

## A APPENDIX

Table 4: Extension of main results for less aggressive, 1.4x speedups compared to 20 step DPM-Solver++ generations. We bold the best *overall* results, and underline the second best.

Settings			Latency		Prompt-aware Metrics		FID ↓	CLIP Image Quality Assessment			
Model	Res.	# Steps	# Blocks	s / img	Image Reward	CLIP	MJHQ	Good	Noisy ↓	Colorful	Natural
PixArt-alpha	1024	20	560	6.38	<u>94.43</u>	<u>28.96</u>	6.51	<b>92.71</b>	<u>23.92</u>	<u>57.79</u>	<u>66.26</u>
PixArt-alpha w/ cache	1024	20	380	4.25 (1.5x)	91.41	28.95	<u>6.09</u>	<u>92.61</u>	26.07	54.11	65.14
PixArt-alpha w/ ours	1024	12	388	4.25 (1.5x)	<b>96.68</b>	<b>29.03</b>	<b>5.92</b>	90.88	<b>23.84</b>	<b>61.39</b>	<b>67.46</b>
PixArt-sigma	1024	20	560	6.63	<u>83.87</u>	29.28	7.28	<b>90.32</b>	<u>27.98</u>	<u>59.60</u>	<u>69.12</u>
PixArt-sigma w/ cache	1024	20	380	4.38 (1.5x)	79.63	<u>29.35</u>	<b>6.56</b>	87.09	33.70	52.31	<b>72.48</b>
PixArt-sigma w/ ours	1024	12	388	4.50 (1.4x)	<b>84.06</b>	<b>29.41</b>	<u>6.95</u>	<u>89.06</u>	<b>26.42</b>	<b>67.94</b>	67.92
PixArt-alpha	512	20	560	1.06	92.03	29.06	7.13	<b>92.79</b>	<u>17.17</u>	<u>66.17</u>	<u>51.59</u>
PixArt-alpha w/ cache	512	20	380	0.72 (1.5x)	88.24	29.03	<b>6.52</b>	92.72	18.20	63.52	50.01
PixArt-alpha w/ ours	512	12	388	0.73 (1.5x)	<b>93.17</b>	<b>29.11</b>	<u>6.78</u>	<u>92.74</u>	<b>16.11</b>	<u>69.80</u>	<b>52.14</b>
PixArt-sigma	512	20	560	1.14	<u>94.17</u>	29.12	7.99	<b>89.57</b>	<u>20.04</u>	<u>65.67</u>	<u>52.69</u>
PixArt-sigma w/ cache	512	20	380	0.75 (1.5x)	92.38	<u>29.15</u>	<b>6.78</b>	88.68	20.82	62.19	<b>53.53</b>
PixArt-sigma w/ ours	512	12	388	0.77 (1.5x)	<b>96.31</b>	<b>29.19</b>	<u>7.31</u>	<u>89.10</u>	<b>19.13</b>	<b>70.96</b>	50.55

Table 5: Main results, high-quality text-to-image generation.

Loop Size		Latency		Prompt-aware Metrics		FID ↓	CLIP Image Quality Assessment			
Start	End	# Blocks	s / img	Image Reward	CLIP	MJHQ	Good	Noisy ↓	Colorful	Natural
0	5	420	0.78 (1.36x)	94.26	29.17	7.32	92.01	17.32	75.54	52.81
11	16	420	0.78 (1.36x)	93.80	29.15	7.07	92.86	15.88	73.63	53.50
22	27	420	0.78 (1.36x)	88.10	29.03	8.39	92.61	18.05	65.37	49.77
0	11	388	0.73 (1.44x)	93.10	29.15	6.47	92.99	15.72	70.56	53.16
8	19	388	0.73 (1.44x)	93.14	29.11	6.75	92.74	16.11	69.79	52.13
16	27	388	0.73 (1.44x)	90.20	29.05	7.56	92.52	17.73	65.68	50.08

