# A Design Choices

In the following section, we investigate random token injection $\alpha$ in Table 4, the number of prediction steps in Table 4, followed by the type of scheduling methods in Table 5, and token ordering in Table 6. Each factor influences efficiency and accuracy, as discussed below.

**Number of Steps** We investigate in Table 4 the effect of the total number of steps $\{8, 16, 24, 32\}$ to predict the full images. On ImageNet, increasing the number of steps improves performance up to $step = 16$, beyond which the benefits plateau. On the other hand, increasing the number of steps to 24 leads to improved results on Cifar10, suggesting that the step count should be scaled proportionally to the total number of tokens to be predicted.

| | | ImageNet | | Cifar10 | |
|---|---|---|---|---|---|
| **Step** | $\alpha$ | **FID50k**↓ | **IS**↑ | **FID10k**↓ | **IS**↑ |
| 8 | 0.0 | 42.84 | 29.17 | 86.58 | 7.57 |
| | 0.1 | 38.70 | 33.38 | 80.30 | 7.82 |
| | 0.2 | **34.93** | 36.68 | 67.14 | 8.43 |
| | 0.3 | 34.98 | **37.6** | **66.52** | **8.64** |
| 16 | 0.0 | 43.76 | 26.21 | 39.30 | 9.86 |
| | 0.1 | 35.94 | 32.71 | 26.88 | 11.19 |
| | 0.2 | **31.27** | 37.79 | **25.76** | **11.06** |
| | 0.3 | 32.66 | **39.09** | 27.04 | **11.06** |
| 24 | 0.0 | 45.15 | 24.68 | 30.50 | 10.85 |
| | 0.1 | 35.82 | 31.75 | **20.66** | **11.77** |
| | 0.2 | **31.96** | 35.77 | 22.07 | 11.44 |
| | 0.3 | 31.98 | **37.81** | 22.85 | 11.46 |
| 32 | 0.0 | 46.86 | 23.40 | 26.53 | 10.69 |
| | 0.1 | 36.03 | 30.56 | **20.78** | **11.87** |
| | 0.2 | **32.47** | 34.87 | 22.26 | 11.62 |
| | 0.3 | 33.57 | **35.42** | 23.23 | 11.61 |

Table 4: **ImageNet 256 / Cifar10:** Ablation on the random token injection $\alpha$ and the number prediction steps, without cfg. We show that enabling token fixing, i.e., $\alpha > 0$, largely improves the metrics, while 16 steps is a good trade-off between FID/IS and compute efficiency.

**Influence of Scheduling Strategy** To analyze the effect of different scheduling methods we measure performance across various configurations {square, arccos, linear, root, constant}, and show the results in Table 5. Similarly to Chang et al. (2023) finding, we find out that the arccos scheduling performs the best while concave scheduling performs worse.

| **Scheduler** | **FID50k**↓ | **IS**↑ |
|---|---|---|
| root | 39.08 | 32.55 |
| linear | 31.29 | 38.02 |
| cosine | 31.45 | 36.11 |
| square | 31.27 | 37.79 |
| arccos | **29.68** | **40.28** |

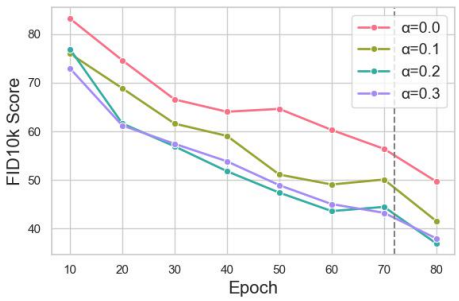
Table 5: Ablation on scheduling methods $\alpha = 0.2$, $Steps = 16$. Results suggest that convex schedulers, like arccos or square, perform the best.

**Token Prediction Order** We compare different token selection strategies {Halton, Spiral, Raster Scan} in Table 6. We find that Halton ordering significantly outperforms raster scan and spiral selection in both metrics. This demonstrates the advantage of structured, but detached, token sampling in guiding the prediction process more effectively. We

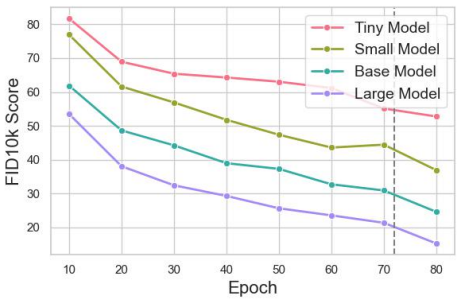
also compare the performance of 'starting from the same token' location versus 'rolling out the sequence' (Halton+Roll). Specifically, we apply a circular shift to the sequence during both training and testing, enabling the model to begin from any token location in the image. Our results indicate that there is no significant boost in performance between those two strategies.

