Multi-Scale Local Speculative Decoding for Image Generation

Supplementary Material

6. Primer on Tar

We provide a concise description of the Tar architecture and the default parameters used to obtain the results reported in this paper. For additional details, we refer the reader to the original publication [13].

Architecture For the purposes of this work, Tar consists of two main components:

1. A Multimodal Large Language Model (MLLM) that processes the input prompt and generates a conditioning sequence,
2. A generative detokenizer that maps the conditioning sequence to a VQ-VAE token sequence, which is then decoded to pixel space by the VQ-VAE decoder.

The MLLM is fine-tuned from QwenVL [1] and extended to predict visual tokens. It is trained to output sequences of three different lengths: 81, 169, and 729 tokens. Each length corresponds to a progressively stronger conditioning signal for the detokenizer.

The autoregressive generative detokenizer (AR-DTok) is based on the LlamaGen [40] model, fine-tuned to use the output of the MLLM as conditioning. Conditioning is implemented by pre-filling the sequence with one of the desired lengths (*e.g.*, 81, 169, or 729). Importantly, the AR-DTok model operates in the latent space of a VQ-VAE [7, 44], which performs $16\times$ down-sampling along both spatial dimensions, resulting in sequence lengths of 256, 1024, and 4096 tokens for the resolutions 256p, 512p and 1024p respectively.

Sampling We now describe the sampling procedure. For the MLLM, we use the default configuration: $\text{top}_k = 1200$, $\text{top}_p = 0.95$, the temperature $\tau_{\text{logits}} = 1.0$ (different from the relaxed acceptance threshold τ defined in Equation 5 in the main paper) and set the sequence length to 729. The absolute latency for generating this conditioning sequence is approximately 17 seconds. Note that this value is not included in our latency analysis. This conditioning sequence is then used to sample from the AR-DTok model. For AR-DTok, we set: $\text{top}_k = 0$, $\text{top}_p = 1.0$ and the temperature $\tau_{\text{logits}} = 1.0$ (*i.e.* sampling from the full distribution of logits). Additionally, we apply classifier-free guidance with a scale of 4.0, we use an empty sequence for the negative prompt.

Sampling from AR-Dtok takes on average 5s, 18s and 80s for each resolution respectively, see Table 2 for an overview. Given that sampling the conditioning sequence takes an average of 17s, it reinforces that MuLo-SD's best setting is the $4\times$ case i.e., going from 256p to 1024p. In this scenario, the total latency is largely dominated by the AR-DTok decoding time, and accelerating the visual token generation will lead to substantial speedups.

Table 2. Summary of AR-Dtok configurations from Tar [13].

Resolution	Seq. Length	Latency
256p	256	5s
512p	1024	18s
1024p	4096	80s

7. MuLo-SD

Method We describe the algorithm of MuLo-SD in Algorithm 1, a full description can be found in Section 3.2 and Section 3.3 of the main paper. The step numbers ① - ⑦ are a reference to the schematic representation in Figure 2 of the main paper. We present speculative decoding and LANTERN in the same style as our main method schema in Figure 6. For a detailed description of their algorithm, see Section 3.1 in the main paper.

Implementation Details The drafter model consists of three main components: an autoregressive model, an up-sampler, and optionally a down-sampler. The autoregressive model is set as AR-DTok @ 256p and remains fixed throughout all experiments.

Learnable Up/Down-Samplers The up- and down- sampler are implemented as lightweight convolutional networks with residual blocks, and use pixel-shuffle to perform the correspondent resampling operation. We progressively train the up- and down- sampler for the $2\times$ and the $4\times$ settings.

In the $2\times$ setup, each module contains approximately 20M learnable parameters. These modules are trained on the LAION-COCO-Aesthetic [24] dataset for 150k steps with a batch size of 32, using the AdamW optimizer with learning rate of $3\text{e}{-4}$. We use a combination of losses for training: MSE, LPIPS, commitment loss, and discriminator loss. The overall objective is defined as:

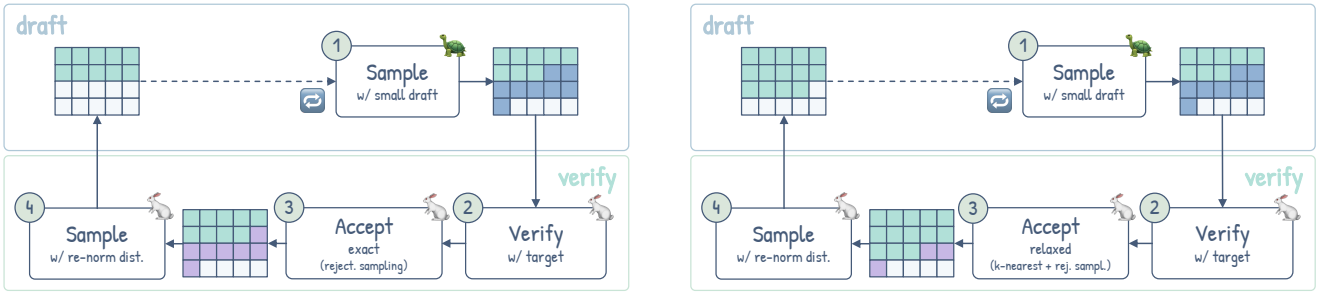
$$\mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{MSE}} + \mathcal{L}_{\text{LPIPS}} + \mathcal{L}_{\text{commit}} + \lambda_{\text{GAN}} \cdot \mathcal{L}_{\text{GAN}}. \quad (9)$$

