Sparse-LaViDa: Sparse Multimodal Discrete Diffusion Language Models

Supplementary Material

7. Additional Technical Details

7.1. Formulation of Discrete Diffusion Models

In this section, we include an overview of the standard formulation of Masked Diffusion Models (MDMs) that are widely adopted by literature [32, 33, 38, 51, 69].

Given a sequence X_0 consisting of discrete tokens $[X_0^1, X_0^2, \ldots, X_0^L]$, where L is the sequence length, the forward process $q(X_t|X_s)$ gradually mask the sequence and convert clean tokens to a special mask token $[M]$ over the continuous time interval $[0, 1]$, with $1 \geq t \geq s \geq 0$. At $t = 1$, the sequence $X_1 = [X_1^1, X_1^2, \ldots, X_1^L]$ consists of only masked token. This forward process is formally defined as

$$q(X_t^i|X_s^i) = \begin{cases} \text{Cat}(X_t^i; \mathbf{M}), & \text{if } X_s^i = [M] \\ \text{Cat}(X_t^i; \frac{1-t}{1-s}\mathbf{X_s^i} + \frac{t-s}{1-s}\mathbf{M}), & \text{if } X_s^i \neq [M], \end{cases} \tag{2}$$

where $\text{Cat}(\cdot)$ denotes a categorical distribution, and $\mathbf{M}, \mathbf{X_0^i}, \mathbf{X_s^i} \in \mathbb{R}^{|V|}$ are probability vectors, and $|V|$ is the vocabulary size. In particular, $\mathbf{M}$ is a one-hot vector corresponding to the special token $[M]$. This forward process yields the following marginal distribution:

$$q(X_t^i|X_0^i) = \text{Cat}(X_t^i; (1-t)\mathbf{X_0^i} + t\mathbf{M}). \tag{3}$$

Prior works [51] show that the posterior of the reverse process $p(X_s|X_t, X_0)$ has the following form:

$$p(X_s^i|X_t^i, X_0^i) = \begin{cases} \text{Cat}(X_s^i; \mathbf{X_t^i}), & \text{if } X_s^i \neq [M] \\ \text{Cat}(X_s^i; \frac{t-s}{t}\mathbf{X_0^i} + \frac{s}{t}\mathbf{M}), & \text{if } X_s^i = [M]. \end{cases} \tag{4}$$

At inference, $\mathbf{X_0}$ is not known, so we replace $\mathbf{X_0^i}$ with the neural network prediction $p_\theta(X_0^i|X_t)$, which gives the following emprical sampling process:

$$p_\theta(X_s^i|X_t) = \begin{cases} \text{Cat}(X_s^i; \mathbf{X_t^i}), & \text{if } X_s^i \neq [M] \\ \text{Cat}(X_s^i; \frac{t-s}{t}p_\theta(X_0^i|X_t) + \frac{s}{t}\mathbf{M}), & \text{if } X_s^i = [M]. \end{cases} \tag{5}$$

Sampling process. To sample a sequence of clean tokens X_0, we start with a fully masked sequence X_1, where $X_1^1 = \cdots = X_1^L = [M]$. We discretize the continuous time interval $[0, 1]$ into discrete timesteps $0 = t_0 < t_1 < \cdots < t_K = 1$, and iteratively sample $X_{t_{k-1}} \sim p_\theta(X_{t_{k-1}}|X_{t_k})$ using Equation 5. We start with $k = K$ and end when we obtain a mask-free sequence X_0. At each step, we assume $p_\theta(X_{t_{k-1}}|X_{t_k})$ factorizes as $\prod_{i=1}^L p_\theta(X_{t_{k-1}}^i|X_{t_k})$, following previous works [38, 45, 51].

Training process. At each training step, given a clean sequence X_0, we sample a random timestep $t \in [0, 1]$ and obtain $X_t \sim q(X_t|X_0)$ through the forward process defined in Equation 3. This gives us a partially masked sequence. The loss is then computed using Equation 1 from Section 2.1.

We highlight that Sparse-LaViDa does not fundamentally change these formulations. Rather, it proposes an equivalent but efficient parameterization by introducing a sparse representation for the partially masked sequence X_t.

