FID) over iterations of parallel decoding for different-sized models. The smallest model reaches a reasonably good FID score with very low FLOPs compared to the biggest model. However, it saturates after a point, and the larger models surpasses the smaller ones in performance, demonstrating their ability to capture finer details and generate higher-quality images when provided with sufficient compute. This trend suggests that dynamically scaling the model size during decoding can exploit the varying task difficulty and achieve compute efficiency.

We use nested models to extract multiple models rather than using models with disjoint parameters. Nested models do not increase the parameter count and it also helps in better alignment of hypothesis when we shift model size over decode steps. The decode time model schedules can be generalized and represented as making the model choice in Equation (1) dependent on the iteration index as follows:

$$\mathbf{X}_k \leftarrow \text{Mask} \circ \text{Sample}(\mathcal{M}_k(\mathbf{X}_{k-1}, c), k)$$
$$\mathcal{M} = \{(m_{p_1})^{k_1}, (m_{p_2})^{k_2}, \ldots, (m_{p_n})^{k_n}\}, \quad \text{s.t.} \sum_i^n k_i = K \tag{2}$$

where $p_1, p_2, \ldots, p_n$ denoting the downscaling factors of the corresponding nested models, and $(m)^k$ denotes that model m will be executed for k iters. K represents the total number of iters. We can think of different model schedules - downscaling (starting with the full model and then gradually moving to the smallest model), upscaling, intermittently switching among a few models, and so on. We can also modify the integers k_i to choose the number of times we stick to a model before switching. However, as intuitively discussed before, we empirically validate that gradually upscaling the model size gets the best trade-off between the compute and generation quality.

Cached Parallel Decoding. Inspired by caching key-value pairs in auto-regressive models, we explore caching in parallel decoding, which retains relevant computations and enhances efficiency. In auto-regressive models, caching progressively happens in one fixed direction. However, in parallel decoding, caching must depend on which tokens are unmasked over the iterations.

Algorithm 1 MaGNeTS Decoding Algorithm

Input: $\mathbf{X}^0$ (Initial Tokens), K (#steps), N (#tokens), $\mathcal{M}$ (Nested Model Schedule), c (class),
Initialize: $k \leftarrow 0$; $cache \leftarrow \{\}$;
Note: $\mathbf{X}^0$ is a list of token ids (Mask token id = -1)
while *step* $k < K$ **do**
  if $k > 0$ *and* $\mathcal{M}_k \neq \mathcal{M}_{k-1}$ **then**
    Clear $cache$
  end
  Get uncached tokens: $\mathbf{X}^k_{uc} \leftarrow \{x_i \mid x_i \in \mathbf{X}^k;\ i \notin cache\}$
  Compute prediction probabilities and key-values: $p^k, (kv) \leftarrow \mathcal{M}_k(\mathbf{X}^k_{uc}; cache)$
  Sample tokens using current predictions p^k, without modifying previous predictions,

$$\mathbf{X}^{k+1} \leftarrow \text{MaskGIT-Sample}(p^k)$$

  New indices to cache: $\mathcal{C} \leftarrow \{i \mid i \notin cache,\ \mathbf{X}^{k+1}_i \neq -1\}$
  Update the kv cache: $cache \leftarrow cache \cup \{i : (kv)_i \mid i \in \mathcal{C}\}$
  $k \leftarrow k + 1$
end
return $\mathbf{X}^K$

Concretely, starting from an empty cached set, we keep adding keys and values to the set for the tokens that are unmasked after the Mask $\circ$ Sample steps (see Section 3). We do not update the predicted token indices for these unmasked tokens. Hence, the cached key and values for the unmasked tokens are the only features the other masked tokens need; hence, we do not need any further computation. In every decoding iteration, we can categorize tokens into three main categories: unmasked tokens (for which we have cached KV), tokens that will be unmasked during the current iteration, and the rest of the

Model	AR	FID ↓	IS ↑	Prec ↑	Rec ↑	# params	# steps	# Gflops
BigGAN-deep$^{\square}$ (Brock, 2018)		7.0	171.4	87	28	160M	1	-
StyleGAN-XL$^{\square g}$ (Sauer et al., 2022)		2.3	265.1	-	-	166M	1	-
Improved DDPM$^{\square}$ (Nichol & Dhariwal, 2021b)		12.3	-	70	62	280M	250	>150k
ADM + Upsample$^{\square g}$ (Dhariwal & Nichol, 2021)		3.9	215.8	83	53	554M	250	371k
LDM-4$^{\square g*}$ (Rombach et al., 2022)		3.6	247.7	-	-	400M	250	51.5k
DiT-XL/2$^{\square g*}$ (Peebles & Xie, 2023)		2.3	278.2	83	57	675M	250	59.5k
MDT$^{\square g*}$ (Gao et al., 2023)		1.8	283.0	81	61	676M	250	>59k
MaskDiT$^{\square g*}$ (Zheng et al., 2023)		2.3	276.6	80	61	736M	250	>28k
CDM$^{\square}$ (Ho et al., 2022a)		4.9	158.7	-	-	-	8100	-
RIN$^{\square}$ (Jabri et al., 2022)		3.4	182.0	-	-	410M	1000	334k
Simple Diffusion$^{\square g}$ (Hoogeboom et al., 2023)		2.4	256.3	-	-	2B	512	-
VDM++$^{\square g}$ (Kingma & Gao, 2023)		2.1	267.7	-	-	2B	512	-
EDiff$^{\square g}$ (Hang et al., 2024)		2.1	-	-	-	450M	50	119k
LPDM-ADM$^{\square g}$ (Wang et al., 2023)		2.7	-	-	-	-	50	7.8k
MAR$^{\square g}$ (Li et al., 2024b)	✓	1.8	296.0	81	60	479M	128	-
VQVAE-2$^{\square}$ (Razavi et al., 2019)	✓	31.1	~45	36	57	13.5B	5120	-
VQGAN$^{\square}$ (Esser et al., 2021)	✓	15.8	78.3	-	-	1.4B	256	-
VQGAN (architecture) + MaskGIT (setup)$^{\square}$		18.7	80.4	78	26	227M	256	-
MaskGIT$^{\square}$(Chang et al., 2022)		6.2	182.1	80	51	227M	8	647
Mo-VQGAN$^{\square}$ (Zheng et al., 2022)		7.2	130.1	72	55	389M	12	~1k
MaskBit$^{\square g}$ Weber et al. (2024)		1.7	341.8	-	-	305M	64	10.3k
PAR-4×$^{\square}$ Wang et al. (2024)	✓	3.8	218.9	84	50	343M	147	-
PAR-16×$^{\square}$ Wang et al. (2024)	✓	2.9	262.5	82	56	3.1B	51	-
MaskGIT++^{g_4}		2.5	260.3	83	54	303M	12	1.3k
MaskGIT++^{g_6}		2.3	280.6	84	51	303M	16	1.8k
MaGNeTS (ours)^{g_4}		3.1	254.8	85	50	303M	12	490
MaGNeTS (ours)^{g_6}		2.9	253.1	84	51	303M	16	608

Table 1: **Class-conditional Image Generation** on ImageNet 256 × 256. "# steps" refers to the number of neural network runs. $^{\square}$ denotes values taken from prior publications. * indicates usage of extra training data. g denotes use of classifier-free guidance (Ho & Salimans, 2022) for all steps. g_x represents use of guidance only for final x steps.

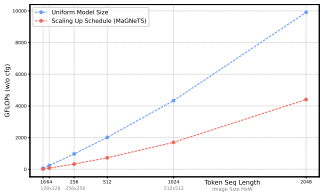


Figure 5: **Compute Comparison** between uniform model schedule (MaskGIT) and MaGNeTS, for 12 decode iters.