We provide some more qualitative results, these from MJHQ prompts, in Figure 12. We also provide Figure 13 and Figure 14 as complements to Figure 7, where these have crops that we zoom in on to help the reader observe fine-grained differences. We provide a complement to Table 1, with results for 1.4x-1.5x speedups in Table 4. Whenever we cache with 1.4x-1.4x, we skip the inner 18 blocks every other step (rather than twice per 3 steps). For our feedback inference scheduling, we skip feedback on the inner 8 steps on a 12 step schedule for 1.4x-1.5x, and the inner 6 steps on a 10 step schedule for 1.7x-1.8x. We would like to emphasize ILF’s outstanding Image Reward and visual appearance, even compared to the non-accelerated baselines, in addition to the fact that for the 28 points of comparison in Table 1, it is superior for 19, and the second best for 6. In Table 5, we show all metrics for the experiments introduced in Table 3.

We show the effect of skipping the feedback for different amounts of steps in Figure 15. Without skipping enough steps, images can be distorted. We observe overall highest quality, without distortions, when skipping the middle 8 steps for a 12 step inference schedule.

In Figure 16 we perform an exploration where we compare our inference strategies on small and large loops. We find that for smaller loops, simply rescaling tends to be the best. However, for larger loops, we need to skip feedback on some steps to avoid distortions. While initially this would seem problematic, this is actually a blessing in disguise. We can save time by skipping feedback on some steps, and the feedback is powerful enough to give good quality by using it on the initial and final steps.

To determine our caching configuration, we run a brief search over some simple options. We use ‘inner’ for our main experiments since it gives superior results according to both quantitative and qualitative inspection. We show qualitative results for different caching schemes in Figure 17, and quantitative results in Table 6. We try caching, for every other step, the first  $c$  blocks, the last  $c$  blocks, the outer  $c$  blocks, the inner  $c$  blocks, and evenly-spaced  $c$  blocks (alternating). We show results for  $c = 14$ , PixArt-alpha 512x512. We find that caching first blocks results in blurriness. Caching last blocks results in detrimental distortions and artifacts. Caching outer blocks is better, but still somewhat blurry. While ‘last’ has similar Image Reward to ‘inner’ we prefer ‘inner’ since it doesn’t result in odd or unnatural generations, that may not be punished appropriately by the Image

Table 6: Caching exploration, different locations for PixArt-alpha, 512x512 images. Same settings as Figure 17.

Caching Location	# Steps	# Block Forward	Image Reward
First	20	440	83.35
Last	20	440	<b>89.54</b>
Outer	20	440	87.75
Inner	20	440	<u>89.37</u>
Alternating	20	440	83.98

Reward. Caching is merely a baseline, and not the focus of our paper. Nevertheless this limited investigation of caching for DiTs for text-to-image generation should hopefully help the community with training-free efficiency approaches.