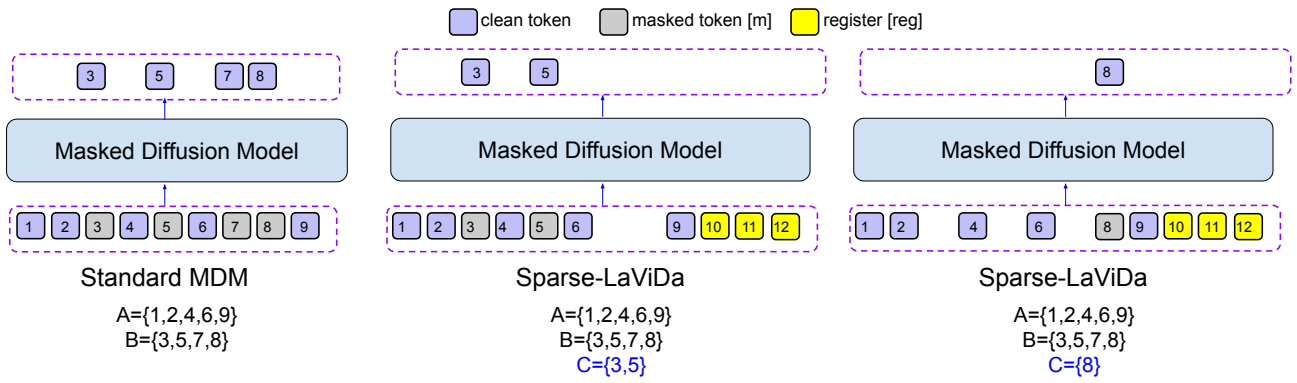


Figure 7. **Sparse-Representation of Partially Masked Sequence.** Given a partially masked sequence, we can partition its tokens into two subsets A and B, where A consists of clean tokens and B consists of masked tokens. The standard MDM (Left) need to pass all $|A|+|B|$ tokens. By contrast, Sparse-LaViDa (Middle and Right) only need to pass $|A|+|C|+m$ tokens, where C is a subset of B and m is the number of register tokens (set to 3) in this case.

7.2. Register Tokens

In this section, we discuss the detail implementation of register tokens. Given a partially masked sequence X_t of length L, which can be partitioned into two subsets X^A, X^B, where A consists of clean token positions and B consists of masked positions. Let $K = \{1, 2..L\}$ be the set of all token positions, we have that $A \cup B = K$ and $A \cap B = \emptyset$. In the standard parameterization, we need to pass in $L = |A| + |B|$ tokens, among which $|B|$ are masked tokens, to the model, even if we are only interested in obtaining the prediction at a subset of masked position $C \subset B$ at the current diffusion step.

The sparse parameterization introduces an extra set of m register tokens $R = [R^1..R^m]$. Concretely, these are achieved by adding special tokens "[reg]" that is similar to mask token "[M]" to the vocabulary. All m register tokens are represented with the same special token "[reg]" at the tokenization level, but their positional ids are $L+1, L+2...L+m$ respectively, leading to different rope embeddings and different beahvior in the attention process. In this setup, the total token count is $|A| + |C| + m$. When $|C| \ll |B|$, we can achieve considerable speedup from reduced sequence length.

We visualize this design in Fig. 7. Sparse-LaViDa allows us to flexibly truncate masked tokens depending on which token prediction we want to obtain. When, $C = B$, Sparse-LaViDa reduces to the standard MDM parameterization. Hence, Sparse-LaViDa is a generalization to the standard MDM.

For simplicity, we use a simple partition A, B to elaborate the design of register tokens and sparse representation, as opposed to more complex partition $C_1...C_{k-1}$ that depends on sampling steps, which are used in Sec. 3.2 of the main text and the following Sec. 7.3 of the appendix.

