AOSR: ASYNCHRONOUS ON-POLICY SPECULATIVE ROLLOUT FOR EFFICIENT DISTILLATION OF AUTOREGRESSIVE IMAGE GENERATORS

Anonymous authors
Paper under double-blind review

ABSTRACT

On-policy distillation (OPD) improves compact autoregressive image generators by training them on image-token sequences generated by the current student. However, producing these sequences autoregressively requires repeated sequential forwards, making student rollout a major training bottleneck. We explore speculative multi-token rollout to reduce this cost while maintaining distillation performance, but find that a batch-synchronous speculative implementation can even reduce efficiency: different sequences obtain different amounts of verified progress, while synchronous execution prevents faster sequences from advancing independently. Based on this observation, we propose **AOSR**, an **A**synchronous **O**n-policy **S**peculative **R**ollout framework that preserves sequence-specific verified progress and adaptively removes completed sequences. We further introduce **Cross-Step Speculative Refill (CSR)**, which reuses completed batch positions to pre-generate sequences for the next rollout and treats the prefetched tokens as speculative candidates. After the student update, these tokens are re-verified with the updated student, resampling mismatched tokens when necessary, thereby reusing computation across OPD steps while maintaining comparable generation quality. Experiments on LlamaGen and Janus-Pro with GKD and VarKD show that AOSR substantially reduces rollout and end-to-end training time, while adding CSR provides further acceleration with comparable generation quality.

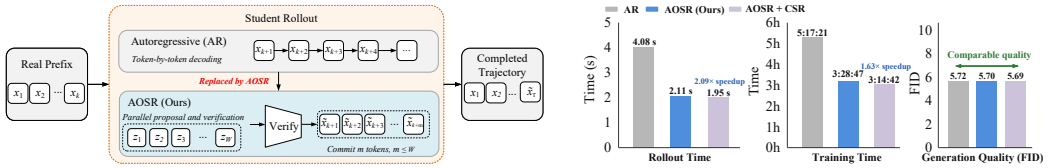
1 INTRODUCTION

Autoregressive (AR) image generators have achieved strong generation quality and scalability by modeling images as sequences of visual tokens (Sun et al., 2024; Tian et al., 2024). As these models grow, knowledge distillation provides a way to transfer their capabilities to compact students (Hinton et al., 2015; Kim & Rush, 2016). Conventional distillation mainly trains the student on real data or teacher-generated sequences, while inference requires the student to condition on its own previous predictions. On-policy distillation (OPD) reduces this training–inference mismatch by training the student on image-token sequences generated by the current student and using the teacher to supervise the resulting student-visited states (Agarwal et al., 2024; Peruzzo et al., 2026). Recent work has shown the effectiveness of this paradigm for distilling autoregressive image generators, making OPD a promising approach for training image-generation models (Peruzzo et al., 2026).

However, OPD introduces substantial training cost for autoregressive image generators. Before teacher–student supervision can be computed, the current student must first generate an image-token sequence online. As illustrated in Fig. 1(a), this student rollout is produced autoregressively, with each image token conditioned on the previously generated tokens. The current student therefore has to perform repeated sequential forwards before the full image-token sequence is available for distillation, making student rollout a major source of training cost. In other words, faster student rollout directly leads to faster OPD training for autoregressive image generators. This naturally raises a key question: *can we substantially accelerate student rollout for autoregressive image generators without degrading distillation performance?*

To answer this question, we first explore speculative multi-token rollout as an alternative to standard token-by-token rollout. Speculative rollout proposes multiple future image tokens and uses

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Student rollout replacement in visual OPD.

(b) Cost and performance comparison.

Figure 1: Overview and main benefit of AOSR. **(a)** AOSR replaces the token-by-token autoregressive student rollout with parallel proposal, verification, and multi-token commitment. **(b)** AOSR reduces rollout and end-to-end training time while maintaining comparable generation quality.