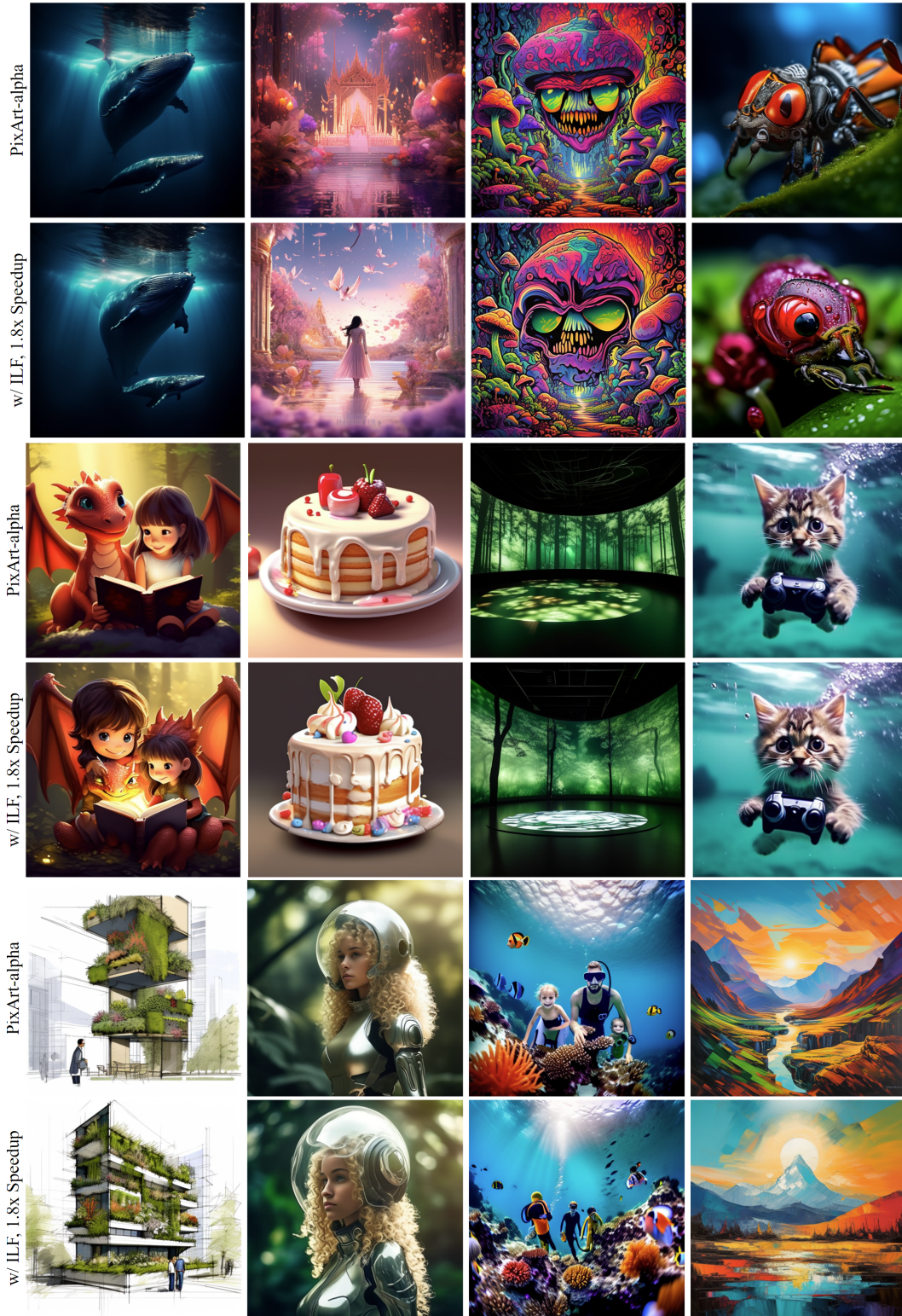


Figure 12: Bonus results page, large size for print readers. Alternating rows of baseline vs. ours, PixArt-alpha 512x512, with **1.8x speedup**. See supplementary for further examples. The top two rows should where our results are roughly equal, second row shows some failure cases, and third row shows some instances where ours are superior.