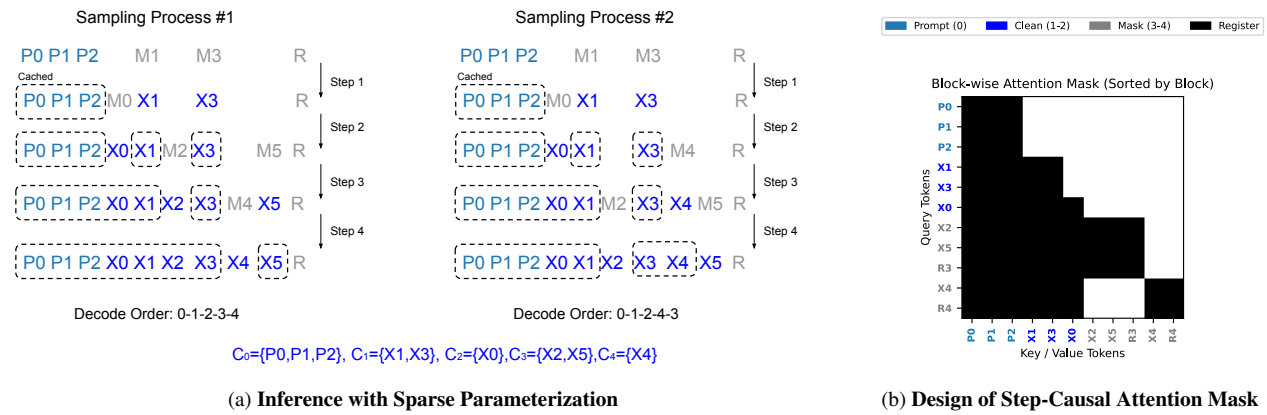
7.3. Step-Causal Attention Masks

In this section, we provide a detailed account of step-causal attention mask, with particular focus on how the design of step-causal attention mask aligns with the inference process with KV caching.

Inference Time. Suppose we have S prompt tokens and generate a clean response with L tokens through k diffusion steps. We can naturally obtain a k-partition of the L response tokens $C_1..C_k$ depending on when they are unmasked during the k diffusion steps. We also define C_0 as the set of prompt tokens, which are never masked.

Fig. 8a illustrates an example, which consists of $S = 3$ prompt tokens $P_0...P_2$ and $L = 6$ response tokens $X_0...X_5$. We also assume we have $k = 4$ diffusion steps and there is only one register token R. As illustrated in the figure, sampling process #1 induces the partition $C_0 = \{P_0, P_1, P_2\}$, $C_1 = \{X_1, X_3\}$, $C_2 = \{X_0\}$, $C_3 = \{X_2, X_5\}$, $C_4 = \{X_4\}$, where tokens in C_1, C_2, C_3, C_4 are unmasked at the first, second, third, and fourth diffusion steps respectively. After a token is generated at ith step, it is passed back to the model in $(i+1)$th step and added to the cache after the model forward call. This is also illustrated in the figure (black dashed box). For example, tokens in $C_1 = \{X_1, X_3\}$ is added to the KV cache after step 2.

At the ith diffusion step, the KV cache consist of all tokens in C_j where $j < i - 1$. The input of the model consist of the previously generated tokens in C_{i-1}, the set of masked tokens to be decoded at the current step C_i, and the register token R.



(a) **Inference with Sparse Parameterization** (b) **Design of Step-Causal Attention Mask**

Figure 8. **Connection between the Diffusion Sampling Process and the Step-Causal-Attention-Mask.** (Left) Given prompt tokens $P_0 ... P_2$ and response tokens $X_0 ... X_5$, we partition the response tokens with $C_1 ... C_4$ and assign prompt tokens to C_0. Both 0-1-2-3-4 and 0-1-2-4-3 are viable sampling order of tokens in a 4-step diffusion process. We use M to represent mask tokens and R represent registers. (Right), for the two sampling process, we can simulate step 3 of both processes simultaneously via a special attention mask.

Since C_{i-1} are to be added to the KV cache, we prevent the queries of C_{i-1} to interact with keys and values of C_i and R as discussed in Sec. 3.3.

Taking step 3 as an example, the KV cache consists of tokens in C_0, C_1 (boxed), which are P_0, P_1, P_2, X_1, X_3. The input consists of tokens in C_2, which is just X_0, and masked tokens in C_3, denoted by M_2, M_5, as well as a register token R. In this step, X_0 from C_2 cannot attention to M_2, M_5, R, (color-coded as gray), while M_2, M_5, R can attend to all other tokens.