For the first 20k iterations, the up- and down- samplers are trained without the discriminator loss; this component is introduced afterward. The discriminator follows the standard PatchGAN design [19], consisting of three convolutional

Algorithm 1: Multi-Scale Local Speculative Decoding

Input: The target model M_p at scale s_p, the draft model M_q at scale s_q, the up- and down-sampler U_r and D_r with a resampling factor of $r = s_p/s_q$, the initial sequence $x_0, \ldots, x_t$, draft sequence length L, the target sequence length T, the cardinality of latent neighborhood k, the TVD threshold δ, the probability mass threshold τ and l the local neighborhood radius.

Initialize: $n \leftarrow t$;
while $n < T$ **do**
  (7) In parallel, down-sample the n tokens to obtain prefix for draft model at scale s_p: $y_{1:n/r} = D_r(x_{1:n})$;
  for $t = 1, \ldots, L/r$ **do**
    (1) In sequence, sample tokens from draft model $\tilde{y}_t \sim M_q(x \mid y_0, \ldots, y_{n/r}, \tilde{y}_1, \ldots, \tilde{y}_{t-1})$;
  (2) In parallel, up-sample the L/r tokens to obtain L draft tokens at scale s_q: $\tilde{x}_{n:n+L} = U_r(\tilde{y}_{n/r:(n+L)/r})$;
  (3) In parallel, compute L sets of logits:
  $M_p(x \mid x_0, \ldots, x_n), M_p(x \mid x_0, \ldots, x_n, \tilde{x}_1), \ldots, M_p(x \mid x_0, \ldots, x_n, \tilde{x}_1, \ldots, \tilde{x}_L)$;
  Initialize set of locally expanded rejected tokens $R_X \leftarrow \{\}$;
  for $t = 1, \ldots, L$ **do**
    Find the neighborhood $A_{k,\delta}(\tilde{x}_t)$;
    if $\sum_{x \in A_{k,\delta}(\tilde{x}_t)} M_p(x \mid x_0, \ldots, x_{n+t-1}) > \tau$ **then**
      (4) Accept: set $x_{n+t} \leftarrow \tilde{x}_t$;
    else
      (5) Reject: expand rejection to local neighborhood $N(t, l)$ around position t with radius l, $R_X \leftarrow R_X \cup N(t, l)$
  Sort indices in R_X;
  for $k \in R_X$ **do**
    (6) In sequence, sample rejected tokens from target model $x_{n+k} \sim M_p(x \mid x_0, \ldots, x_{n+k-1})$;
  Set $n \leftarrow n + L$
Output: $x_{t+1}, \ldots, x_T$



(a) Speculative Decoding (b) LANTERN

Figure 6. Overview of the standard speculative decoding [22] and LANTERN [20] methods. They are drawn in the same style as our main method figure for ease of comparison. Blue indicates draft tokens, green accepted tokens, purple rejected tokens, blank placeholder tokens. indicates sequential operations, parallel operations, a drawing discontinuity due to looping.

layers, and is trained from scratch using AdamW with a learning rate of $5\mathrm{e}{-4}$ with $\lambda_{GAN} = 0.25$.

Next, we add an additional block of convolutions for the $4\times$ case (resulting in approximately 30M parameters for each module). The up- and down- sampler are warm-started from the $2\times$ checkpoints and trained for another 150k steps. We use the same configurations, except a smaller batch size of 8 to fit into memory.

Inference Hyperparameter During inference, MuLo-SD introduces one primary hyperparameter: the acceptance threshold τ (see Equation 5). We perform a sweep over various values and ultimately fix $\tau = 1\mathrm{e}{-4}$, unless otherwise specified. Additionally, two other hyperparameters

control the probability aggregation from neighboring elements (see Step 4 of Fig. 2 of the main paper). These are set to $k = 1000$ and $\delta = 0.1$, and remain constant across all experiments. As discussed in the main paper, (k, δ) and τ play a similar role in relaxing the acceptance criterion; therefore, we primarily experiment with τ while keeping the others fixed.