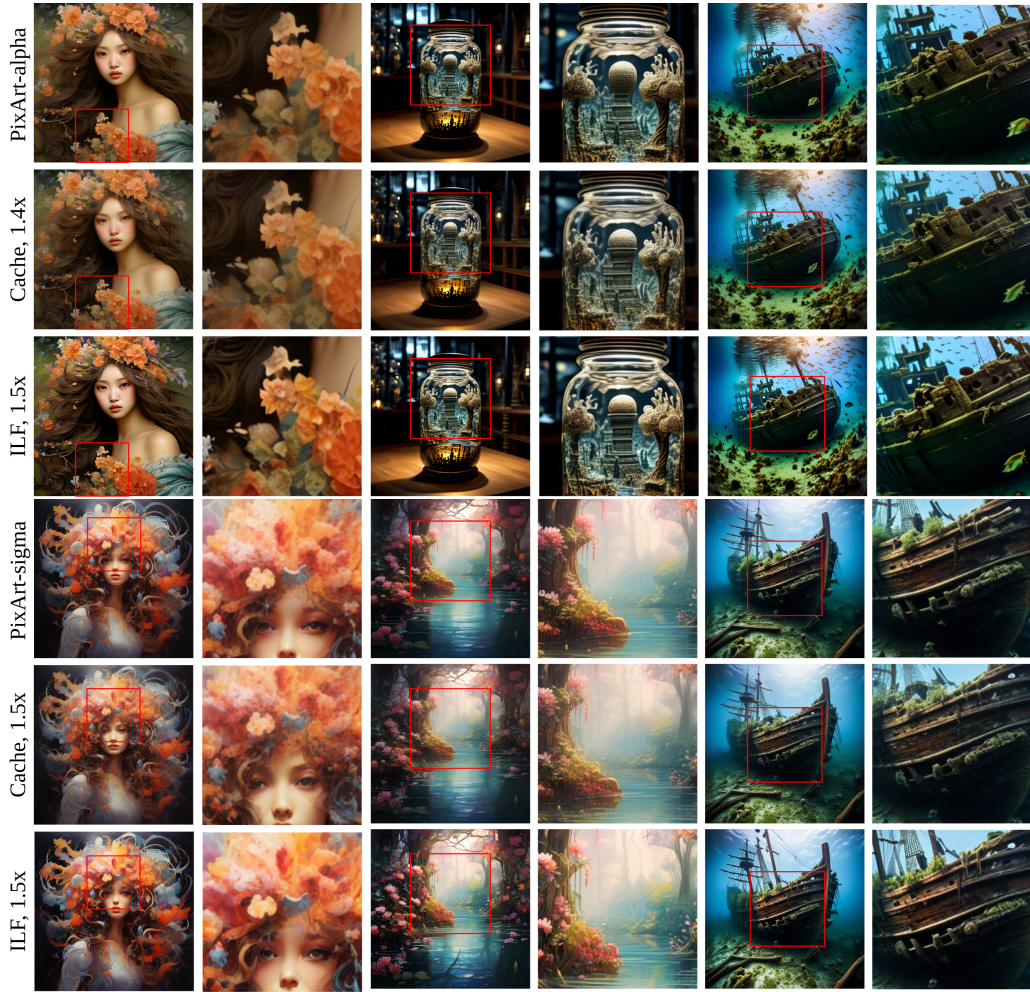


Figure 13: 512x512 results, alpha (top 3 rows) and sigma (bottom 3 rows), with baseline, caching, and our results, respectively, for 1.4x-1.5x acceleration. ILF yields images of similar content and quality to the un-accelerated baseline, and clearly superior to the caching, for both models. Zoomed in and cropped for convenience.