Note that while the partition $C_0, C_1 ... C_4$ derives from the sampling process $\#1$, its decoding order $0-1-2-3-4$ does not uniquely leads to this partition. For instance, sampling process $\#2$ with decoding order $0-1-2-4-3$ also has the same partition (Fig. 8a, Right). Moreover, we note that at step 3 of sampling process $\#2$, the KV cache is exactly the same as sampling process $\#1$. Among the input tokens, X_0 from C_2 has exactly the same behavior as sampling process $\#1$ because of the attention mask. The only differences lies in the input of masked tokens. In sampling process $\#1$, we have M_2, M_5 since at this step we want to unmask C_3. In sampling process $\#2$, we have M_4 since at this step we want to unmask C_4. This observation gives us the opportunity to parallelize the training by simulate sampling process $\#1$ and sampling process $\#2$ simultaneously through a special attention mask.

Training time. We can merge sampling process $\#1$ and sampling process $\#2$ by constructing the sequence $P_0, P_1, P_2, X_1, X_3, X_0, X_2, X_5, R_3, X_5, R_4$, where R_3, R_4 are duplicates of the same register token R. We assign a step-causal attention mask for tokens $P_0, P_1, P_2, X_1, X_3, X_0$, as they have exactly the same attention pattern in both sampling steps. To accommodate the difference of masked token inputs where sampling process $\#1$ has M_2, M_5, R and sampling process $\#2$ has M_4, R we concatenate the sequences to form M_2, M_5, R_3, M_4, R_4 and design a block-diagonal attention mask that ensures input tokens from one sampling process cannot attend to tokens from another same sampling process (bottom-right section of Fig. 8b). We denote two copies of R as R_3, R_4 since M_2, M_5 comes from partition C_3 and M_4 comes from partition C_4. This design of attention mask is illustrated in Fig. 8b.

The training-time attention mask is formally defined in Algorithm 1. In the above example, to simulate sampling process $0-1-2-3-4$ and $0-1-2-4-3$, we set the number of clean blocks $M = 2$ (i.e. C_0, C_1) and the number of masked blocks $N = 2$ (i.e. C_3, C_4). We assign the block number to tokens based on which of the partition $C_0 ... C_4$ they belong to. The duplicated register tokens R_3, R_4 are assigned with block numbers 3 and 4 respectively.

Algorithm 1 Step-Causal Attention Mask Construction

Require: Block assignments $A \in \mathbb{Z}^{L+S}$, number of clean blocks M, number of masked blocks N

Initialize mask $\in \{0,1\}^{(L+S)\times(L+S)}$ to zeros
for $q_i = 0$ **to** $L+S-1$ **do**
  $q_b \leftarrow A[q_i]$ ▷ Block index of query token
  for $k_j = 0$ **to** $L+S-1$ **do**
    $k_b \leftarrow A[k_j]$ ▷ Block index of key token
    if $q_b = 0$ **then** ▷ Prompt token: full attention to prompt
      if $k_b \leq 0$ **then**
        $\text{mask}[q_i, k_j] \leftarrow 1$
      end if
    else if $1 \leq q_b \leq M$ **then** ▷ Clean token: attend up to its block
      if $k_b \leq q_b$ **then**
        $\text{mask}[q_i, k_j] \leftarrow 1$
      end if
    else if $M+1 \leq q_b \leq M+N$ **then** ▷ Masked tokens: attend to prompt/clean tokens, or those in the same block
      if $(k_b \leq M) \ \vee \ (k_b = q_b)$ **then**
        $\text{mask}[q_i, k_j] \leftarrow 1$
      end if
    end if
  end for
end for
return mask

8. Additional Experiment Details and Results

8.1. Data pipeline

Our training dataset consist of the following tasks.

- *A: Text-to-Image Pairs.* We source data from LAION-2B [53] and COYO-700M [6]. We additionally include BLIP3o-60k [10], and ShareGPT4o-Image [9]. Each dataset is heavily filtered to remove NSFW prompts, low CLIP scores [49], low aesthetic scores [52], and low-resolution images following LaViDa-O's pipeline. We include all data from BLIP3o-60k and ShareGPT4o-Image, and select highest-quality samples from LAION and COYO based on CLIP scores and aesthetic scores. The final data mix consist of 20M images. Unlike LaViDa-O, we did not include SA-1B [25], JourneyDB [56] because of quality issues. Specifically, human faces in SA-1B are blurred, while JourneyDB images have many artifacts even after heavy filtering.
- *B: Image-level Understanding Data.* We include MAmmoth-VL [18], and VisualWebInstruct [22].
- *C: Region-level Understanding Data.* We include GranD [50] and RefCOCO [23].
- *D: Image Editing Data.* We include ShareGPT4o-Image [9], GPT-Edit-1.5M [59], and the image editing subset of UniWorld-V1 [20].