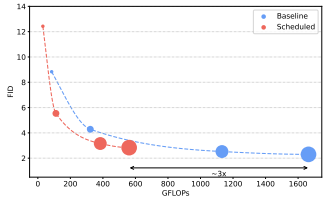


Figure 6: **Compute Scaling Curve.** Generation performance vs compute for different model sizes. The blob size indicates parameter count.

tokens. Note that the KV cache for the second category tokens cannot be used in the next iteration but only in the iteration after that once we know their token indices after the forward pass. We cache them in the next iteration for use in the immediately next iteration.

Caching is more useful for decode time model schedules. For a schedule that progressively scales up the model size as decoding progresses, smaller models process more tokens, while the larger models process fewer tokens, leading to an efficient yet good quality image generation process.

Intermittent Cache Refresh. Caching the key-value pairs for the unmasked tokens helps reduce computation, but it can slightly degrade performance. This happens because - (a) when we cache, the unmasked tokens are not updated in the subsequent iterations. (b) when we shift model size during generation, in the attention layer, the query size differs from the cached KV (see Section 3).

To remedy this, we strategically refresh the cache while changing the model size. Refreshing involves discarding the cached KV for that iteration and caching a newly computed KV for the immediate next iteration. We empirically find that it bridges the performance gap that arises due to caching. The proposed decode time model scaling algorithm is presented in Algorithm 1, which uses MaskGIT's sampling strategy (Chang et al., 2022; Yu et al., 2023a) to sample tokens from predicted logits.

Training Nested Models. MatFormer (Kudugunta et al., 2023) opts for a joint optimization of losses w.r.t. ground-truth from all models with equal weights. We found this mode of training to hurt performance with larger downscaling factors p. We introduce a combination of ground truth and distillation loss to address this issue. We perform online distillation progressively, where the teacher for model m_i is model m_{i+1}. The full model $m_N (= M)$ is trained with only ground truth loss. This provides a simpler optimization for the smaller nested models while maintaining the overall objective. Progressive distillation also reduces the teacher-student size gap, which can otherwise hurt distillation performance (Stanton et al., 2021; Beyer et al., 2022; Mirzadeh et al., 2019). Given input $\mathbf{X}$, ground truth label $\mathbf{Y}$ and loss function $\mathcal{L}$, our training loss is expressed as:

$$\mathcal{L}_{train} = \frac{1}{N}\Big(\mathcal{L}(m_N(\mathbf{X}), \mathbf{Y}) + \sum_{i=1}^{N-1} \alpha_i \mathcal{L}(m_i(\mathbf{X}), \mathbf{Y}) + (1-\alpha_i)\mathcal{L}(m_i(\mathbf{X}), m_{i+1}(\mathbf{X}))\Big) \quad (3)$$

where α_i, the weight between the distillation and ground truth loss, is linearly decayed from 1 to 0 as training progresses. Note that a stop gradient is applied during distillation on m_{i+1} in the third term of the equation.

Classifier-Free Guidance. We find that adding classifier-free guidance (Ho & Salimans, 2022) only to few final sampling steps offers similar quality images as applying to all (refer Figure 9b). See Appendix C for detailed analysis.

Method	Class	FVD ↓	IS↑	# params	# steps	# GFlops
RaMViD$^{\square\star}$ (Höppe et al., 2022)		-	21.71 ± 0.21	308M	500	-
StyleGAN-V$^{\square\star}$(Skorokhodov et al., 2022)		-	23.94 ± 0.73	-	1	-
DIGAN$^{\square}$ (Yu et al., 2022)		577±21	32.70± 0.35	-	1	~148
DVD-GAN$^{\square}$ (Clark et al., 2019)	✓	-	32.97± 1.70	-	1	-
Video Diffusion$^{\square\star}$ (Ho et al., 2022b)			57.00± 0.62	1.1B	256	-
TATS$^{\square}$ (Ge et al., 2022)		420± 18	57.63± 0.24	321M	1024	-
CCVS+StyleGAN$^{\square}$ (Le Moing et al., 2021)		386± 15	24.47± 0.13	-	-	-
Make-A-Video$^{\square\star}$ (Singer et al., 2022)	✓	367	33.00	-	-	-
TATS$^{\square}$ (Ge et al., 2022)	✓	332± 18	79.28± 0.38	321M	1024	-
CogVideo$^{\square\star}$ (Hong et al., 2022)	✓	626	50.46	9.4B	-	-
Make-A-Video$^{\square\star}$ (Singer et al., 2022)	✓	81	82.55	≫3.5B	≫250	-
PAR-4×$^{\square}$ Wang et al. (2024)	✓	99.5	-	792M	323	-
PAR-16×$^{\square}$ Wang et al. (2024)	✓	103.4	-	792M	95	-
MAGVIT-B$^{\square}$ (Yu et al., 2023a)	✓	159± 2	83.55± 0.14	87M	12	~1.3k
MAGVIT-L (Yu et al., 2023a)	✓	74.4± 2	89.54± 0.21	306M	12	~4.3k
MaGNeTS (ours)	✓	96.4±2	88.53±0.20	306M	12	~1.7k

Method	FVD ↓	IS ↑	# params	# steps	# GFlops
CogVideo$^{\square}$ (Hong et al., 2022)	109.2	-	9.4B	-	-
CCVS$^{\square}$ (Le Moing et al., 2021)	55.0±1.0	-	-	-	-
Phenaki$^{\square}$ (Villegas et al., 2022)	36.4 ± 0.2	-	1.8B	48	-
TrIVD-GAN-FP$^{\square}$ (Luc et al., 2020)	25.7 ± 0.7	12.54 ± 0.06	-	1	-
Transframer$^{\square}$ (Nash et al., 2022)	25.4	-	662M	-	-
RaMViD$^{\square}$ (Höppe et al., 2022)	16.5	-	308M	500	-
Video Diffusion$^{\square}$ (Ho et al., 2022b)	16.2 ± 0.3	15.64	1.1B	128	-
MAGVIT-B$^{\square}$	24.5± 0.9	-	87M	12	~1.3k
MAGVIT-L	7.2 ± 0.1	16.48 ± 0.01	306M	12	~ 4.3k
MAGVIT-L^{g_2}	6.6 ± 0.1	16.29 ± 0.01	306M	12	~ 5.1k
MaGNeTS (ours)	10.8 ± 0.1	16.25 ± 0.02	306M	12	~1.2k
MaGNeTS (ours)^{g_2}	9.6 ± 0.1	16.25 ± 0.01	306M	12	~1.4k

Table 2: **Class-conditional Video Generation** on UCF-101 (left) and **Frame prediction** on K600 (right). Methods in gray are pretrained on additional large video data. Methods with ✓ in the Class column are class-conditional, while the others are unconditional. In UCF101, methods marked with * use custom resolutions, while the others use 128×128. $^{\square}$ denotes values taken from prior publications. No guidance is used for UCF101. For K600, g_x denotes use of guidance only for final x steps.

5 Experiments and Results

We evaluate our model on ImageNet 256 × 256 (Deng et al., 2009) for image generation, UCF101 (Soomro et al., 2012) for video generation and Kinetics600 (Carreira et al., 2018) for frame prediction (5-frame condition). We use Fréchet Inception Distance (FID) (Heusel et al., 2017; Dhariwal & Nichol, 2021) for image generation, Fréchet Video Distance (FVD) (Unterthiner et al., 2019) for the video generation tasks, Inception Score (Salimans et al., 2016) for both tasks, as well as precision and recall for image generation. For efficiency, we compare algorithms using inference-time GFLOPs. Refer Appendix F for GFLOPs computation and Appendix B for implementation details.