| Sequence | FID50k↓ | IS↑ |
|---|---|---|
| Halton | 31.62 | **37.87** |
| Halton + Roll | **31.27** | 37.79 |
| Raster Scan | 43.60 | 34.29 |
| Spiral | 36.34 | 26.71 |

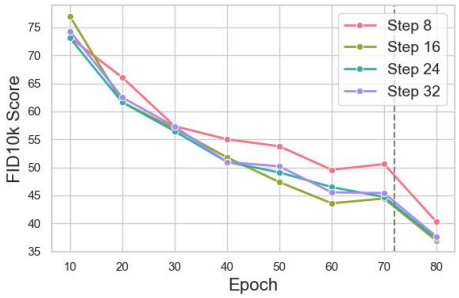
Table 6: Ablation on the sequence token ordering on ImageNet 256. Results show that predicting tokens uniformly (Halton sequence) in the image yields better generation.



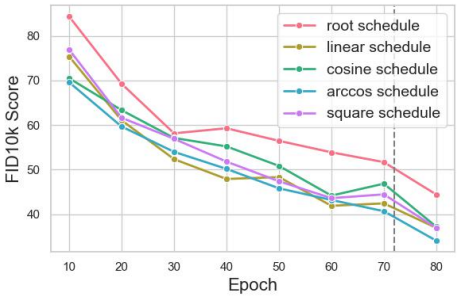

(a) Ablation on the random token injection $\alpha$




(b) Ablation on model sizes




(c) Ablation on prediction steps




(d) Ablation on scheduling methods

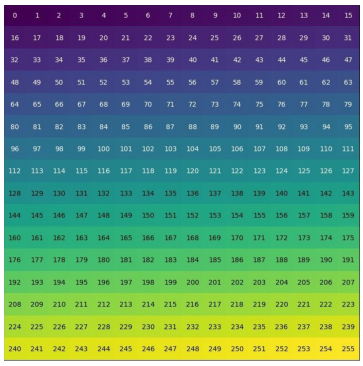
Figure 7: **ImageNet 256:** FID10k evolution across model training on ImageNet-256, without cfg and 410k iterations. The moment where learning rate decay was applied is showcased by the dash grey line.

**Summary of Findings:** Our ablation study highlights key insights into the impact of different hyper-parameters on model performance Figure 7. We find that using a moderate level of random token injection ($\alpha = 0.2$) drastically improves the performance. Setting the number of prediction steps to between 16 and 32 provides an optimal trade-off between efficiency and quality. Additionally, increasing the number of tokens following an arccos-based scheduling strategy outperforms alternative approaches for guiding token prediction. Finally, leveraging the Halton sequence for the token ordering leads to significantly enhanced image quality and therefore it serves as the baseline method we compare to in the main paper.

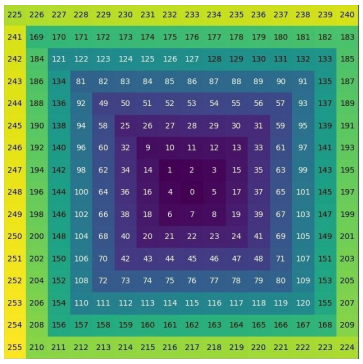
## B Sampling Pattern

An important aspect of our method is the order in which tokens are predicted. In the previous section, we show that the Halton ordering outperforms alternative approaches. Figure 8 illustrates

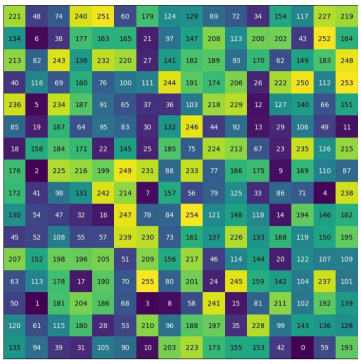
these token sequences on a $16 \times 16$ grid. Notably, the Raster scan method immediately reveals its limitations when applied to a 2D grid, as it enforces a rigid left-to-right, top-to-bottom structure. In contrast, both the Random and Halton sequences achieve a more uniform distribution across the grid, avoiding biases toward specific regions. Compared to random ordering, the Halton sequence is more robust to gaps; for instance, in the random ordering example below, the 17th token is sampled early, while its neighboring tokens are selected much later, leading to a less balanced and structured prediction process.