8.2. Training Setup

We perform SFT training on the LaViDa-O weights using our proposed sparse-parameterization and step-causal attention mask. The hyper-parameters largely followed the SFT stage of LaViDa-O. We document the details of training setup and relevant hyperparameters in Tab. 9. The main experiments is conducted with 64 A100 GPUs across 8 nodes. The training takes 5 days in total, which is 15% of the LaViDa-O's training budget (34.2 days from scratch).

8.3. Additional Results on Object Grounding

As shown in Fig. 6, one of the advantages of Sparse-LaViDa is that it does not employ a block-causal attention mask like block-diffusion, making it capable of tasks that requires bi-directional context such as inpainting, object grounding through infilling, and constrained text generation just like vanilla MDM.

We report quantitative results on object grounding tasks in Table 10. Overall, Sparse-LaViDa achieves comparable performance as the LaViDa-O baseline. Since all coordinates are generated in a single step, the two methods have identical

Table 9. **Training configurations of Sparse-LaViDa.** We report the relevant hyperparameters for training, including the learning rate, number of training steps, optimizer setup, image resolution for understanding and generation tasks.

	SFT
Learning Rate	2×10^{-5}
Steps	100k
β_1	0.99
β_2	0.999
optimizer	AdamW
Loaded Parameters	10.4B
Trainable Parameters	10.4B
Und. resolution	384 $\times\{(1,3),(2,2)\}$
Gen. resolution	1024

inference speed because Sparse-LaViDa cannot benefit from token truncation in this setup.

Table 10. **Precision@0.5 on RefCOCO, RefCOCO+, and RefCOCOg REC tasks.**

Model	**RefCOCO** ↑			**RefCOCO+** ↑			**RefCOCOg** ↑	
	val	testA	testB	val	testA	testB	val	test
SegLLM-7B[58]	90.0	92.1	86.2	82.2	85.5	76.1	83.9	85.9
Qwen2.5-VL-7B [3]	90.0	92.5	85.4	84.2	89.1	76.9	87.2	87.2
GroundingDINO [35]	90.6	93.2	88.2	88.2	89.0	75.9	86.1	87.0
InternVL3-8B [77]	92.5	94.6	88.0	88.2	92.5	81.8	89.6	90.0
LaViDa-O	91.9	94.6	88.4	87.4	91.7	82.2	89.5	89.8
Sparse-LaViDa	92.3	94.9	89.1	87.4	92.5	81.8	89.6	89.9

8.4. Addtional Results with Larger Batch Size

In Table 11, we report results at a larger batch size for T2I tasks. Results show that LaViDa-O consistently outperforms baselines at varying batch sizes.

Bs.	**LaViDa-O** ↑	**Sparse-LaViDa** ↑
1	2.82	5.52
2	3.85	7.26
4	4.24	8.65

Table 11. **Throughput Analysis. We report images/min. (Higher is better)**

8.5. Additional Qualitative Comparisons

In Figure 9, we provide side-by-side qualitative comparisons of Sparse-LaViDa and the LaViDa-O baseline on text-to-image generation and image editing tasks. As shown in the figure, Sparse-LaViDa achieves comparable generation quality while offering a speedup of $1.95\times$ on text-to-image generation and a speedup of $2.83\times$ on image editing.

9. Limitations

Despite the promising results of Sparse-LaViDa, it has several limitations. First, the speedup only benefits long sequence generation such as text-to-image generation, image-editing, or visual math problem-solving with long reasoning chains. It does not improves the speed of short QA tasks or object grounding tasks, where the number of output tokens are very small.