5.1 Image Generation

Comparison with Baselines. Here we compare MaGNeTS with state-of-the-art methods for image generation. We list the results in Table 1 for 256×256 image generation on ImageNet-1k. Table 1 shows that MaGNeTS can speed up the generation process by $2.65 - 3\times$ (depending on total step count), with a similar FID. Refer Appendix F for real-time gains. Figure 5 illustrates that MaGNeTS significantly accelerates parallel decoding, which gets more pronounced as image resolution grows. Figure 1 and Figure 10 show generated images from MaskGIT++ and MaGNeTS (ours). As shown in recent literature, using a superior tokenizer (Yu et al., 2023b; Weber et al., 2024) or optimized training/inference configurations (Ni et al., 2024) can further boost MaGNeTS's performance. Note that several recent diffusion-based methods report results only on the low-resolution of ImageNet (typically 64×64), and therefore a direct comparison is not possible.

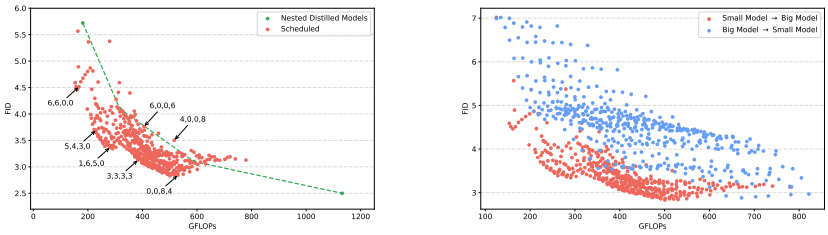
Scaling Analysis. To understand the scaling properties of MaGNeTS we train various sized models - S (22M), B (86M), L (303M) and XL (450M). We use the same hyper-parameters for all, such as learning rate, epochs, weight decay, etc. We present the results in Figure 6. It shows the compute vs performance of different models, with the blob size denoting the model size. For a certain parameter count, the baseline uses the full model for all 12 decoding steps, while the scheduled routines use a sequence of nested models with downsampling factors $p = 8, 4, 2, 1$ for 3 steps each. It can be seen that scaling up model size lead to almost $3\times$ cheaper compute scaling of MaGNeTS than baseline.

5.2 Video Generation

We use the MAGVIT (Yu et al., 2023a) framework to train parallel decoding based video generation and frame prediction models. Figure 11 shows generated videos of UCF101. We summarize the results for class-conditional video generation on UCF101 and for frame prediction on Kinetics600 in Table 2. Despite the challenging nature of video generation relative to image generation, results indicate that the decode time scaling of model size holds true even for video generation. MaGNeTS remains competitive to MAGVIT for frame prediction with $\sim 3.7\times$ lower compute.

5.3 Ablation Studies

Impact of Decode Time Model Schedule. We can think of different model schedules - scaling up model size, scaling down, periodic scaling up and down, and so on. For this analysis, we consider the L-sized model, with three nested models within it with parameter reduction by roughly $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}$. We can denote the number of times these four models are called during decoding as (k_1, k_2, k_3, k_4), s.t., $\sum_{i=1}^{4} k_i = 12$. We drop the model notation m_p in Equation (2) for simplicity and explicitly mention the model names in the text as discussed next.



(a) Scaling Up Schedules

(b) Scaling up vs down

Figure 7: **Scheduling Options.** (a) This shows the compute-performance trade-off for different schedule options while always scaling up model size over generation iters. The four numbers for each point denote the number of iters each model size operates in the order of downsampling factor $p = (8, 4, 2, 1)$. (b) This shows the benefit of scaling up model size compared to scaling it down during decoding.

Algorithm	Baseline	+ Cache	+ Refresh	Scheduled	+ Cache	+ Refresh
FID	2.5	3.4	2.6	3.1	4.8	3.1
FLOP Gains (times)	1.0	1.3	1.2	2.1	3.5	3.0

Dataset	Nested Models	Standalone Models
ImageNet (FID)	3.1	3.1
UCF101 (FVD)	96.4	115.0

Table 3: **Caching Ablation** (left) and **Nested vs Standalone Models** (right). We use L-sized models.

First, we evaluate all combinations of k_i for which we always scale up in Figure 7a in red and scale down in Figure 7b in blue. The green curve shows the performance of the individual nested models. We have the following observations - (1) for a certain compute budget, the scheduling of models over generation iterations (red dots) can offer better performance than using a single nested models (green curve) for all steps. (2) Models that have smoother transitions in nested models, such as $(3, 3, 3, 3)$ or $(0, 0, 8, 4)$, offer much better performance than the ones which has abrupt model transition such as $(6, 0, 0, 6)$ or $(3, 0, 0, 9)$, i.e., directly jumping from the smallest to the biggest model. (3) Figure 7b shows that scaling up nested model size offers much better performance than scaling down model size. This shows that bigger models are better utilized in the later iterations.

Impact of Caching and Refresh. We now discuss the impact of caching and its refresh. For this analysis, we use a uniform model schedule: $k_1 = k_2 = k_3 = k_4 = 3$. We also perform caching and refresh on the baseline model, which has not been trained with any nesting and has the same model applied for all iterations. We also refresh the cache at exactly the same steps as the scheduled model for the baseline. We present the results on ImageNet in Table 3. The columns "Baseline" and "Scheduled" do not involve any cache. While caching degrades the performance a bit, refreshing it intermittently avoids the degradation entirely at the cost of slight compute overhead. Scheduling of models with caching and refresh has the best compute-performance trade-off.

The efficiency of using nested models. In MaGNeTS we use nested models instead of separately trained smaller sized models. This has two advantages - (a) parameter sharing, which limits the number of parameters to just that of the full model, compared to $1.875\times$ $(= 1 + 1/2 + 1/4 + 1/8)$ for disjoint models, reducing memory requirements. (b) Nested models are trained efficiently in just a single training run. When trained with distillation, they generate better models than training like-sized standalone models (refer Appendix E). For performance comparison, we trained standalone models of the same size as the nested models for both UCF101 and ImageNet. The results are presented in Table 3. Nested models can efficiently share parameters without loss in performance (ImageNet) and offer constraints that help in better performance (UCF101) than standalone models.

6 Conclusion

In this paper, we propose MaGNeTS, a novel approach for allocating different compute to different steps of the image/video generation process. We show that instead of always using the same sized transformer model for all decoding steps, we can start from a model which is nested and fraction of its full size, and then gradually increase model size. This along with key-value caching in the parallel decoding paradigm obtains significant compute gains. We believe that our exploration of dynamic compute opens exciting new directions for research in efficient and scalable generative models. In future works, we plan to explore token-dependent model schedules for further compute gains.

REFERENCES

Fan Bao, Shen Nie, Kaiwen Xue, Yue Cao, Chongxuan Li, Hang Su, and Jun Zhu. All are worth words: A vit backbone for diffusion models, 2023. URL https://arxiv.org/abs/2209.12152.

David Berthelot, Arnaud Autef, Jierui Lin, Dian Ang Yap, Shuangfei Zhai, Siyuan Hu, Daniel Zheng, Walter Talbott, and Eric Gu. Tract: Denoising diffusion models with transitive closure time-distillation, 2023. URL https://arxiv.org/abs/2303.04248.

Lucas Beyer, Xiaohua Zhai, Amélie Royer, Larisa Markeeva, Rohan Anil, and Alexander Kolesnikov. Knowledge distillation: A good teacher is patient and consistent, 2022. URL https://arxiv.org/abs/2106.05237.