the current student to verify them before commitment. With standard speculative verification and resampling, multiple accepted tokens can be committed at once, reducing sequential decoding while preserving the student's autoregressive distribution. However, our experiments in Sec. 3.1 show that directly replacing standard rollout with a batch-synchronous speculative implementation does not necessarily improve efficiency and can even be slower than standard autoregressive rollout. We analyze this behavior in detail and find that different sequences often verify and commit different numbers of tokens in the same rollout window. Under synchronous execution with shared rollout boundaries, sequences that make more progress cannot advance to the next rollout window until the slowest sequences finish the current one, limiting the benefit of speculative multi-token generation. Importantly, this synchronization is imposed by the execution schedule rather than by autoregressive generation itself, since each sequence is conditioned only on its own generated history. The above analysis provides a key insight: *speculative multi-token rollout naturally produces heterogeneous verified progress across sequences, and allowing this progress to evolve independently can unlock its acceleration potential while maintaining distillation performance.*

Based on this insight, we propose **AOSR**, an **A**synchronous **O**n-policy **S**peculative **R**ollout framework for autoregressive image generators. AOSR consists of three key designs. First, **Sequence-wise Asynchronous Advancement** allows each sequence to preserve its own verified progress and immediately continue from its current decoding position, removing shared rollout boundaries while retaining batched student forwards. Second, **Completion-aware Batch Adaptation** dynamically removes completed sequences when the saved computation outweighs the overhead of batch reorganization and KV-cache access, reducing later-stage rollout cost. Third, **Verifier-agnostic Draft Refinement** allows AOSR to incorporate verifier-side improvements, such as confidence-based draft reuse, to further reduce redundant verification. However, these designs only accelerate the current rollout, while batch positions released by early-finished sequences remain unused. We therefore introduce **Cross-Step Speculative Refill (CSR)**, which uses these positions to pre-generate a short prefix of the next rollout. Since the student is updated between consecutive OPD steps, the prefetched tokens are treated as speculative candidates and re-verified with the updated student before being used. In this way, AOSR accelerates the current rollout through asynchronous advancement, adaptive batch execution, and draft refinement, while CSR further reuses the resulting rollout tail across OPD steps. AOSR and CSR only modify student rollout, leaving the teacher, distillation objective, and optimization procedure unchanged. As summarized in Fig. 1(b), AOSR accelerates rollout and training, while AOSR + CSR provides further gains with comparable generation quality.

Our contributions are summarized as follows:

- We reveal that batch-synchronous speculative multi-token rollout in OPD can be inefficient because different image-token sequences obtain different amounts of verified progress, while synchronized execution forces them to advance together. Our analysis shows that preserving such sequence-specific progress independently can further accelerate rollout without changing the underlying distillation objective.

- We propose **AOSR**, an Asynchronous On-policy Speculative Rollout framework that preserves independent verified progress and adaptively removes completed sequences. We introduce **Cross-Step Speculative Refill (CSR)**, which reuses completed batch positions to pre-generate the next rollout and re-verifies prefetched tokens after the student update.

- Extensive experiments across different model architectures and distillation methods demonstrate that AOSR consistently reduces student rollout and end-to-end training time. Adding CSR provides further acceleration while maintaining generation quality.

2 Related Work

2.1 On-Policy Distillation

Knowledge distillation transfers knowledge from a teacher to a smaller student through distribution- or sequence-level supervision (Hinton et al., 2015; Kim & Rush, 2016). For autoregressive generative models, MiniLLM studies sequence-level distillation with student-generated sequences, while GKD establishes a general on-policy paradigm in which the teacher supervises states visited by the current student (Gu et al., 2024; Agarwal et al., 2024). This paradigm has also been extended to visual tasks. Vision-OPD studies multimodal understanding through regional-to-global self-distillation, using a crop-conditioned teacher with privileged local evidence to supervise a full-image student along student-generated rollouts (Yuan et al., 2026). For autoregressive image generation, VarKD studies supervised, sequence-level, and on-policy distillation for models such as VAR and LlamaGen (Tian et al., 2024; Sun et al., 2024; Peruzzo et al., 2026). These works mainly focus on the supervision or distillation formulation, while our work keeps the learning formulation unchanged and accelerates the student rollout required by OPD.