Latency Analysis As discussed in the main paper, one of the key characteristics of MuLo-SD is the computational cost associated with the drafter model, which shares the same architecture as the target model. This allows us to estimate the theoretical speedup under different acceptance rates by considering the reduction in the number of function evaluations (NFE) throughout the model. The theoretical speedup S_T can be computed as follows. Using the notation from the main paper, let M_p denote the target model and M_q the drafter model, and define T_p and T_q as the sequence lengths for the target and drafter respectively, and let a denote the acceptance rate. Then:

$$S_T = \frac{T_p}{(1-a) \cdot T_p + T_q}. \tag{10}$$

We compute the empirical speedup by measuring the time required to generate 500 prompts from MS-COCO 2017 Validation Set [29] on a single NVIDIA A100 GPU with a batch size of 1. We break down the individual cost of each component in Figure 7 (top row). First, we observe that the cost of the drafter is fixed, regardless of the acceptance rate, since we always sample the same number of tokens from it. This eventually becomes the bottleneck in the 512p case, reducing the overall utility of our method. Conversely, at higher resolutions, the number of tokens generated by the target model is so large that the drafter's cost becomes negligible. This further reinforces the suitability of the 4× setting (256p → 1024p) for our model. As shown visually, almost all of the latency budget is spent sequentially sampling from either the target model or the drafter. This leads to two important considerations: (i) our proposed multi-scale speculative decoding introduces only a negligible overhead—about 5% and 3% for the 512p and 1024p settings, respectively; and (ii) there is still room for improvement by reducing the cost of the drafter and the verifier.

By introducing parallel decoding, we substantially reduce end-to-end latency (Figure 7, bottom row). As detailed in Section 3.4, we perform drafter and target sampling using a parallel decoding technique. Although these stages still dominate the latency budget, their impact is markedly reduced, yielding a larger overall speedup. Note the different scales on the x- and y-axes when comparing the top and bottom panels of the breakdown.

8. Experiments

Quantitative Evaluation We extend the quantitative results from the main paper by providing a graphical visualization of Table 1 in the main paper in Figure 8. It shows the pareto front of MuLo-SD and contextualizes its performance with competing methods such as ZipAR [16], EAGLE-2 [26] and LANTERN [20]. To create a pareto front, we vary the acceptance rate by sweeping different values for the relaxed acceptance threshold τ as defined in Equation 5 in the main paper. We can see that ZipAR dominates all other methods, with mostly unchanged perceptual quality compared to the reference, and only slight degradation to GenEval. Next comes our method MuLo-SD, which across the semantic alignment metrics dominates EAGLE-2 and LANTERN. When it comes to perceptual quality metrics, FID tends to suffer for MuLo-SD compared to other methods, and HPSv2 is sligthly better for MuLo-SD.

Qualitative Evaluation We supplement the qualitative results with additional visual comparisons. In Figure 10 we show samples from Tar-1.5B 512p and MuLo-SD for the 2× case (256p → 512p). In Figure 11 and Figure 12 we show additional results from Tar-1.5B 1024p and MuLo-SD for the 4× case (256p → 1024p). In both the 512p and 1024p cases, the acceleration comes at a slight cost in perceptual quality, howevere the semantic alignment is mostly unaltered. Finally, in Figure 13 we showcase the effect of sweeping the relaxed acceptance threshold τ on the output image quality, where the different τ used correspond to the points in Figure 8. The third column $\tau = 1e{-}4$ corresponds the setting reported in Table 1 in the main paper. Note that it seems like the best tradeoff between speedup and perceptual quality, where the rightmost column ($\tau = 1e{-}5$) shows the greatest speedup but largest degradation in quality, and the leftmost column ($\tau = 1e-3$) is the closest to the original but ends up slower for more complex prompts.

Note that for all qualitative figures, both in the main text and the supplementary, we use prompts sourced from the DPG-Bench [18] benchmark dataset. We report the IDs of the prompts used in Fig. 4 of the main paper (order top-bottom, left-right) and refer to the official code for the actual text: `78.txt`, `midjourney32.txt`, `COCOval2014000000580698.txt`, `stanford34.txt`, `5.txt`, `COCOval2014000000183648.txt`, `62.txt`, `diffusiondb10.txt`.

Additional Ablation We extend the ablation presented in Figure 5 (c) in the main paper. We show the effect of the

https://github.com/TencentQQGYLab/ELLA/tree/main/dpg_bench/prompts.