Andrew Brock. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

Mu Cai, Jianwei Yang, Jianfeng Gao, and Yong Jae Lee. Matryoshka multimodal models. *arXiv preprint arXiv:2405.17430*, 2024a.

Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. Flextron: Many-in-one flexible large language model. *arXiv preprint arXiv:2406.10260*, 2024b.

Joao Carreira, Eric Noland, Andras Banki-Horvath, Chloe Hillier, and Andrew Zisserman. A short note about kinetics-600, 2018. URL https://arxiv.org/abs/1808.01340.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11315–11325, 2022.

Huiwen Chang, Han Zhang, Jarred Barber, AJ Maschinot, Jose Lezama, Lu Jiang, Ming-Hsuan Yang, Kevin Murphy, William T Freeman, Michael Rubinstein, et al. Muse: Text-to-image generation via masked generative transformers. *arXiv preprint arXiv:2301.00704*, 2023.

Jooyoung Choi, Jungbeom Lee, Chaehun Shin, Sungwon Kim, Hyunwoo Kim, and Sungroh Yoon. Perception prioritized training of diffusion models, 2022. URL https://arxiv.org/abs/2204.00227.

Aidan Clark, Jeff Donahue, and Karen Simonyan. Adversarial video generation on complex datasets. *arXiv preprint arXiv:1907.06571*, 2019.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL https://arxiv.org/abs/1810.04805.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *CVPR*, pp. 12873–12883, 2021.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis, 2024. URL https://arxiv.org/abs/2403.03206.

Weilun Feng, Chuanguang Yang, Zhulin An, Libo Huang, Boyu Diao, Fei Wang, and Yongjun Xu. Relational diffusion distillation for efficient image generation, 2024. URL https://arxiv.org/abs/2410.07679.

Shanghua Gao, Pan Zhou, Ming-Ming Cheng, and Shuicheng Yan. Masked diffusion transformer is a strong image synthesizer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 23164–23173, 2023.

Songwei Ge, Thomas Hayes, Harry Yang, Xi Yin, Guan Pang, David Jacobs, Jia-Bin Huang, and Devi Parikh. Long video generation with time-agnostic vqgan and time-sensitive transformer, 2022. URL https://arxiv.org/abs/2204.03638.

Shuyang Gu, Dong Chen, Jianmin Bao, Fang Wen, Bo Zhang, Dongdong Chen, Lu Yuan, and Baining Guo. Vector quantized diffusion model for text-to-image synthesis, 2022. URL https://arxiv.org/abs/2111.14822.

Tiankai Hang, Shuyang Gu, Chen Li, Jianmin Bao, Dong Chen, Han Hu, Xin Geng, and Baining Guo. Efficient diffusion training via min-snr weighting strategy, 2024. URL https://arxiv.org/abs/2303.09556.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance, 2022. URL https://arxiv.org/abs/2207.12598.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Jonathan Ho, Chitwan Saharia, William Chan, David J Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *Journal of Machine Learning Research*, 23(47):1–33, 2022a.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *Advances in Neural Information Processing Systems*, 35:8633–8646, 2022b.

Wenyi Hong, Ming Ding, Wendi Zheng, Xinghan Liu, and Jie Tang. Cogvideo: Large-scale pretraining for text-to-video generation via transformers. *arXiv preprint arXiv:2205.15868*, 2022.

Emiel Hoogeboom, Jonathan Heek, and Tim Salimans. simple diffusion: End-to-end diffusion for high resolution images. In *International Conference on Machine Learning*, pp. 13213–13232. PMLR, 2023.

Tobias Höppe, Arash Mehrjou, Stefan Bauer, Didrik Nielsen, and Andrea Dittadi. Diffusion models for video prediction and infilling. *arXiv preprint arXiv:2206.07696*, 2022.

Wenbo Hu, Zi-Yi Dou, Liunian Harold Li, Amita Kamath, Nanyun Peng, and Kai-Wei Chang. Matryoshka query transformer for large vision-language models. *arXiv preprint arXiv:2405.19315*, 2024.

Allan Jabri, David Fleet, and Ting Chen. Scalable adaptive computation for iterative generation. *arXiv preprint arXiv:2212.11972*, 2022.

Gagan Jain, Nidhi Hegde, Aditya Kusupati, Arsha Nagrani, Shyamal Buch, Prateek Jain, Anurag Arnab, and Sujoy Paul. Mixture of nested experts: Adaptive processing of visual tokens, 2024. URL https://arxiv.org/abs/2407.19985.

Diederik P Kingma and Ruiqi Gao. Understanding the diffusion objective as a weighted integral of elbos. *arXiv preprint arXiv:2303.00848*, 2, 2023.

Dan Kondratyuk, Lijun Yu, Xiuye Gu, José Lezama, Jonathan Huang, Rachel Hornung, Hartwig Adam, Hassan Akbari, Yair Alon, Vighnesh Birodkar, et al. Videopoet: A large language model for zero-shot video generation. *ICML*, 2024.

Zhifeng Kong and Wei Ping. On fast sampling of diffusion probabilistic models, 2021. URL https://arxiv.org/abs/2106.00132.

Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham Kakade, Ali Farhadi, Prateek Jain, et al. Matformer: Nested transformer for elastic inference. *arXiv preprint arXiv:2310.07707*, 2023.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.

Guillaume Le Moing, Jean Ponce, and Cordelia Schmid. Ccvs: Context-aware controllable video synthesis. *Advances in Neural Information Processing Systems*, 34:14042–14055, 2021.

Haeil Lee, Hansang Lee, Seoyeon Gye, and Junmo Kim. Beta sampling is all you need: Efficient image generation strategy for diffusion models using stepwise spectral analysis, 2024. URL https://arxiv.org/abs/2407.12173.

Yunsung Lee, Jin-Young Kim, Hyojun Go, Myeongho Jeong, Shinhyeok Oh, and Seungtaek Choi. Multi-architecture multi-expert diffusion models, 2023. URL https://arxiv.org/abs/2306.04990.

Lijiang Li, Huixia Li, Xiawu Zheng, Jie Wu, Xuefeng Xiao, Rui Wang, Min Zheng, Xin Pan, Fei Chao, and Rongrong Ji. Autodiffusion: Training-free optimization of time steps and architectures for automated diffusion model acceleration, 2023. URL https://arxiv.org/abs/2309.10438.

Tianhong Li, Yonglong Tian, He Li, Mingyang Deng, and Kaiming He. Autoregressive image generation without vector quantization. *arXiv preprint arXiv:2406.11838*, 2024a.

Tianhong Li, Yonglong Tian, He Li, Mingyang Deng, and Kaiming He. Autoregressive image generation without vector quantization, 2024b. URL https://arxiv.org/abs/2406.11838.

Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. Pseudo numerical methods for diffusion models on manifolds, 2022. URL https://arxiv.org/abs/2202.09778.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Forty-first International Conference on Machine Learning*.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps, 2022. URL https://arxiv.org/abs/2206.00927.

Pauline Luc, Aidan Clark, Sander Dieleman, Diego de Las Casas, Yotam Doron, Albin Cassirer, and Karen Simonyan. Transformation-based adversarial video prediction on large-scale data. *arXiv preprint arXiv:2003.04035*, 2020.

Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik P. Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models, 2023. URL https://arxiv.org/abs/2210.03142.

Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant, 2019. URL https://arxiv.org/abs/1902.03393.