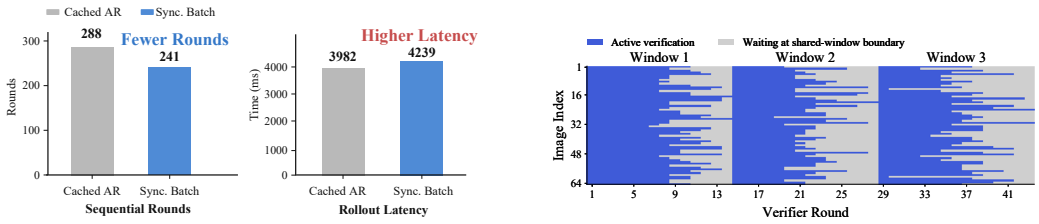
2.2 Autoregressive Generation Acceleration

Speculative decoding reduces the sequential cost of autoregressive generation by proposing multiple future tokens and verifying them in parallel (Leviathan et al., 2023; Chen et al., 2023). Related methods have been developed for autoregressive image generation, including ZipAR, Continuous Speculative Decoding, LANTERN, LANTERN++, and GSD (He et al., 2024; Wang et al., 2026; Jang et al., 2025; Park et al., 2025; So et al., 2025b). Speculative Jacobi Decoding (SJD) removes the need for an auxiliary draft model by reusing predictions from previous Jacobi iterations, with subsequent variants further improving proposal and verification efficiency (Teng et al., 2024; 2025; So et al., 2025a; Teng et al., 2026; Shan et al., 2026; Zhang et al., 2026; Yu et al., 2026; Kang et al., 2026; Liao et al., 2026; Lei et al., 2026). Serving systems such as Orca and vLLM improve autoregressive execution through iteration-level scheduling and efficient KV-cache management (Yu et al., 2022; Kwon et al., 2023). BASS supports heterogeneous accepted lengths through ragged attention, while EXSPEC uses same-length grouping to reduce realignment overhead (Qian et al., 2024; Zhang et al., 2025). AOSR focuses on completing each OPD rollout through sequence-wise speculative advancement and completion-aware batch adaptation, without requiring same-length regrouping. For training rollouts, SPEC-RL reuses cached trajectories for recurring prompts (Liu et al., 2025), TLT uses idle GPUs during the rollout tail to train a drafter (Hu et al., 2026), and Iso et al. (2026) integrate speculative decoding into synchronous and asynchronous RL pipelines. CSR instead uses released batch positions to pre-generate short drafts for the next rollout and re-verifies them after the student update, without requiring historical trajectories for those inputs.

3 Method

3.1 Motivation Analysis

We examine whether speculative multi-token rollout can directly improve student-rollout efficiency in OPD. We compare **Cached AR**, which generates the student suffix token by token with KV caching, against **Batch-Synchronous Speculative Rollout**, where all sequences perform multi-token proposal and verification under shared rollout-window boundaries. We use the same student checkpoint and input batch for both methods, with 64 ImageNet samples, a 288-token prefix, and a 288-token student-generated suffix. As shown in Fig. 2a, Batch-Synchronous Speculative Rollout reduces the number of sequential rounds from 288 to 241, yet increases rollout latency from 3982 ms to 4239 ms. To understand this counterintuitive result, we further inspect the execution trace in Fig. 2b. Different image-token sequences make different amounts of verified progress within each rollout window. Some sequences advance quickly, while others require more verifier rounds to reach the same shared boundary. However, under batch-synchronous execution, the next rollout window



(a) Direct batch-synchronous speculative rollout.

(b) Shared-window waiting.

Figure 2: Empirical analysis of direct speculative rollout in OPD. **(a)** Batch-synchronous speculative rollout reduces the number of sequential rounds but increases rollout latency compared with Cached AR. **(b)** Different image-token sequences make different amounts of verified progress within each rollout window, forcing faster sequences to wait at shared rollout boundaries.

cannot start until all sequences complete the current one, forcing faster sequences to remain idle while waiting for slower ones. Across all sequences and verifier rounds, 46.15% of the observed sequence states are spent waiting at shared rollout boundaries (see Appendix A.2 for measurement details). This waiting overhead is particularly costly because each speculative verifier round processes multiple tokens and is more expensive than a single Cached AR decoding step. As a result, the reduction in sequential rounds does not translate into lower wall-clock latency.

The above analysis shows that the main limitation of direct speculative rollout in OPD lies in the mismatch between heterogeneous sequence-wise progress and batch-synchronous execution. Although different sequences can benefit differently from speculative verification, shared rollout boundaries repeatedly force their progress to be synchronized, substantially offsetting the acceleration benefit brought by reducing sequential decoding rounds.