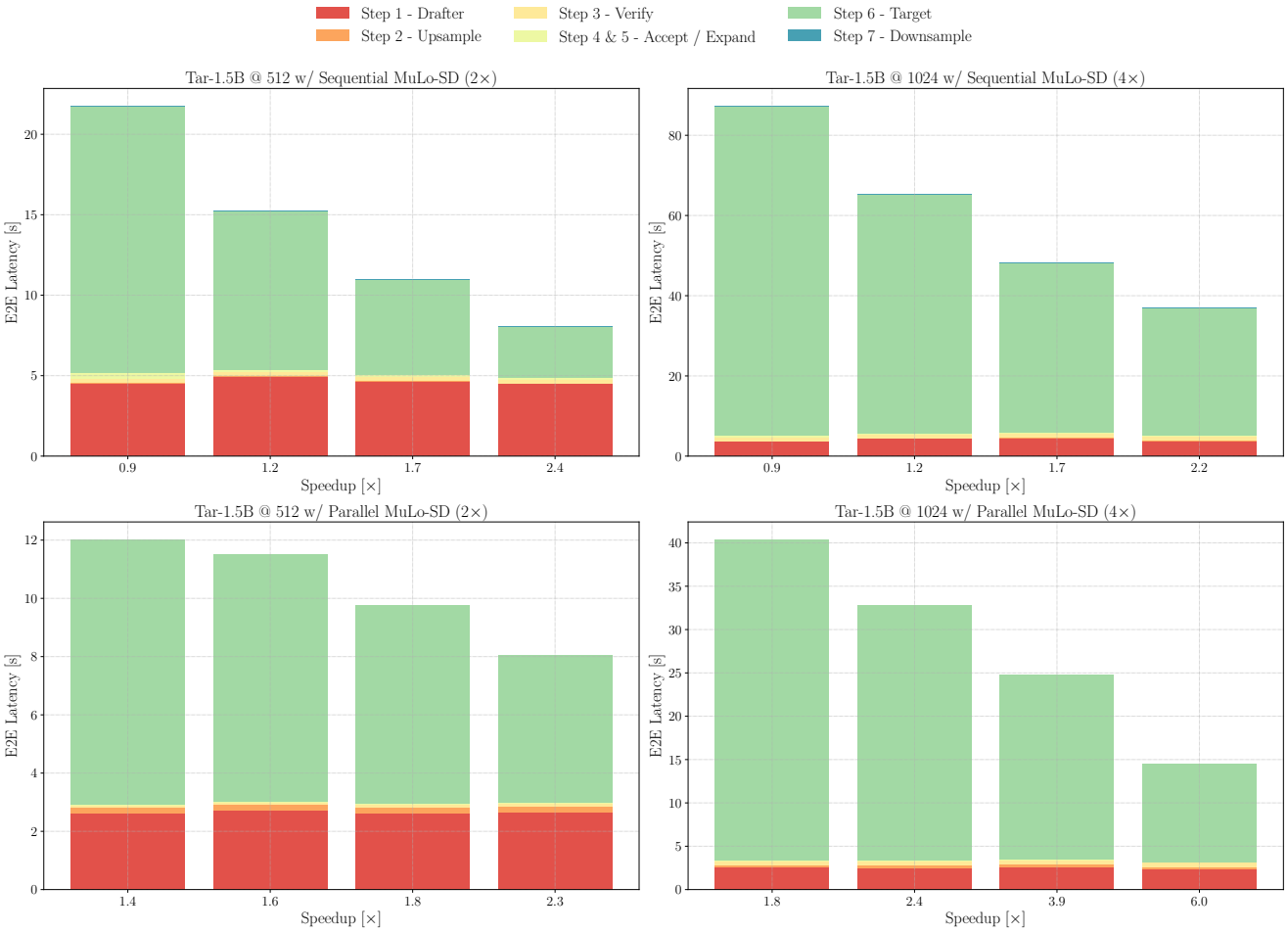


Figure 7. Breakdown of latency analysis. The figure illustrates the proportion of time spent in each step of our algorithm relative to the total latency. The step number in the legend refers to Fig. 2 in the main paper. The top row shows results *without* applying parallel decoding to Steps 1 and 6; the bottom row incorporates it. Note the different x- and y-axis scales across rows.

local expansion radius l in Figure 9, showcasing $l = 1$ and $l = 5$ in addition to our default value of $l = 3$ shown in the main paper. Similar to the other ablations in the main paper, the study is performed in the 2× case (256p → 512p). We can see that $l = 3$ provides the best boost in GenEval performance across the 1 - 1.5× speedup range of interest. It is closely followed by $l = 1$, with $l = 5$ lagging behind. We expect the optimal value for l to depend heavily on the resolution, as large resolution will benefit from larger radii, and conversely smaller resolution will suffer from larger radii as it will lead to high rejection rate even for permissive relaxed acceptance thresholds τ. We anyway use $l = 3$ for the 1024p case based on the result of this ablation due to lack of computational resource and time to ablate the parameter on the higher resolution.

Discussion on LANTERN As discussed in the main paper, porting LANTERN [20] (and EAGLE-2 [26]) to Tar [13] proved significantly more challenging—yielding worse performance—than what was originally reported for LlamaGen [40]. In this section, we detail our training procedure and provide additional justifications for the observed results. We follow the original training script from the LANTERN codebase. The drafter consists of a single transformer layer and is trained using activations from the last transformer block of the target model (*i.e.* before the final softmax fully connected layer).