Charlie Nash, Joao Carreira, Jacob Walker, Iain Barr, Andrew Jaegle, Mateusz Malinowski, and Peter Battaglia. Transframer: Arbitrary frame prediction with generative models. *arXiv preprint arXiv:2203.09494*, 2022.

Zanlin Ni, Yulin Wang, Renping Zhou, Jiayi Guo, Jinyi Hu, Zhiyuan Liu, Shiji Song, Yuan Yao, and Gao Huang. Revisiting non-autoregressive transformers for efficient image synthesis, 2024. URL https://arxiv.org/abs/2406.05478.

Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models, 2021a. URL https://arxiv.org/abs/2102.09672.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pp. 8162–8171. PMLR, 2021b.

Yong-Hyun Park, Mingi Kwon, Jaewoong Choi, Junghyo Jo, and Youngjung Uh. Understanding the latent space of diffusion models through the lens of riemannian geometry, 2023. URL https://arxiv.org/abs/2307.12868.

William Peebles and Saining Xie. Scalable diffusion models with transformers, 2023. URL https://arxiv.org/abs/2212.09748.

Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019.

Oren Rippel, Michael Gelbart, and Ryan Adams. Learning ordered representations with nested dropout. In *International Conference on Machine Learning*, pp. 1746–1754. PMLR, 2014.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022. URL https://arxiv.org/abs/2112.10752.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in neural information processing systems*, 35:36479–36494, 2022.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models, 2022. URL https://arxiv.org/abs/2202.00512.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016. URL https://arxiv.org/abs/1606.03498.

Axel Sauer, Katja Schwarz, and Andreas Geiger. Stylegan-xl: Scaling stylegan to large diverse datasets, 2022. URL https://arxiv.org/abs/2202.00273.

Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, et al. Make-a-video: Text-to-video generation without text-video data. *arXiv preprint arXiv:2209.14792*, 2022.

Ivan Skorokhodov, Sergey Tulyakov, and Mohamed Elhoseiny. Stylegan-v: A continuous video generator with the price, image quality and perks of stylegan2. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3626–3636, 2022.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models, 2022. URL https://arxiv.org/abs/2010.02502.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models, 2023. URL https://arxiv.org/abs/2303.01469.

Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild, 2012. URL https://arxiv.org/abs/1212.0402.

Samuel Stanton, Pavel Izmailov, Polina Kirichenko, Alexander A. Alemi, and Andrew Gordon Wilson. Does knowledge distillation really work?, 2021. URL https://arxiv.org/abs/2106.05945.

Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction, 2024. URL https://arxiv.org/abs/2404.02905.

Thomas Unterthiner, Sjoerd van Steenkiste, Karol Kurach, Raphaël Marinier, Marcin Michalski, and Sylvain Gelly. Fvd: A new metric for video generation. 2019.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Ruben Villegas, Mohammad Babaeizadeh, Pieter-Jan Kindermans, Hernan Moraldo, Han Zhang, Mohammad Taghi Saffar, Santiago Castro, Julius Kunze, and Dumitru Erhan. Phenaki: Variable length video generation from open domain textual descriptions. In *International Conference on Learning Representations*, 2022.

Yuqing Wang, Shuhuai Ren, Zhijie Lin, Yujin Han, Haoyuan Guo, Zhenheng Yang, Difan Zou, Jiashi Feng, and Xihui Liu. Parallelized autoregressive visual generation, 2024. URL https://arxiv.org/abs/2412.15119.

Zhendong Wang, Yifan Jiang, Huangjie Zheng, Peihao Wang, Pengcheng He, Zhangyang Wang, Weizhu Chen, and Mingyuan Zhou. Patch diffusion: Faster and more data-efficient training of diffusion models, 2023. URL https://arxiv.org/abs/2304.12526.

Daniel Watson, William Chan, Jonathan Ho, and Mohammad Norouzi. Learning fast samplers for diffusion models by differentiating through sample quality, 2022. URL https://arxiv.org/abs/2202.05830.

Mark Weber, Lijun Yu, Qihang Yu, Xueqing Deng, Xiaohui Shen, Daniel Cremers, and Liang-Chieh Chen. Maskbit: Embedding-free image generation via bit tokens, 2024. URL https://arxiv.org/abs/2409.16211.

Jing Nathan Yan, Jiatao Gu, and Alexander M Rush. Diffusion models without attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8239–8249, 2024.

Shuai Yang, Yukang Chen, Luozhou Wang, Shu Liu, and Yingcong Chen. Denoising diffusion step-aware models, 2024. URL https://arxiv.org/abs/2310.03337.

Jiahui Yu, Xin Li, Jing Yu Koh, Han Zhang, Ruoming Pang, James Qin, Alexander Ku, Yuanzhong Xu, Jason Baldridge, and Yonghui Wu. Vector-quantized image modeling with improved vqgan. *arXiv preprint arXiv:2110.04627*, 2021.

Lijun Yu, Yong Cheng, Kihyuk Sohn, José Lezama, Han Zhang, Huiwen Chang, Alexander G Hauptmann, Ming-Hsuan Yang, Yuan Hao, Irfan Essa, et al. Magvit: Masked generative video transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10459–10469, 2023a.

Lijun Yu, José Lezama, Nitesh B Gundavarapu, Luca Versari, Kihyuk Sohn, David Minnen, Yong Cheng, Agrim Gupta, Xiuye Gu, Alexander G Hauptmann, et al. Language model beats diffusion–tokenizer is key to visual generation. *arXiv preprint arXiv:2310.05737*, 2023b.

Qihang Yu, Ju He, Xueqing Deng, Xiaohui Shen, and Liang-Chieh Chen. Randomized autoregressive visual generation, 2024a. URL https://arxiv.org/abs/2411.00776.

Qihang Yu, Mark Weber, Xueqing Deng, Xiaohui Shen, Daniel Cremers, and Liang-Chieh Chen. An image is worth 32 tokens for reconstruction and generation, 2024b. URL https://arxiv.org/abs/2406.07550.

Sihyun Yu, Jihoon Tack, Sangwoo Mo, Hyunsu Kim, Junho Kim, Jung-Woo Ha, and Jinwoo Shin. Generating videos with dynamics-aware implicit generative adversarial networks. *arXiv preprint arXiv:2202.10571*, 2022.

Sihyun Yu, Sangkyung Kwak, Huiwon Jang, Jongheon Jeong, Jonathan Huang, Jinwoo Shin, and Saining Xie. Representation alignment for generation: Training diffusion transformers is easier than you think, 2024c. URL https://arxiv.org/abs/2410.06940.

Zhihang Yuan, Hanling Zhang, Pu Lu, Xuefei Ning, Linfeng Zhang, Tianchen Zhao, Shengen Yan, Guohao Dai, and Yu Wang. Ditfastattn: Attention compression for diffusion transformer models. *arXiv preprint arXiv:2406.08552*, 2024.

Chuanxia Zheng, Long Tung Vuong, Jianfei Cai, and Dinh Phung. Movq: Modulating quantized vectors for high-fidelity image generation, 2022. URL https://arxiv.org/abs/2209.09002.

Hongkai Zheng, Weili Nie, Arash Vahdat, and Anima Anandkumar. Fast training of diffusion models with masked transformers. *arXiv preprint arXiv:2306.09305*, 2023.