3.2 AOSR: Asynchronous On-policy Speculative Rollout

Based on the above analysis, we propose **AOSR**, an Asynchronous On-policy Speculative Rollout framework that preserves the verified progress of each sequence during batched student rollout. AOSR is compatible with different speculative proposal mechanisms; in this work, we instantiate it with Speculative Jacobi Decoding (SJD) (Teng et al., 2024). SJD is particularly suitable for OPD because it requires neither an auxiliary draft model nor additional training, avoiding the potential mismatch between a separate draft model and the continuously updated student during OPD training. Instead, it reuses predictions from previous Jacobi iterations as future-token proposals, allowing speculative rollout to be performed entirely with the current student.

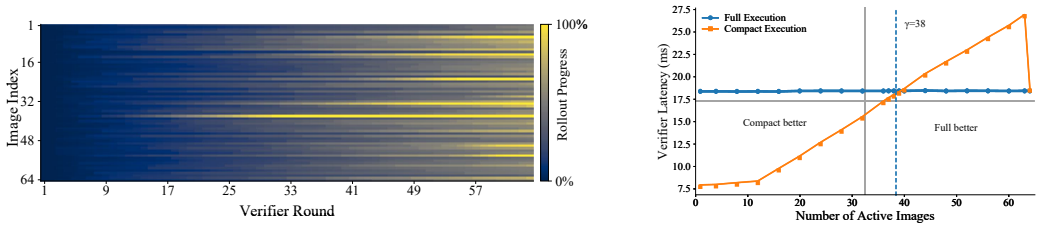
SJD-based speculative rollout. AOSR uses SJD (Teng et al., 2024) to generate multiple image tokens in each student forward. Given the currently generated sequence, SJD maintains a Jacobi window containing W draft tokens for the next W positions. The current student processes the entire window in parallel and obtains a conditional distribution p at each position, while q denotes the proposal distribution associated with the corresponding draft token from the previous Jacobi iteration. Under probabilistic verification, SJD verifies the draft tokens from left to right. A draft token z is accepted if

$$r < \min\left(1, \frac{p(z)}{q(z)}\right), \qquad r \sim \mathcal{U}[0,1]. \tag{1}$$

Verification continues until the first rejection. If a draft token z is rejected, it is replaced by sampling from the normalized residual distribution:

$$z \leftarrow \text{Sample}\left(\text{Normalize}\left([p-q]_+\right)\right). \tag{2}$$

The verified prefix together with the resampled token determines the committed progress of the current round. Under probabilistic verification, repeating this process allows multiple tokens to be committed in each iteration while preserving the original autoregressive distribution.



(a) Asynchronous verified progress.

(b) Adaptive batch execution.

Figure 3: Execution behavior of AOSR. **(a)** Different image-token sequences preserve their own verified progress and advance independently while remaining in batched student forwards. **(b)** Full and Compact execution have different advantages as the number of active sequences decreases, motivating adaptive removal of completed sequences.

Sequence-wise asynchronous advancement. The key difference from batch-synchronous speculative rollout lies in how verified progress is maintained. AOSR assigns an independent decoding cursor c_b to each image-token sequence. Let $m_b^{(r)}$ denote the number of tokens committed by sequence b at verification round r. Its cursor is then updated by

$$c_b^{(r+1)} = c_b^{(r)} + m_b^{(r)}. \tag{3}$$

Since $m_b^{(r)}$ can differ across sequences, sequences processed in the same student forward may advance by different amounts. Instead of returning to a shared rollout boundary, each sequence immediately starts the next proposal window from its own verified position.

Although different sequences may be at different decoding positions, AOSR does not synchronize them to a common decoding cursor. Instead, each sequence keeps its own generated prefix, decoding cursor, position indices, and KV-cache state. During each student forward, the proposal window of each active sequence is verified against its own generated prefix, while proposal windows from different sequences are processed together in the same batch. With KV caching, the preceding prefix is already stored in the corresponding cache, so only the current verification tokens need to be processed again. In this way, AOSR allows each sequence to advance according to its own verified progress while retaining batched student execution.

Completion-aware batch adaptation. Independent progress naturally causes sequences to finish at different times, as shown in Fig. 3(a). Keeping completed sequences in later student forwards introduces unnecessary computation, but removing them immediately is not always beneficial because compacting the batch and accessing selected KV-cache rows introduce additional overhead. As shown in Fig. 3(b), Full execution is more efficient when most sequences remain active, whereas Compact execution becomes preferable after the active batch becomes sufficiently small. AOSR therefore supports two execution modes. Full execution keeps the original physical batch and marks completed sequences as inactive, while Compact execution removes completed sequences and processes only the remaining active sequences and their KV-cache states. Before training, we run a short, data-independent latency calibration to determine a fixed mode-switching threshold γ. Compact execution is used when the number of active sequences falls below γ.