The training objective combines two losses: (i) standard cross-entropy loss for next-token prediction, and (ii) an L1 loss to regress the hidden state of the teacher (*i.e.* the target model). The overall loss is weighted using the configuration reported in LANTERN, with $\lambda_{L1} = 0.1$ for the regression term. We train the drafter on a subset of the LAION-COCO-Aesthetic dataset [29], using 100k samples for training and reserving 1k samples for evaluation. Since LANTERN does not specify the dataset used to train the drafter, a one-to-one

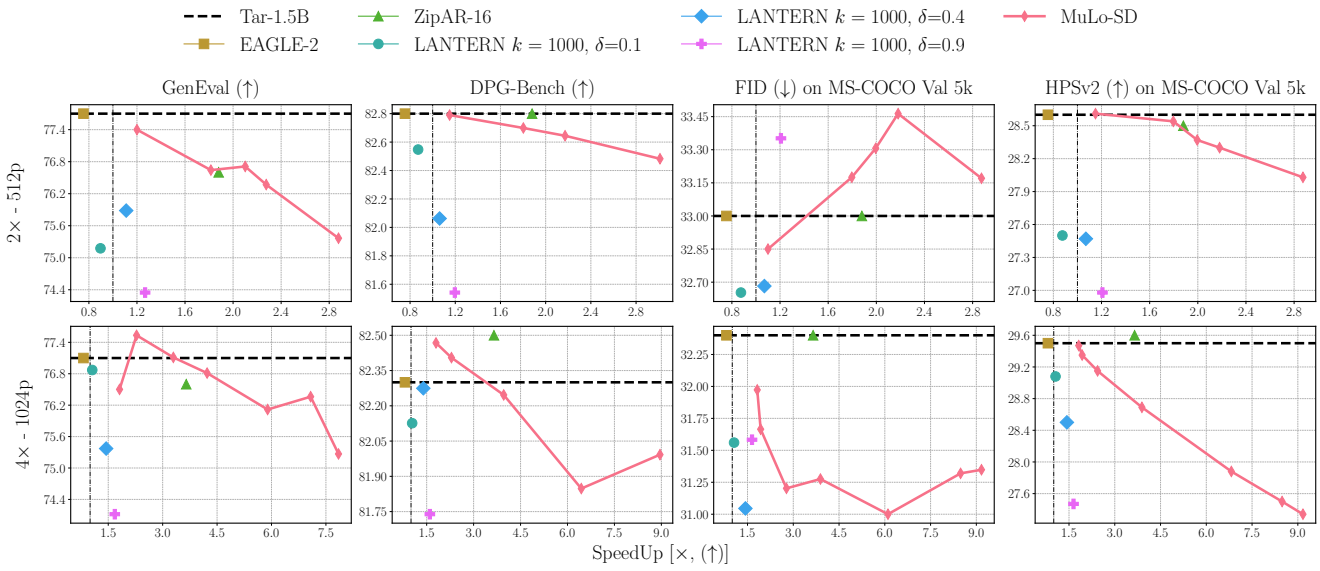


Figure 8. Quantitative evaluation of ZipAR [16], EAGLE-2 [26], LANTERN [20] and our method MuLo-SD. We report the GenEval [9] and DPG-Bench [18] semantic alignment metrics, along with the FID [17] and HPSv2 [50] perceptual quality metrics as computed on MS-COCO [29] 2017 Val 5k. To obtain a curve for MuLo-SD, we sweep the acceptance relaxation parameter τ, as described in Section 3.3 and Equation (5) in the main paper.

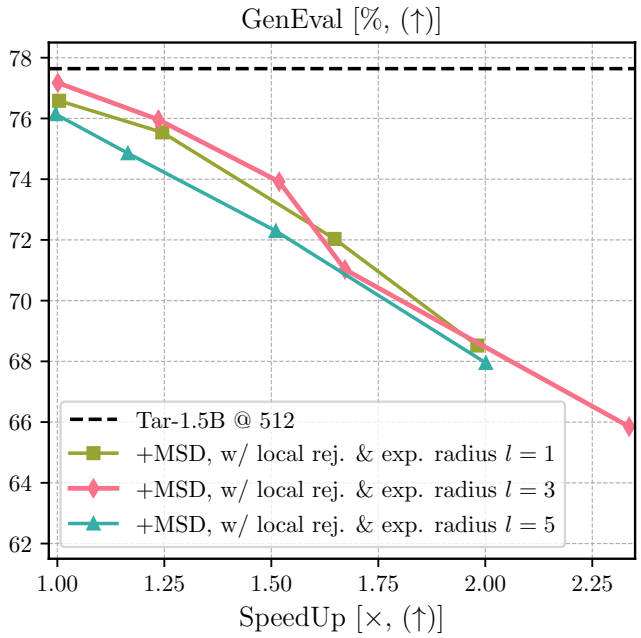


Figure 9. We study the effect of the local expansion radius l in MuLo-SD on GenEval [9] for the 2×case (256p → 512p). This expands the ablation in Figure 5 (c) in the main paper.