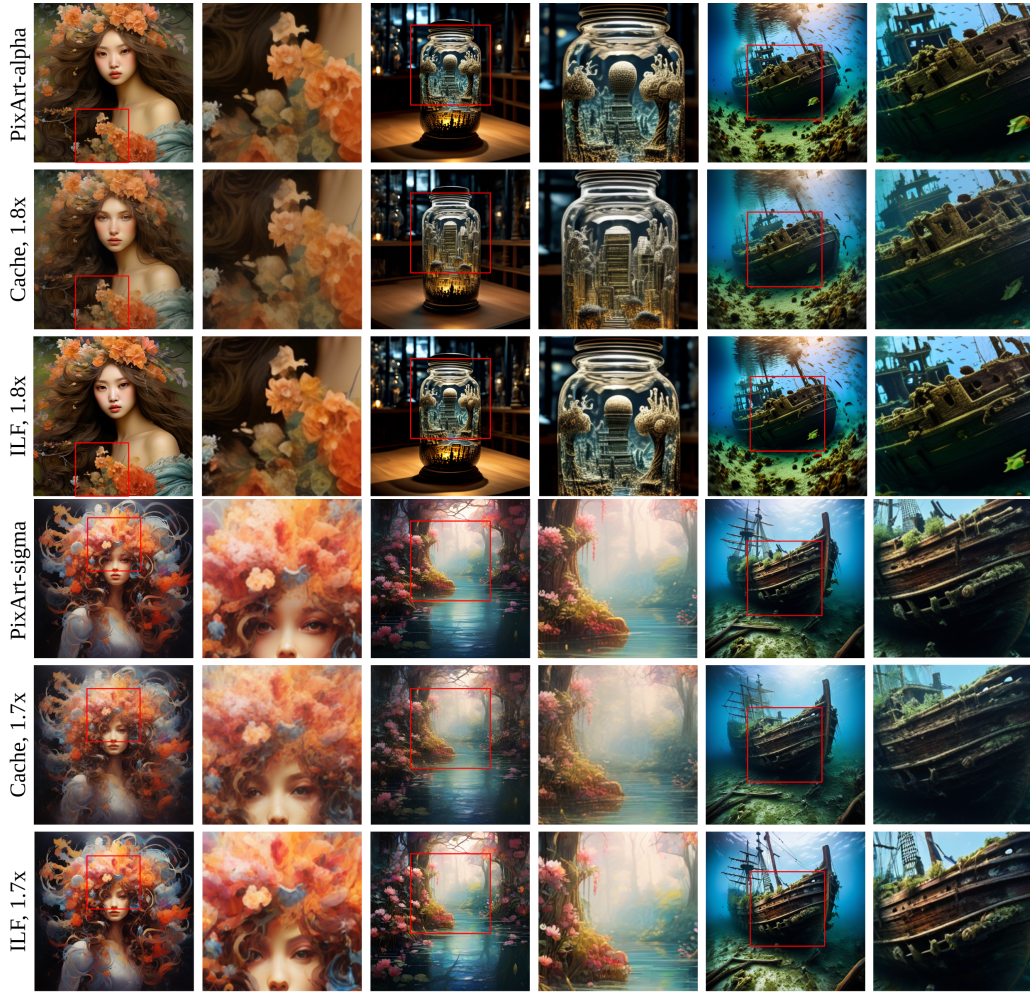


Figure 14: 512x512 results, alpha (top 3 rows) and sigma (bottom 3 rows), with baseline, caching, and our results, respectively, for 1.7x-1.8x acceleration. ILF yields images of similar content and quality to the un-accelerated baseline, and clearly superior to the caching, for both models. Zoomed in and cropped for convenience.





Figure 15: We try skipping feedback more and less often for 12 step inference. On the top, we do not skip any feedback (equivalent to our feedback time step rescaling strategy). On the next row, we skip the middle 4 steps, then on the next row the middle 8 steps, and finally, for the bottom row, we skip the inner 10 steps. All results use PixArt-alpha, 512x512 images, inner loop from block  $b = 8$  to  $b = 19$ . We trade off quantity of details (such as number of stars) and sharpness in exchange for smoother, better lit, more natural images. Notably this is controllable, and can be adjusted according to the practitioner’s preference.





Figure 16: We compare different inference strategies (default, rescaled, and skipping, in order) for two different loop sizes; on the top, a larger loop from block  $b = 8$  to  $b = 19$ , and on the bottom, a smaller loop connecting  $b = 0$  to  $b = 5$ . Note that when using default inference, without accounting for the feedback, the image content and quality degrades (zoom-in required). For both models this default inference results in images with excessive details and overly bright, over-saturation. Using our feedback time step rescaling (2nd and 5th rows) helps mitigate this to an extent, and for the smaller loop, this produces the best images. However, for the larger loop, we get the most natural, but still detailed images, when both rescaling and skipping feedback on some time steps (bottom row). One can control the level of detail by changing the frequency of the skipping.