Verifier-agnostic draft refinement. AOSR operates on the verified progress returned by each speculative round and is agnostic to how the underlying SJD verifier updates its draft window. Therefore, verification-side improvements developed for SJD can be directly incorporated into AOSR as long as they preserve this interface. In this work, we instantiate this flexibility with the confidence-based token reuse strategy of SJD++ (Teng et al., 2025). After the first rejection, SJD++ measures the stability of each remaining draft token by

$$C_i^{(j)} = \frac{p_\theta(x_i^{(j)} \mid \mathbf{x}_{<i}^{(j)})}{p_\theta(x_i^{(j)} \mid \mathbf{x}_{<i}^{(j-1)})}, \qquad x_i^{(j+1)} = \begin{cases} x_i^{(j)}, & C_i^{(j)} > \tau, \\ \tilde{x}_i^{(j+1)} \sim p_\theta(\cdot \mid \mathbf{x}_{<i}^{(j)}), & \text{otherwise.} \end{cases} \tag{4}$$

This allows high-confidence draft tokens to be reused rather than resampling the entire unverified suffix, reducing redundant verification overhead and further improving the efficiency of AOSR. We adopt this verifier refinement strategy throughout the following experiments.

3.3 Cross-Step Speculative Refill

AOSR reduces rollout latency by allowing each image-token sequence to advance according to its own verified progress. However, the current rollout still finishes only when the last sequence completes. Although adaptive removal reduces the computation spent on sequences that finish earlier, their released batch positions are not used to advance future generation, limiting the further acceleration potential of AOSR. We therefore introduce **Cross-Step Speculative Refill (CSR)**, which refills these positions with sequences from the next rollout and starts generating their tokens before the current rollout finishes. A key challenge is that the student is updated between consecutive OPD steps. Let θ_t denote the student before the update and θ_{t+1} the updated student. Tokens generated in advance by θ_t cannot be directly used by the next rollout, so CSR treats them as speculative drafts and re-verifies them with θ_{t+1} after the update. Moreover, pre-generating a long draft provides limited benefit: once an early draft token is rejected, later tokens are conditioned on an outdated prefix and become increasingly unlikely to be reused. CSR therefore pre-generates only a short prefix of the next rollout. When CSR is enabled, we disable the Full/Compact mode switching in AOSR and keep the physical batch size fixed during cross-step refill. After the update, the prefetched prefix is verified by θ_{t+1}, and generation continues from the verified result. Detailed execution procedures are provided in the Supplementary Material (Appendix A.3). In this way, CSR uses the rollout tail created by AOSR to advance the next OPD step while avoiding excessive speculative computation.

4 Experiments

4.1 Experimental Setup

Models and distillation. We conduct our main experiments on ImageNet using LlamaGen (Sun et al., 2024). We use LlamaGen-L and LlamaGen-XL as teachers and LlamaGen-B as the student, covering teacher–student pairs with different capacity gaps. We evaluate two on-policy distillation objectives, GKD (Agarwal et al., 2024) and VarKD (Peruzzo et al., 2026), and include conventional KD as a reference. For each teacher–student pair and distillation objective, all rollout methods use the same training data, optimization configuration, and number of training steps. AOSR and CSR only modify how student rollout is executed, while the teacher, distillation objective, and optimization procedure remain unchanged.

Evaluation and efficiency. Generation quality is evaluated on 50K generated samples using FID and Inception Score (IS), with additional precision and recall results provided in the Supplementary Material. For rollout efficiency, we report the latency of generating the student image-token sequence and its speedup over the corresponding Cached AR baseline. We further report end-to-end training-step speedup, where each step includes student rollout, teacher supervision, loss computation, backward propagation, and optimization. All compared methods are measured under the same NVIDIA H200 hardware and training configuration, and the main latency results are averaged over three independent runs.