comparison is not possible. Nevertheless, in our setting, we measure Top-1 and Top-3 accuracy on the held-out test set as proxies for drafter quality. Higher accuracy correlates with greater inference-time speedups, as more tokens are accepted by the target model. We select the drafter achieving the highest test accuracy as our final model. Our results are as follows:

- **512p**: Top-1 = 0.12, Top-3 = 0.19
- **1024p**: Top-1 = 0.22, Top-3 = 0.33

When compared to LlamaGen results reported in the LANTERN paper (see Fig. 2(b) in [20]), our Top-1 accuracy is substantially lower (0.12 vs. 0.38). We attribute this discrepancy to Tar being a much stronger model than LLamaGen, making it harder to approximate due to its closer alignment with the true data distribution. For instance, Tar achieves significantly higher scores on benchmarks such as GenEval, where LlamaGen reportedly [47] scores 32% compared to 78% for Tar. Furthermore, the original paper notes that the drafter performs worse on slightly stronger models like Anole [5] compared to LlamaGen, reinforcing our hypothesis. Finally, we emphasize that the test sets differ, so direct comparisons are not strictly valid, although they provide context for interpreting our results.

9. Acknowledgments

We thank the authors of Tar [13], LANTERN [20] and ZipAR [16] for sharing their models and implementations.