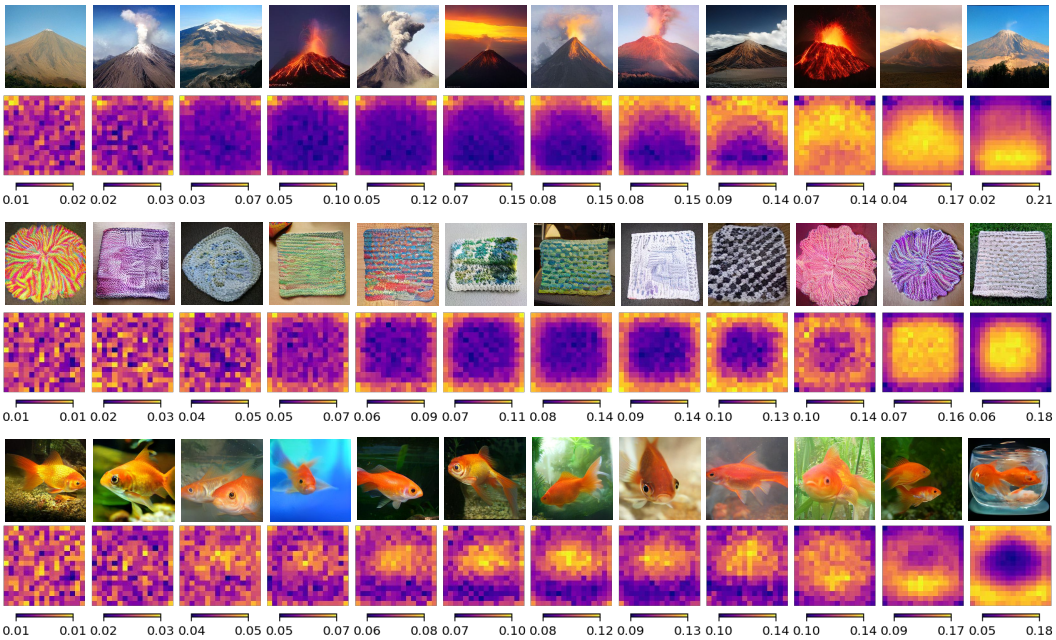


Figure 8: Visualization of token density unmasked in each iteration averaged over 10k generated samples on different categories of ImageNet. The top example shows category *volcano* (non-center-focused). Middle and bottom examples show *dishrag,dishcloth* and *goldfish,Carassius auratus* center-focused categories, respectively. Yellow color represents higher density, and each pixel represents a token from the 16×16 token space.

A Motivation for Decode Time Model Scaling

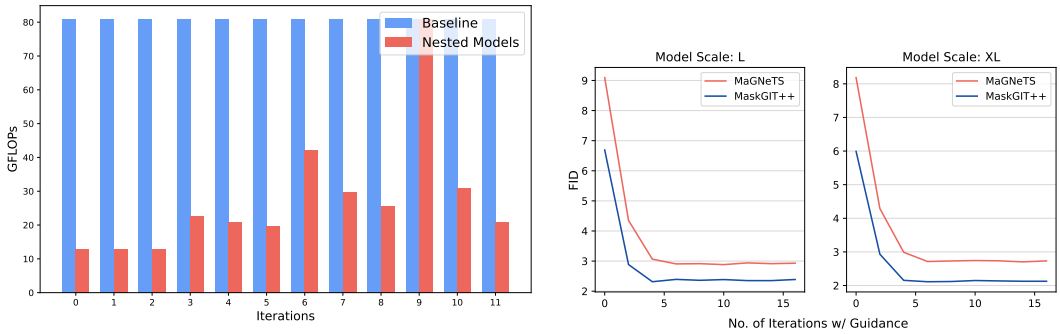
Our visualization of token density averaged across 50k ImageNet samples reveals a dynamic pattern - initial decoding iterations prioritize background regions. In contrast, later iterations focus on the center where foreground objects or region of interest typically reside. This highlights the need to allocate resources efficiently during generation. To further investigate this behavior, we examine token density across various ImageNet categories (refer Figure 8). This category-wise analysis further motivates our focus on decode time scaling. Figure 10 shows more qualitative results on ImageNet256×256 and Figure 11 shows samples on UCF101.

B Implementation Details

We utilize the pretrained tokenizers from MaskGIT (Chang et al., 2022) (for images) and MAGVIT (Yu et al., 2023a) (for videos) with the codebook size of 1024 tokens. We train models for image size 256×256. The tokenizer compresses it to 16×16 discrete tokens. For videos, we learn models for $16 \times 128 \times 128$, where the tokenizer outputs $4 \times 16 \times 16$ tokens. Following MaskGIT, we utilize the Bert model (Devlin et al., 2019) as a transformer backbone. We perform experiments at several model scales to understand the scaling behaviors of our algorithm. We utilize the same training hyper-parameters to train our nested models as these baselines. We train our model for 270 epochs for all the experiments. Unless otherwise mentioned, throughout the paper, we employ same number of steps per model before switching to the next model, i.e., $k_1 = k_2 = ... = k_n$. We follow a cosine schedule of unmasking tokens during inference. For image generation and frame prediction, we use classifier-free guidance for both MaGNeTS and respective baselines. Following literature, we drop input class condition labels for 10% of the training batches in image generation.

C Hyper-parameter Details

The MaskGIT algorithm has the following hyper-parameters which we discuss next.



(a) GFLOPs per sampling step.

(b) Guidance Analysis.

Figure 9: (a) Inference GFLOPs per step for baseline and MaGNeTS. (b) generation performance (FID) on ImageNet vs Number of decoding iterations w/ guidance for different model scales. Note that we start from last decoding iteration. For example, "No. of iterations w/ Guidance = 6" means we use guidance only for final six iterations (out of total 16 iterations). This shows that using guidance only for few final iterations is enough in the parallel decoding setup.

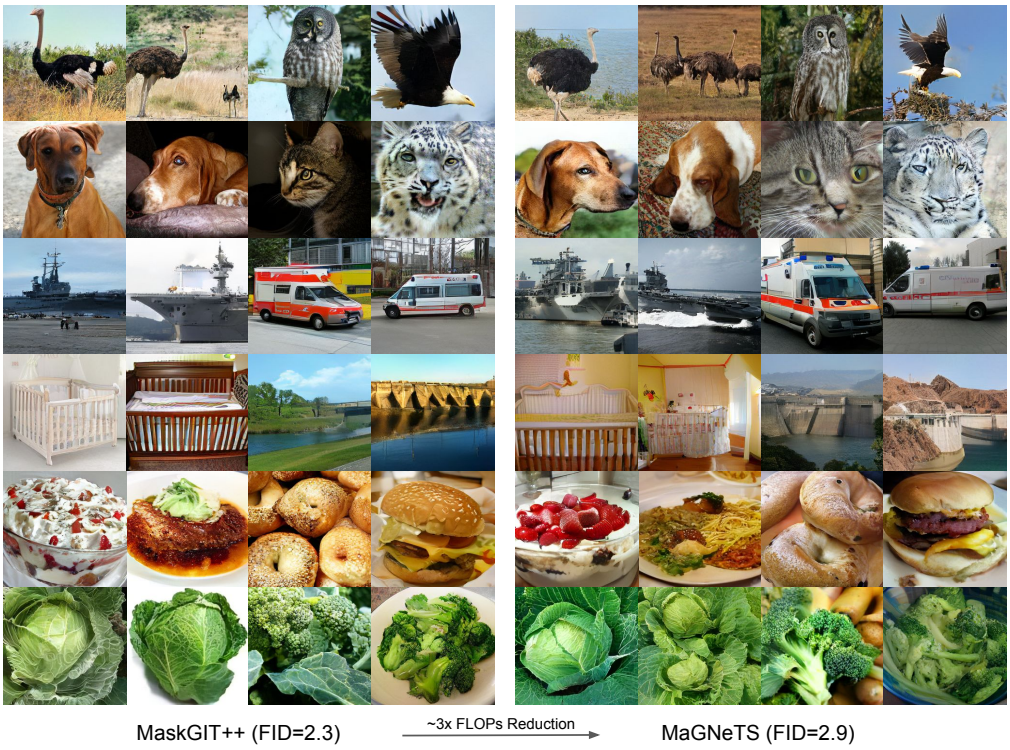


Figure 10: **Class-conditional Image Generation.** More qualitative results on ImageNet. Comparing MaskGIT++ and MaGNeTS (size: L, epochs: 270).