Rollout configuration. We use standard KV-cached token-by-token generation as the autoregressive rollout baseline, denoted as *Cached AR*. Following VarKD (Peruzzo et al., 2026), we use conditional student sampling with $\lambda_{\mathrm{GKD}} = 1$, where the ground-truth prefix length k is uniformly sampled from $[0.25T, 0.75T]$ and the remaining suffix is generated by the student. Unless otherwise specified, AOSR is instantiated with SJD (Teng et al., 2024) using a Jacobi window size of $W = 16$, classifier-free guidance with scale 2.0, and probabilistic sampling. CSR is applied on top of AOSR with a prefetch length of $K = 16$, using batch positions released by completed sequences to pre-generate a short prefix of the next rollout, which is re-verified after the student update. All reported efficiency measurements include proposal generation, verification, batch-management overhead, and CSR re-verification. Probabilistic verification, proposal initialization, and additional implementation details are provided in the Supplementary Material.

Table 1: Main results on ImageNet.

Method	Generation Quality		Rollout Efficiency		E2E Step Speedup ↑
	FID ↓	IS ↑	Time (ms) ↓	Speedup ↑	
LlamaGen-L → LlamaGen-B					
LlamaGen-B	6.51	156.30	–	–	–
+ KD	5.67	182.53	–	–	–
+ GKD (AR)	5.72	159.15	4080	1.00×	1.00×
+ GKD + AOSR	5.70	162.35	2114	1.93×	1.52×
+ GKD + AOSR + CSR	5.69	161.65	1946	2.09×	1.63×
+ VarKD (AR)	5.49	167.43	4073	1.00×	1.00×
+ VarKD + AOSR	5.49	168.35	2161	1.88×	1.50×
+ VarKD + AOSR + CSR	5.51	168.65	1873	2.17×	1.65×
LlamaGen-XL → LlamaGen-B					
LlamaGen-B	6.51	156.30	–	–	–
+ KD	5.57	186.68	–	–	–
+ GKD (AR)	5.77	157.68	4122	1.00×	1.00×
+ GKD + AOSR	5.78	160.42	2179	1.89×	1.35×
+ GKD + AOSR + CSR	5.80	157.65	1808	2.27×	1.49×
+ VarKD (AR)	5.51	172.82	4080	1.00×	1.00×
+ VarKD + AOSR	5.50	171.52	2305	1.77×	1.31×
+ VarKD + AOSR + CSR	5.53	172.79	2030	2.01×	1.42×

4.2 Main Results

Quantitative results. Table 1 reports the main results across two teacher–student pairs and two on-policy distillation objectives. AOSR consistently accelerates student rollout across all evaluated configurations, while adding CSR provides further improvement by reusing batch positions released by asynchronously completed sequences. These rollout gains also translate into faster end-to-end training. Importantly, the acceleration does not come at the cost of generation quality, with FID and IS remaining comparable to the corresponding AR baselines. The consistent gains across different teacher scales and distillation objectives demonstrate that AOSR and CSR effectively reduce OPD training cost without changing the underlying distillation procedure.

Qualitative results. Figure 4 presents generated samples using LlamaGen-XL as the teacher and LlamaGen-B as the student. Under GKD and VarKD, models trained with AOSR produce samples visually comparable to Cached AR in object semantics, global structure, and local appearance. Together with Table 1, these examples show that AOSR accelerates student rollout without noticeable degradation in generation quality.

4.3 Ablation Study

Does our method generalize to different autoregressive image generators? To investigate whether the acceleration of our method depends on a specific autoregressive image-generation architecture, we further evaluate AOSR and CSR on Janus-Pro (Chen et al., 2025). We use Janus-Pro-7B as the teacher and Janus-Pro-1B as the student. We keep the model, data, distillation objective, and training configuration unchanged across rollout methods, and only modify the student rollout procedure. As shown in Table 2, both AOSR and AOSR + CSR substantially improve rollout and end-to-end efficiency while maintaining comparable distillation performance, with CSR providing further acceleration on top of AOSR. These results demonstrate that our method generalizes beyond LlamaGen to a different autoregressive image-generation architecture.