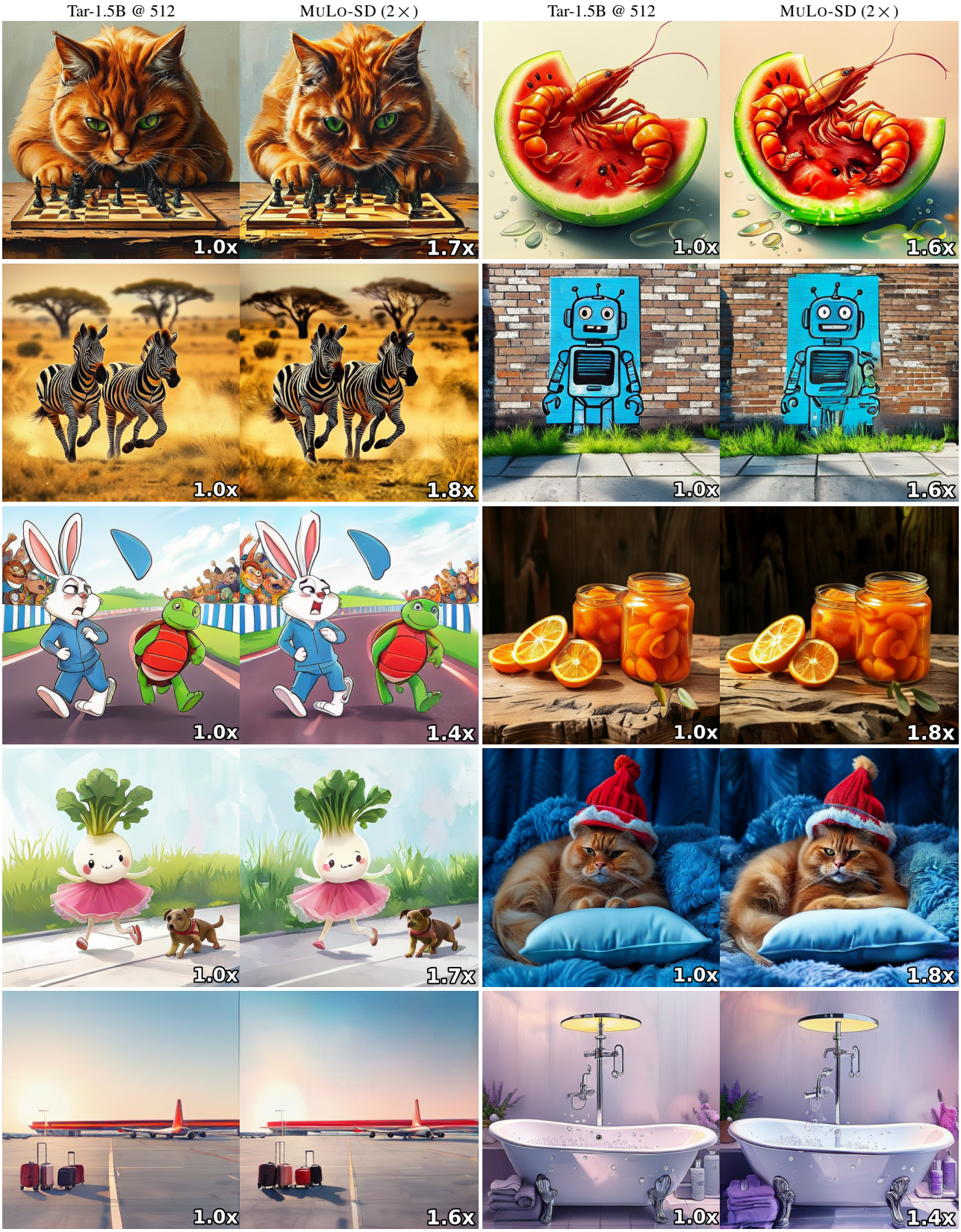


Figure 10. Visual comparison of 512p resolution, speedup displayed at the bottom-left corner. Prompts from DPG-Bench (top-bottom, left-right): `partiprompts175.txt`, `55.txt`, `partiprompts124.txt`, `partiprompts303.txt`, `stanford6.txt`, `180.txt`, `partiprompts177.txt`, `COCOval2014000000231527.txt`, `stanford36.txt`, `189.txt`

Figure 11. Visual comparison of 1024p image generations. Prompts from DPG-Bench: `partiprompts175.txt, 55.txt`.

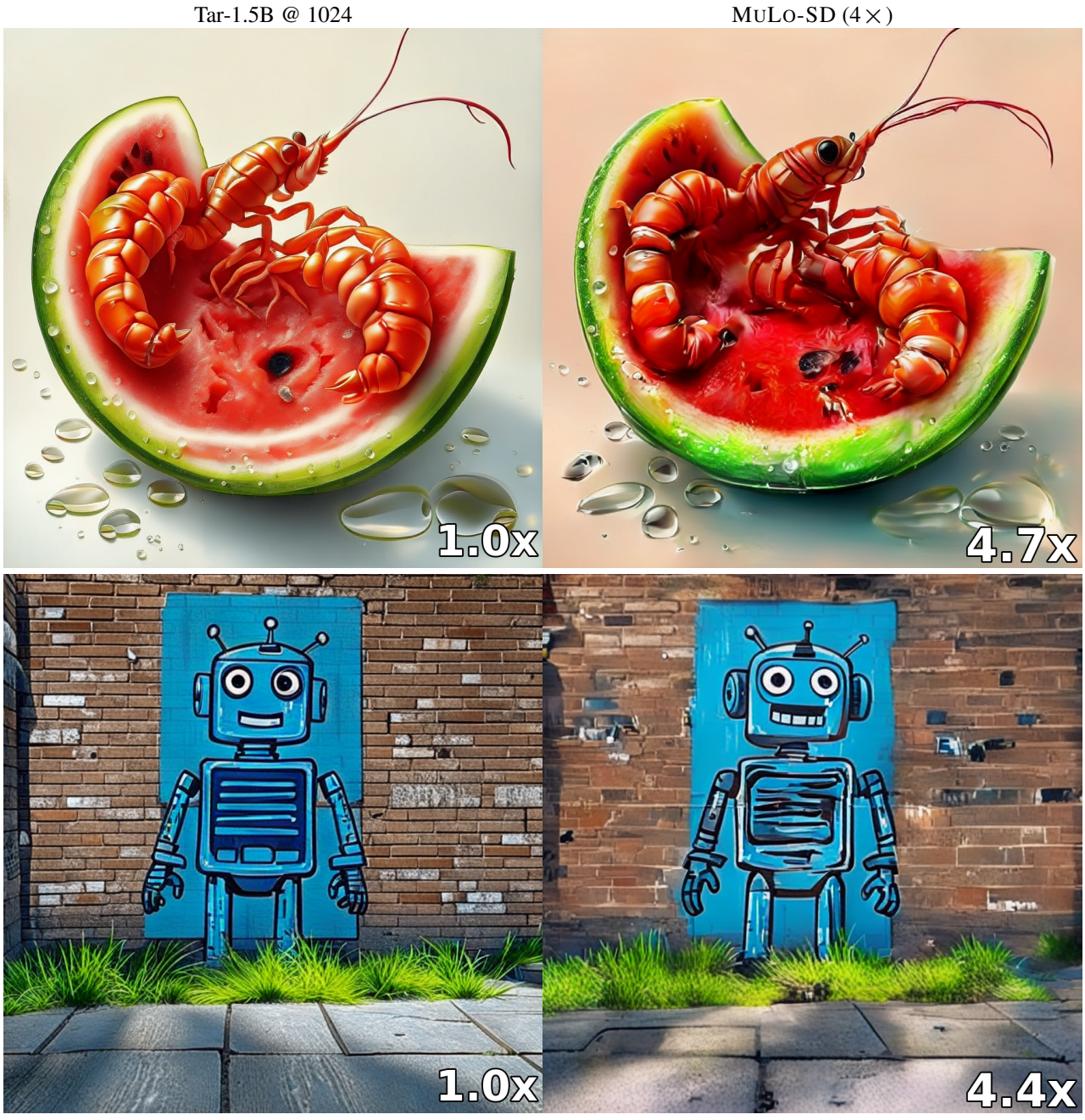


Figure 12. Visual comparison of 1024p image generations. Prompts from DPG-Bench: `180.txt, partiprompts177.txt`.