Guidance Scale (gs). It is used in classifier-free guidance (Ho & Salimans, 2022) and governs the calculation of final logits during inference as shown in Equation (4).

$$\text{logits}_{\text{final}} = \text{logits}_{\text{cond}} + \lambda \cdot \text{gs} \cdot (\text{logits}_{\text{cond}} - \text{logits}_{\text{uncond}}) \tag{4}$$

where $\text{logits}_{\text{cond}}$ are from class-conditional input, $\text{logits}_{\text{uncond}}$ are from unconditional input, and λ depends on the mask-ratio of the current decoding iteration.

Figure 8 shows that the initial decoding iterations of parallel decoding focus on the background region, and focus gradually shifts to the main object/region in the final decoding iterations. Motivated by this, we experimented with applying guidance to only few final decoding iterations and

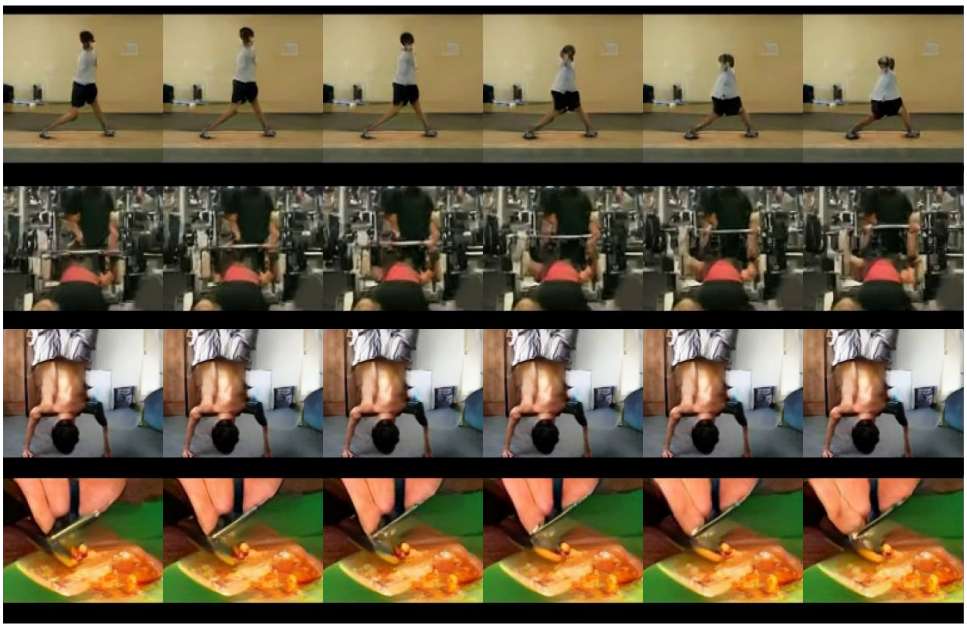


Figure 11: **Class-conditional Video Generation on UCF101.** 16-frame videos are generated at 128×128 resolution 25 fps. Every third frame is shown for each video. The classes from top to bottom are *Lunges*, *Bench Press*, *Handstand Pushups*, *Cutting In Kitchen*.

Figure 12: **Failure cases.** Similar to existing methods, our system can produce results with noticeable artifacts.

present our findings in Figure 9b. As we can see, most of the decoding iterations do not require guidance. We use guidance only for final few decoding iterations for class-conditional generation in ImageNet256×256 and frame prediction in Kinetics600. Following MAGVIT (Yu et al., 2023a), for class-conditional generation in UCF101 we do not use classifier-free guidance.

Mask Temperature (MTemp). It controls the randomness introduced on top of the token predictions to mask tokens.

Sampling Temperature (STemp). It controls the randomness of the sampling from the categorical distribution of logits. Tokens are sampled from $\text{logits}/\text{STemp}$. STemp is calculated by Equation (5).

$$\text{STemp} = \text{bias} + \text{scale} \cdot (1 - (k+1)/K) \tag{5}$$

where bias and scale are hyperparameters (see Table 4), k is the current decoding iteration and K is the total number of decoding iterations. We report the hyperparameters we use in in Table 4. We use bias=0.5 and scale=0.8 for all experiments.

Dataset	Method	gs	MTemp
ImageNet	MaskGIT++	65	6
	MaGNeTS	65	5
UCF101	MAGVIT/ MaGNeTS	0	5
Kinetics600	MAGVIT	10	12.5
	MaGNeTS	5	10

Table 4: Best Sampling Hyperparameters.

D Additional Experiments

D.1 Preliminary Diffusion experiments

We conduct initial experiments using model scheduling on diffusion models. Instead of training a new diffusion model with nesting and distillation, we focus solely on inference-time experiments. We use publicly available pretrained checkpoints of UViT (Bao et al., 2023) on ImageNet64×64. Specifically, we employ two models - U-ViT-L/4 (large) and U-ViT-M/4 (mid) - to investigate the impact of model scheduling during inference.

Implementation Details We use the default number of sampling steps of 50 and batch size of 500 in all experiments. We do not use classifier-free guidance. We do not use any caching for these experiments due to the continuous nature of the input. We use a single A100 GPU.

Optimal Model Schedule Since the initial denoising steps play a crucial role in shaping the final output of the reverse diffusion process, we utilize the L model for these early stages and transition to the M model for the later denoising steps. Given that the L model has greater denoising capacity than the M model, we customize the noise schedule with larger denoising step sizes for L and smaller step sizes for M, balancing efficiency and performance.

Quantitative Results Refer Table 5 for results. With only model scheduling, we are able to achieve ~**1.53x** inference compute gains with almost similar performance as baseline. Exploring more refined schedules, training the models with nesting and distillation will offer better compute gains. Additionally, nesting would enable parameter sharing, unlike the current setup, which relies on separate models. This shows that the proposed method of model scheduling over multi-step decode process in image/video generation is generic enough to be applied to different modeling approaches.

Method	FID (50k)	# params	# steps	Time (sec/iter)
U-ViT-M/4	5.92	131M	50	17.12
U-ViT-L/4	4.21	287M	50	32.34
Ours (model sched)	4.58	(131 + 287) M	50	21.10

Table 5: Class-conditional Image Generation on ImageNet64×64. "# steps" refers to the number of neural network runs.

Method	FID	# params	# steps	# GFLOPs
DPM-Solver$^{\square g}$ (Lu et al., 2022)	4.1	422M	12	>3k
MaskGIT++^{g_4}	3.2	303M	12	1.3k
MaGNeTS (ours)^{g_4}	3.9	303M	12	490

Table 6: Class-conditional Image Generation on ImageNet128×128. "# steps"refers to the number of neural network runs. $^{\square}$ denotes values taken from prior publications. g denotes use of classifier-free guidance (Ho & Salimans, 2022) for all steps. g_x represents use of guidance only for final x steps.

D.2 RESULTS ON IMAGENET128×128

We utilize a pretrained tokenizer tailored for 128×128 resolution. Tokenizer compresses the images to 16 × 16 discrete tokens. Refer Table 6 for results.