Can AOSR be applied to other speculative decoding methods? To investigate whether AOSR depends on the Jacobi-based proposal mechanism used by SJD, we further integrate AOSR with EAGLE-based student rollout in an OPD setting. We use Qwen3-4B-Instruct-2507 as the teacher, Qwen3-1.7B as the student, and a fixed EAGLE3 drafter corresponding to the student. Task perfor-

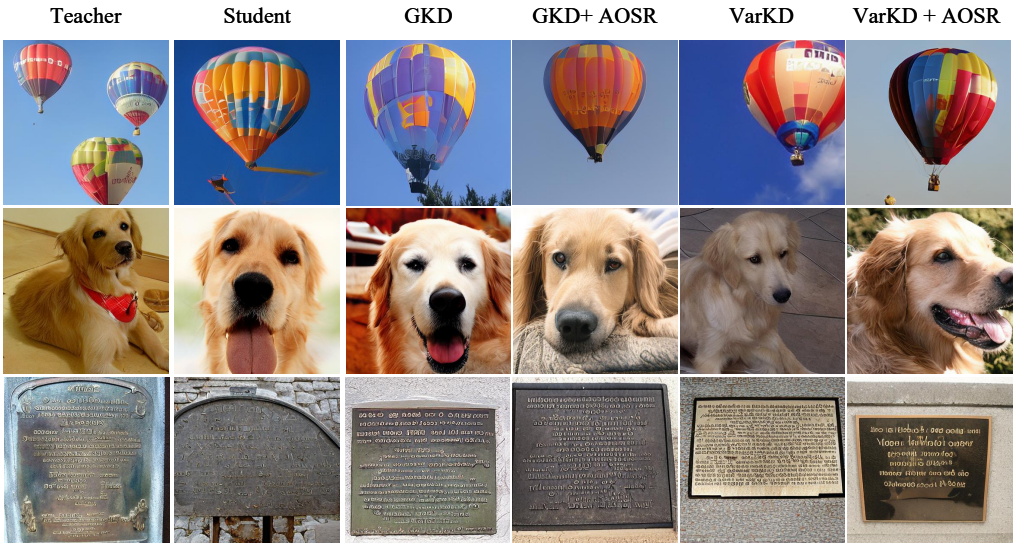


Figure 4: Qualitative comparison of generation results with AOSR.

Table 2: Generalization to Janus-Pro.

Method	FID↓	CLIP↑	Roll.↑	E2E↑
Janus-Pro-1B	32.75	32.05	–	–
GKD (AR)	30.83	32.68	1.00×	1.00×
+ AOSR	30.82	32.64	1.82×	1.34×
+ AOSR + CSR	30.82	32.63	2.07×	1.48×

Table 3: Compatibility with EAGLE.

Method	Pass@8	Time (ms)	Roll.	E2E
Qwen3-1.7B	6.7%	–	–	–
GKD (AR)	50.00%	6494	1.00×	1.00×
+ EAGLE	53.33%	5323	1.22×	1.20×
+ AOSR	53.33%	3912	1.66×	1.60×

mance is evaluated on AIME2024 using Pass@8, where a problem is considered solved if at least one of eight sampled responses is correct. As shown in Table 3, EAGLE already improves rollout efficiency over Cached AR, while adding AOSR provides further rollout and end-to-end acceleration without reducing task performance. These results show that AOSR can be effectively combined with a learned speculative drafter and is not specific to SJD.

How much does each component contribute to AOSR? To evaluate the contribution of each component, we progressively introduce draft refinement, asynchronous advancement, and batch adaptation on top of batch-synchronous speculative rollout. As shown in Table 4, draft refinement improves efficiency, but the rollout remains limited by shared synchronization boundaries. Enabling asynchronous advancement substantially reduces latency by allowing each sequence to continue from its own verified position, while batch adaptation provides a further gain by removing unnecessary computation after sequences finish. These results show that shared-window waiting is the main bottleneck, and that asynchronous advancement provides the largest improvement.

How much of the next rollout should CSR pre-generate? We vary the CSR prefetch length and measure both the number of prefetched tokens retained after the student update and the resulting end-to-end training speedup. As shown in Table 5, increasing the prefetch length allows more computation from the next rollout to be moved into the tail of the current one. However, the gain gradually saturates for longer prefixes because tokens farther from the verified prefix are less likely to remain reusable after the student update, while additional pre-generation also introduces extra computation. Thus, moderate prefetching captures most CSR gains efficiently.

What is the computational and memory overhead of AOSR and CSR? To better expose the runtime and memory overhead of different rollout backends, we evaluate them under a 100% rollout workload using the same model and batch configuration. As shown in Table 6, AOSR substantially

Table 4: Ablation of AOSR components.

Method	Rollout Time (ms)↓	Rollout Spd.↑
GKD (AR)	8003.57	1.00×
Batch-Sync.	8514.44	0.94×
+ Draft Refine.	5972.81	1.34×
+ Async. Adv.	4125.26	1.94×
+ Batch Adapt.	3904.18	2.05×

Table 5: Effect of CSR prefetch length.