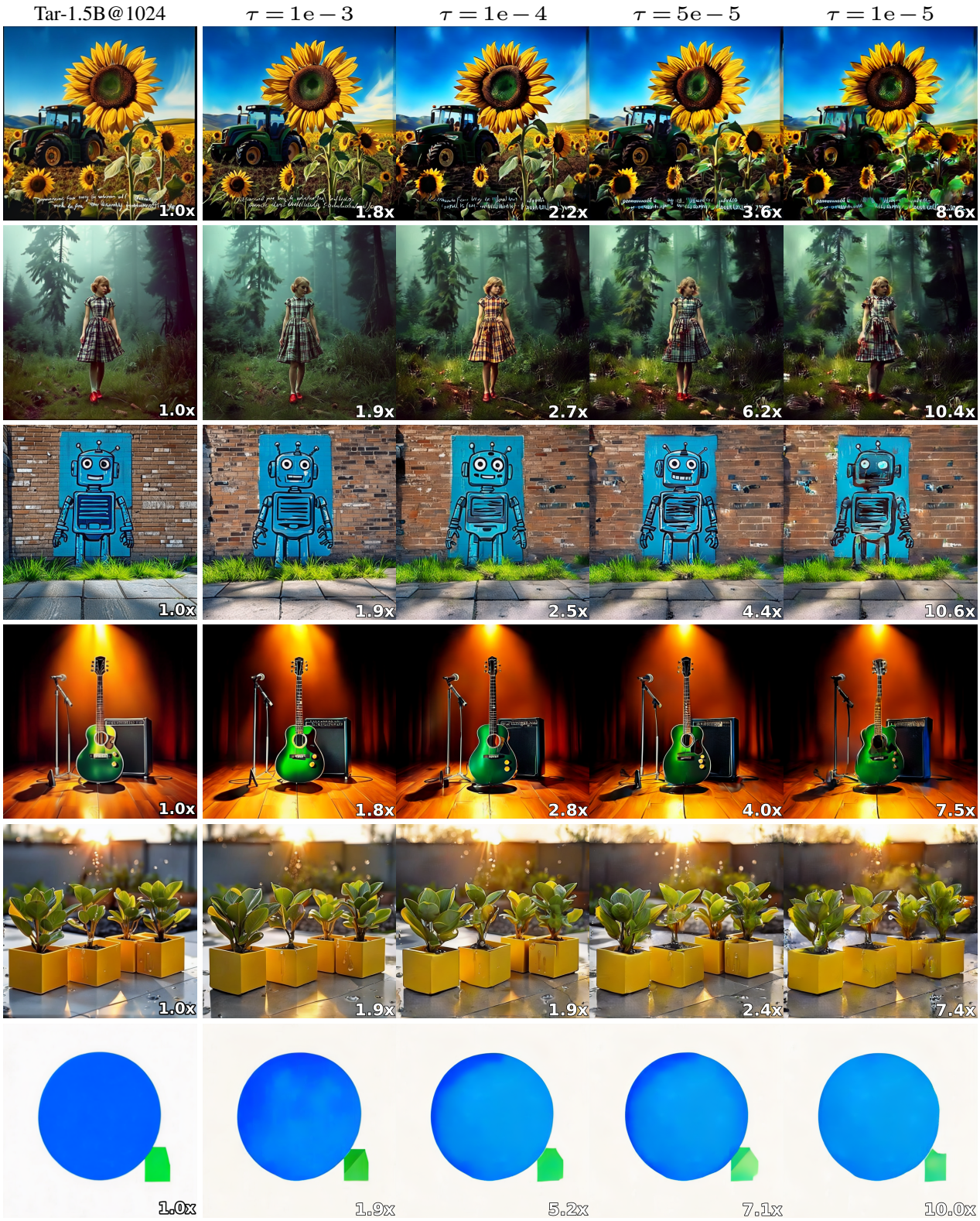


Figure 13. Visual comparison of 1024p image generations. We sweep the value of τ, our relaxed acceptance threshold as defined in Equation 5 in the main paper and show the related results. Prompts from DPG-Bench: `drawtext19.txt`, `midjourney33.txt`, `partiprompts177.txt`, `73.txt`, `74.txt`, `partiprompts77.txt`.