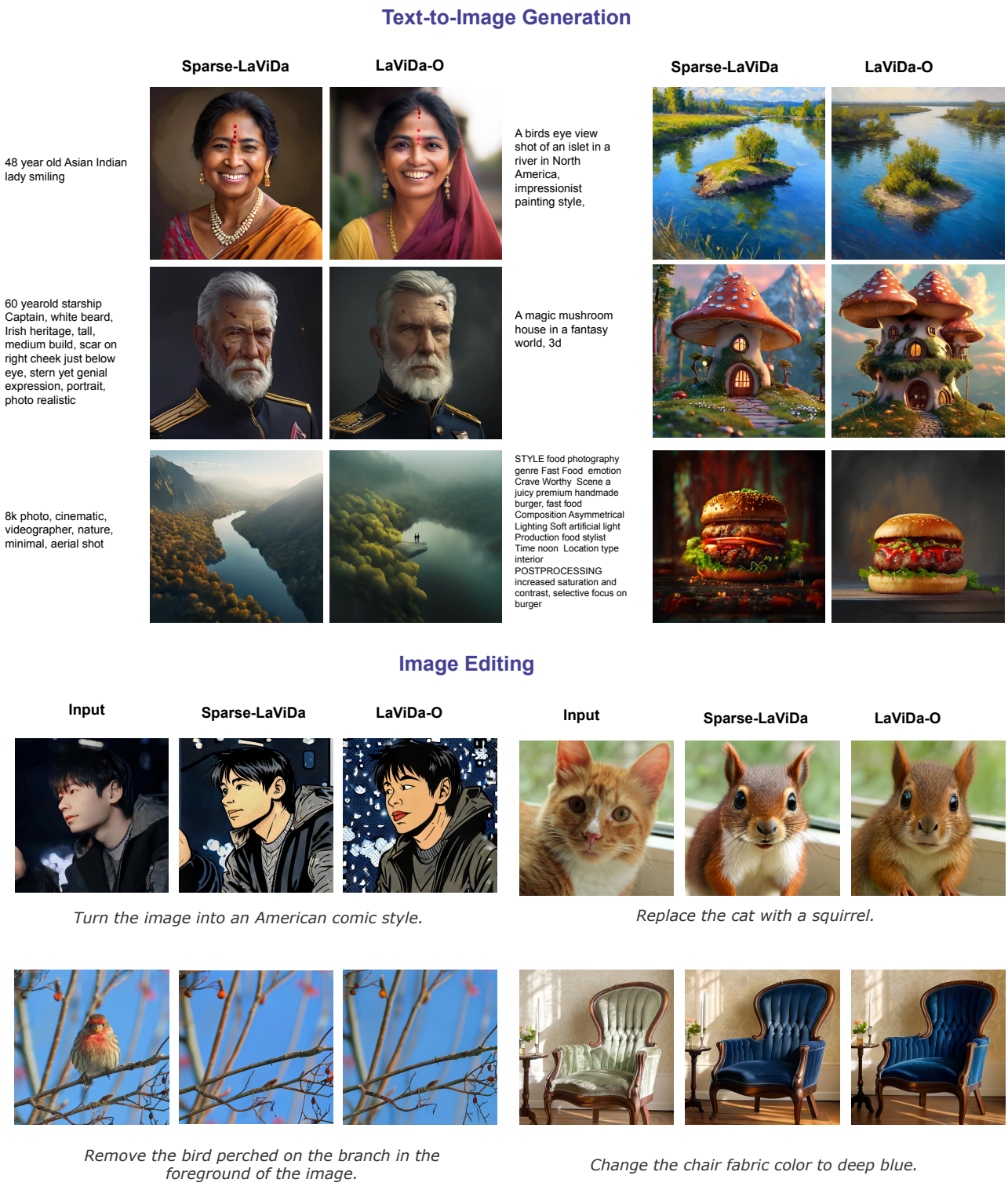


Figure 9. **Qualitative Comparisons of Sparse-LaViDa and LaViDa-O baseline.** We show qualitative results of text-to-image generation and image editing results of Sparse-LaViDa and LaViDa-O baseline.Sparse-LaViDa achieves a speedup of $1.95\times$ on text-to-image generation and a speedup of $2.83\times$ on image editing, while maintaining comparable visual quality

We emphasize that Sparse-LaViDa will benefit any tasks involving image generation, since an image is represented by 4096 VQ tokens.

Second, our model inherits many limitations from the base model LaViDa-O, such as hallucination in responses. Further, we observe the same pixel shift issue discovered by LaViDa-O, where image editing results may have subtle changes in un-edited regions. We hope this issue will be addressed by future foundational models.

Lastly, while we have demonstrated Sparse-LaViDa is an efficient post-training technique, in principle it should be general to all stages including pretraining. It would be interesting for researchers with more compute resources to apply this to different stages of training.