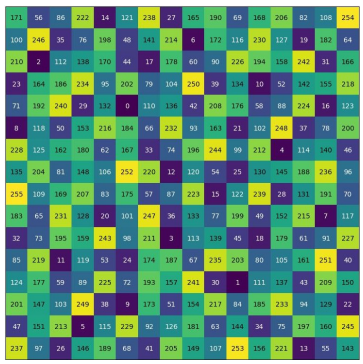
(a) Raster Scan in 2D



(b) Spiral Sequence in 2D



(c) Random Sequence in 2D



(d) Halton Sequence in 2D

Figure 8: Visualization of different sequence orderings in 2D.

## C Hyper-parameters

In Table 8, we provide the hyper-parameters used for training our class-to-image and class-to-video models on CIFAR-10, ImageNet, UCF101 and NuScenes datasets. The training process differs primarily in the number of steps and batch sizes, reflecting the scale of each dataset. A cosine learning rate decay is applied only for the last 10% of the iterations and we use 2,500 warmup steps to stabilize early training. We incorporate gradient clipping (norm = 1) to prevent exploding gradients and classifier-free guidance (CFG) dropout of 0.1 for better sample diversity. The CIFAR-10 model applies horizontal flip augmentation, while no data augmentation is used for ImageNet. Both models are trained using bfloat16 precision for computational efficiency. These hyper-parameters were chosen to ensure stable training while balancing efficiency and performance across different datasets. Finally, we sweep our model size according to Table 7.

## D Inference Speed Analysis

Given the low number of sampling steps ($\leq$ 32), our method is significantly faster than auto-regressive models that rely on long sequences, which is inherited feature from Besnier et al. (2025). For instance, even with optimizations such as KV-cache, models like LlamaGEN-XL require 576 steps and remain slower.

| Model | Parameters | GFLOPs | Heads | Hidden Dim | Width |
|---|---|---|---|---|---|
| BIGFix-Tiny | 24M | 4.0 | 6 | 384 | 6 |
| BIGFix-Small | 50M | 9.0 | 8 | 512 | 8 |
| BIGFix-Base | 143M | 25.0 | 12 | 768 | 12 |
| BIGFix-Large | 480M | 83.0 | 16 | 1024 | 24 |
| BIGFix-XLarge | 693M | 119.0 | 16 | 1152 | 28 |

Table 7: Transformer model configurations for $16 \times 16$ input size.

| Condition | Cifar10 | ImageNet | UCF101 | NuScenes |
|---|---|---|---|---|
| Training steps | $400k$ | $1.5M$ | $410k$ | $410k$ |
| Batch size | 128 | 256 | 256 | 8 |
| Learning rate | $1 \times 10^{-4}$ | $1 \times 10^{-4}$ | $1 \times 10^{-4}$ | $2 \times 10^{-4}$ |
| Weight decay | 0.03 | 0.03 | 0.03 | 0.03 |
| Optimizer | AdamW | AdamW | AdamW | AdamW |
| Momentum | $\beta_1 = 0.9, \beta_2 = 0.999$ | $\beta_1 = 0.9, \beta_2 = 0.999$ | $\beta_1 = 0.9, \beta_2 = 0.999$ | $\beta_1 = 0.9, \beta_2 = 0.999$ |
| Lr scheduler | Cosine | Cosine | Cosine | Cosine |
| Warmup steps | 2500 | 2500 | 2500 | 2500 |
| Gradient clip norm | 1 | 1 | 1 | 1 |
| CFG dropout | 0.1 | 0.1 | 0.1 | 0.1 |
| dropout | 0.1 | 0.1 | 0.1 | 0.1 |
| Data aug. | Horizontal Flip | No | No | No |
| Precision | bf16 | bf16 | bf16 | bf16 |

Table 8: Hyper-parameters used in the training of class-to-image and class-to-video models.

On an NVIDIA A100 GPU, our BIGFix-Large model generates a single image with classifier-free guidance (CFG) in 0.25 seconds, making it 2.86× faster than LlamaGEN-XL optimized with vLLM. To the best of our knowledge, masked image modeling (MIM) approaches do not exploit KV-cache during sampling yet.