E ADDITIONAL ABLATIONS

Impact of number of nested models. We train different settings p={1, 2} (two models), p={1, 2, 4} (three models), p={1, 2, 4, 8} (four models), p={1, 2, 4, 8, 16} (five models), and p={1, 2, 4, 8, 16, 32} (six models). We observe that for all of these models, the biggest model performance remains almost same for all cases. However, the performance of the smaller models degrades as shown in Table 7.

We hypothesize that the drop in performance of smaller models is due to their lower representational power. As we add more nested models, the complexity of the shared representation increases, and burdens the smaller model. However, this drop in performance does not significantly impact the performance of model scheduling, as the larger models dominate the final results. Note that all of these results are on top of models trained with progressive distillation, which helps to retain the performance up to some extent. Table 8 also shows that introducing distillation loss helps improving the performance.

Model Schedules	2	3	4	5	6
$p = 1$	**2.3**	2.4	2.4	2.4	2.4
$p = 2$	**2.6**	2.7	2.7	2.8	2.8
$p = 4$	-	**3.4**	3.5	3.7	3.8
$p = 8$	-	-	**5.2**	5.7	6.3
$p = 16$	-	-	-	**8.9**	10.8
(0, 0, 0, 0, 6, 6)	**2.6**	2.6	2.6	2.7	2.7
(0, 0, 0, 4, 4, 4)	-	**2.7**	2.8	2.8	2.8
(0, 0, 3, 3, 3, 3)	-	-	**3.1**	3.2	3.2
(0, 3, 3, 2, 2, 2)	-	-	-	**6.3**	6.6

Table 7: **Ablation of number of nested models.** Rows represent different model schedules and columns represent the number of nested models used.

Impact of Distillation. We use two types of losses to train the nested sub-models - loss w.r.t the ground-truth tokens and distillation loss using the progressively bigger model as the teacher. The weight between the two losses is also linearly interpolated from the former to the latter. We compare this training strategy with the two extremes – only ground truth loss and only distillation loss and present the results in Table 8. As we can see, using only distillation loss results in divergence. Using ground-truth loss is inferior to linearly annealing between ground-truth and distillation loss.

Dataset	Training Algo.	$p = 1$	$p = 2$	$p = 4$	$p = 8$	Scheduled
ImageNet	Only GT	2.4	2.9	3.9	5.7	3.1
	Only Distill	⟵ Training Diverged ⟶				
	GT → Distill	2.4	**2.7**	**3.5**	**5.2**	3.1
UCF101	Only GT	80.0	101.3	143.8	221.8	112.6
	Only Distill	⟵ Training Diverged ⟶				
	GT → Distill	**78.3**	**91.2**	**115.4**	**164.4**	**96.4**

Table 8: **Distillation Ablation.** This shows the impact of different training losses used for the nested sub-models on ImageNet256×256 (size: L) and UCF101 (size: L). Using only distillation diverges while using only ground-truth losses performs worse than our approach (third row), where we combine ground-truth and distillation losses with a linear decay from the former to the latter.

Nested Attention Heads We also investigate nesting along the number of attention heads (n_h), applying the same partial computation strategy as discussed before. However, this generally performed worse than nesting along the head feature dimension in attention, which is what we use for this work.

F COMPUTE GAINS

Per-step FLOPs. Figure 9a illustrates the inference-time computational cost, measured in GFLOPs, per iteration for the baseline model and MaGNeTS. As we can see the amount of FLOPs are drastically reduced using MaGNeTS. This is for a schedule with $k_1 = k_2 = k_3 = k_4 = 3$. The spikes after every 3 iterations are due to the cache refresh step. Mechanisms to get rid of the cache refresh can further reduce the total compute needed.

Calculation of GFLOPs. We illustrate the calculation of inference GFLOPs via Python pseudocode in Table 10. We double the GFLOPs in decoding iterations where classifier-free guidance (Ho & Salimans, 2022) is used. Note that we always use a cosine schedule to determine the number of tokens to be unmasked in every step.

Real-Time Inference Benefits. In addition to the theoretical FLOP gains offered by MaGNeTS , here we want to analyze the real-time gains that it offers. We implement MaGNeTS on a single TPUv5 chip and present the results in Table 9.

Algorithm →	Baseline (MaskGIT++)	MaGNeTS
Images/Sec	22.5	56.3
Latency (ms)	712	285

Table 9: **Real-Time Inference Efficiency.** These show the number of generated images per sec and latency. These results are on ImageNet256×256 with model size XL.

G LIMITATIONS.

While our approach demonstrates strong performance in image and video generation, we acknowledge certain limitations. Some artifacts inherent to MaskGIT++ may also appear in our generated outputs (see Figure 12 for examples on ImageNet256×256). Such artifacts are common in models trained on controlled datasets like ImageNet. Moreover, the quality of the pretrained tokenizers (Yu et al., 2023b; Weber et al., 2024) directly impacts our method's effectiveness; however, improving these tokenizers is beyond the scope of this work. Although, use of nesting and decode time scaling does not have any specific requirement for model architecture and sampling scheme, KV caching requires discrete tokens.

```python
# Function to get the GFlops for current decoding iteration
def get_flops(num_tokens_cached, num_tokens_processed, model_id, params, version):
    num_layers, hidden_size, mlp_dim, num_heads = params[version]
    qkv = 4 * num_tokens_processed * hidden_size * (hidden_size // model_id)
    attn = 2 * num_tokens_processed * (num_tokens_processed + num_tokens_cached) * hidden_size
    mlp = 2 * num_tokens_processed * (mlp_dim // model_id) * hidden_size
    return (qkv + attn + mlp) * num_layers // 1e9

# Function to get the total inference GFlops
def get_total_flops(version, num_iters, use_cache, refresh_cache_at, total_tokens, model_id_schedule, params, num_cond_tokens=0):
    assert num_cond_tokens < total_tokens
    refresh_cache_at = [int(x) for x in refresh_cache_at.split(',') if x]
    assert len(model_id_schedule) == num_iters
    num_cached = 0
    total_flops = 0

    # MaGNeTS (ours) doesn't need to process the conditioned tokens in the frame prediction task
    total_tokens -= num_cond_tokens

    for i in range(num_iters):
        ratio = i / num_iters

        # Cosine masking schedule
        num_processed = np.cos(np.pi/2. * ratio) * total_tokens

        # Even if we are performing caching, all tokens are processed in first iteration and iterations where cache is refreshed
        if i == 0 or i in refresh_cache_at and use_cache:
            total_flops += get_flops(0, total_tokens+num_cond_tokens, model_id_schedule[i], params, version)

        # we always cache the conditioned tokens
        else:
            total_flops += get_flops(num_cached+num_cond_tokens, total_tokens-num_cached, model_id_schedule[i], params, version)

        if use_cache:
            num_cached = total_tokens - num_processed
    return total_flops
```

```python
# Sample function call for class-conditional image generation
# params is a dictionary of the form {version: (num_layers, hidden_size, mlp_dim, num_heads)}
common = {'version': 'L', 'num_iters': 12, 'total_tokens': 257, 'params': params}
baseline = {'use_cache': False, 'refresh_at': '', 'model_id_schedule': (1,)*12, **common}
ours = {'use_cache': True, 'refresh_at': '3,6,9', 'model_id_schedule': (8,)*3+(4,)*3+(2,)*3+(1,)*3, **common}

print(get_total_flops(**baseline), get_total_flops(**ours))

# total_tokens = 1025 for class-conditional video generation and frame prediction
# num_cond_tokens = 512 for frame prediction
```

Table 10: **Python pseudo-code illustrating the calculation of inference GFLOPs.**