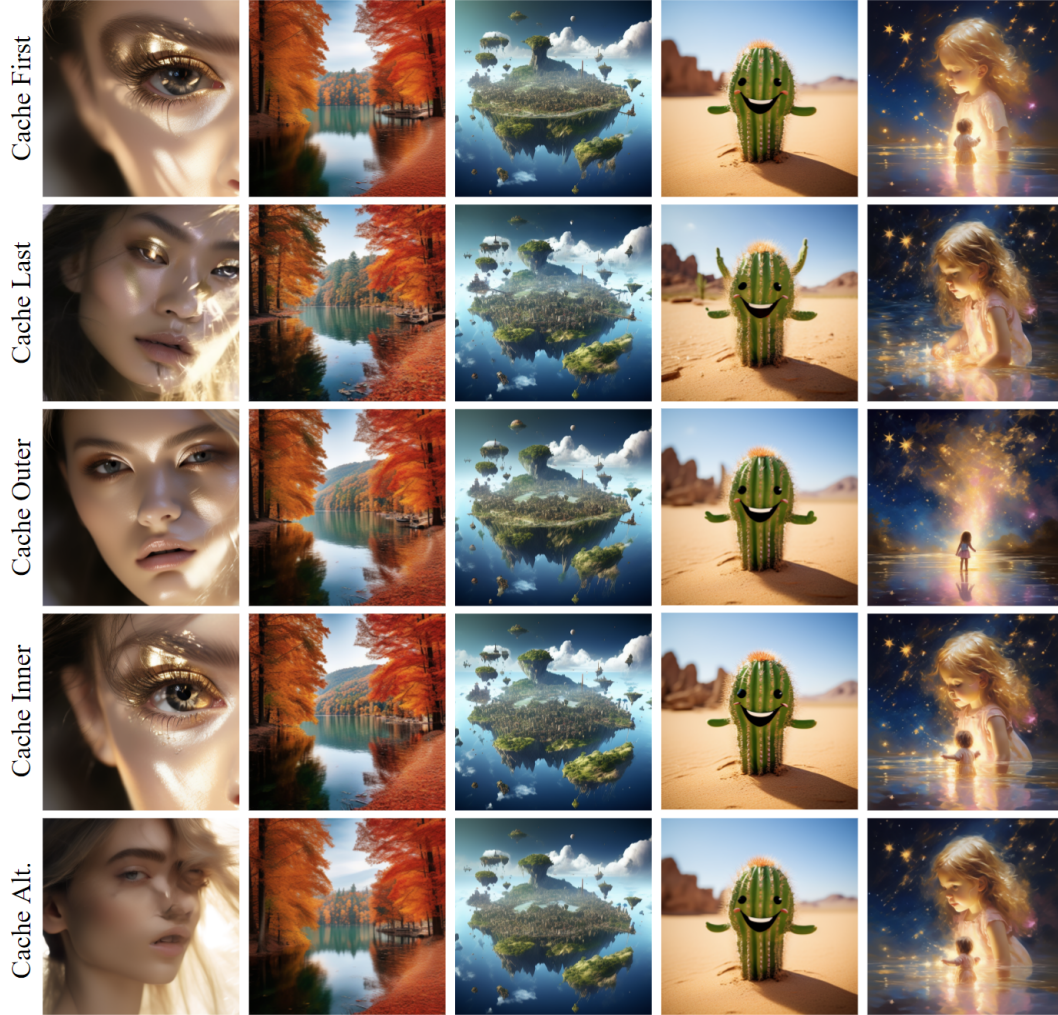


Figure 17: We compare different caching locations for PixArt-alpha, 512x512 images, caching 14 blocks every other step. We cache the first blocks (top), last blocks, outer blocks, inner blocks, and alternating blocks (bottom). Notice the blurriness when caching first and alternating (Alt.) blocks, the distortions when caching last blocks, and overall poor quality when caching outer blocks. None of these are ideal, but we find the best performance for inner; thus, in our main results we compare to caching inner blocks.