## E Video Synthesis

To test our method beyond image synthesis, we explore class-to-video on UCF101 (Soomro et al., 2012) dataset and img-to-video on NuScenes (Caesar et al., 2020). As demonstrated previously in Table 1, introducing self-correction substantially improves the quality of generated video samples, mirroring the results of our image synthesis experiments.

In Table 9, we compare BIGFix-Large against state-of-the-art video generation models on UCF101. Despite using a smaller model (480M parameters) and fewer training steps (32), our approach achieves a competitive FVD of 242.16. Performance is limited by our reliance on the open-weight OmniTokenizer, which yields a higher rFVD (42) compared to closed-source tokenizers used by MAGVIT (rFVD 25) and MAGVIT2 (rFVD 8.62). This highlights that while our framework is efficient, generative quality remains constrained by tokenization quality and scale. These results demonstrate the potential of BIGFix but indicate that further improvements would require larger models or more advanced tokenizers.

| Model | #Para. | Train (steps) | Steps | FVD↓ |
|---|---|---|---|---|
| MAGVIT Yu et al. (2023)† | 306M | - | 12 | 76 |
| MAGVIT2 (Yu et al., 2024)† | 307M | - | 24 | 58 |
| LARP (Wang et al., 2025)† | 632M | - | 1024 | **57** |
| OmniTokenizer (Wang et al., 2024a) | 650M | 4M | 1280 | 191.14 |
| OmniTokenizer (Wang et al., 2024a) | 227M | 4M | 1280 | 313.14 |
| BIGFix-Large | 480M | 410k | 32 | 242.16 |

Table 9: UCF101 results. † use closed source tokenizer.

In Table 10, we report results on the NuScenes dataset. Despite using a relatively small model (BIGFix-Large, 480M parameters) and limited training time (15 hours), our method achieves a competitive FVD of 290.5. Compared to much larger models, such as GenAD (2.7B parameters, FVD 184.0) or Vista (2.5B parameters, FVD 89.4), BIGFix demonstrates strong efficiency and highlights the potential for scaling to larger models and longer training to achieve a lower FVD.

| **Model** | **#Para.** | **Train (h)** | **Steps** | **FVD** |
|---|---|---|---|---|
| GenAD (Zheng et al., 2024)‡ | 2.7B | 2 kh | – | 184.0 |
| Vista (Gao et al., 2024)‡ | 2.5B | 1.7 kh | 100 | 89.4 |
| DriveDreamer (Wang et al., 2024b)† | 1.45B | 15 h | – | 452.0 |
| WoVoGen (Lu et al., 2024) | - | 15 h | – | 417.7 |
| Drive-WM (Wang et al., 2024c)† | 1.45B | 15 h | 50 | 122.7 |
| BIGFix-Large | 480M | 15 h | 32+8 | 290.5 |

Table 10: NuScenes results. † use pre-trained weight from SD (Rombach et al., 2022). ‡ Zero-shot FID.

## F Additional qualitative results

In Figure 9, Figure 10, Figure 11, Figure 12, and Figure 13, we present qualitative results from BIGFix-Large. Without any cherry-picking, but with classifier-free guidance (CFG), we demonstrate that our model is capable of generating realistic and diverse images. Maintaining intricate details on both the objects and the background, using only 24 steps.

## G Limitations

Like other auto-regressive approaches, BIGFix requires a fixed token unmasking schedule defined at training and maintained at inference, which limits flexibility during prediction. While not an inherent limitation of the method, our experiments were restricted to models below 1 billion parameters to keep computational costs manageable. Extending BIGFix to large-scale image and video generation remains future work.

(a) Tree frog **031**

(b) Chicken **008**

Figure 9: Random samples from our BIGFix-Large model. $\alpha = 0.2$, $cfg = 4.0$ and 24 steps.

(a) LadyBug **301**

(b) Macaw **88**

Figure 10: Random samples from our BIGFix-Large model. $\alpha = 0.2$, $cfg = 4.0$ and 24 steps.

(a) Axolotl **29**

(b) Bald Eagle **22**

Figure 11: Random samples from our BIGFix-Large model. $\alpha = 0.2$, $cfg = 4.0$ and 24 steps.

(a) Rock crab **119**

(b) Great white shark **002**

Figure 12: Random samples from our BIGFix-Large model.
$\alpha = 0.2$, $cfg = 4.0$ and 24 steps.

(a) Bobsleigh **450**

(b) Norwegian Elkhound **174**

Figure 13: Random samples from our BIGFix-Large model.
$\alpha = 0.2$, $cfg = 4.0$ and 24 steps.