Length	Avg. Acc. Tok.	E2E Spd.↑
0 (AOSR)	0.00	1.00×
16	15.01	1.13×
32	27.30	1.15×
64	45.44	1.16×
128	62.50	1.17×

Table 6: Computational and memory overhead during student rollout.

Method	Rollout Time (ms)↓	Rollout Spd.↑	Peak Rollout Mem. (GB)↓	ΔMem.↓
GKD (AR)	8003.57	1.00×	7.92	0.00%
+ AOSR	3904.18	2.05×	7.83	-1.08%
+ AOSR + CSR	3605.21	2.22×	8.18	+3.26%

Table 7: Effect of re-verification in CSR.

Method	FID↓	IS↑	E2E Spd.↑
AOSR	5.78	160.42	1.35×
AOSR + CSR	5.80	157.65	1.49×
CSR w/o Re-verif.	5.96	152.34	1.57×

Table 8: Comparison with batching baseline.

Method	Time (ms)↓
Batch-Sync. + SJD++	5972.81
Regrouping + SJD++	4313.60
AOSR + SJD++	3904.18

reduces rollout time without increasing peak memory, while CSR provides further acceleration with only a small additional memory overhead from cross-step prefetching. These results show that the efficiency gains do not rely on substantially higher GPU memory consumption.

Is post-update re-verification necessary for CSR? To examine the role of post-update re-verification, we compare standard CSR with a variant that directly retains the prefetched tokens after the student update. As shown in Table 7, removing re-verification provides slightly higher efficiency but noticeably degrades generation quality. In contrast, standard CSR preserves comparable quality while still improving end-to-end efficiency over AOSR. This result shows that re-verification is important for safely reusing tokens generated by the previous student across OPD steps.

Does AOSR remain effective against a stronger batching baseline? To determine whether AOSR's gain comes only from comparison with batch-synchronous execution, we introduce a regrouping-based baseline adapted from the sequence-reorganization strategy of Zhang et al. (2025). This baseline preserves heterogeneous sequence progress and reorganizes sequences for batched verification. All methods use the same SJD++ verifier and rollout configuration. As shown in Table 8, regrouping substantially improves over batch-synchronous execution, while AOSR achieves lower rollout latency under the same verifier. This controlled comparison shows that AOSR provides further latency reduction beyond sequence regrouping under the same rollout workload.

5 Conclusion

We presented **AOSR**, an asynchronous speculative rollout framework for accelerating on-policy distillation of autoregressive image generators. AOSR addresses the heterogeneous verified progress produced by batched speculative rollout through sequence-wise asynchronous advancement and completion-aware batch adaptation, while remaining compatible with stronger speculative verifiers. We further introduced **Cross-Step Speculative Refill (CSR)**, which exploits batch positions released by early-finished sequences to advance the next rollout and re-verifies the prefetched tokens after the student update. Experiments across different teacher scales, distillation objectives, and autoregressive image-generation architectures show that AOSR substantially reduces student-rollout and end-to-end training cost, while CSR provides further acceleration with comparable generation quality. Additional results with EAGLE demonstrate that the asynchronous execution principle of AOSR is not limited to SJD and can also benefit speculative rollout with a learned drafter.

AI USAGE STATEMENT

Generative AI tools were used to assist with parts of the code implementation and to improve the clarity and presentation of the manuscript. They were not used to make scientific conclusions or replace the authors' independent research decisions. All AI-generated or AI-assisted content was independently reviewed, verified, and approved by the authors, who take full responsibility for the correctness and integrity of the work.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Xiaokang Chen, Zhiyu Wu, Xingchao Liu, Zizheng Pan, Wen Liu, Zhenda Xie, Xingkai Yu, and Chong Ruan. Janus-pro: Unified multimodal understanding and generation with data and model scaling. *arXiv preprint arXiv:2501.17811*, 2025.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 32694–32717, 2024.

Yefei He, Feng Chen, Yuanyu He, Shaoxuan He, Hong Zhou, Kaipeng Zhang, and Bohan Zhuang. Zipar: Parallel auto-regressive image generation through spatial locality. *arXiv preprint arXiv:2412.04062*, 2024.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Qinghao Hu, Shang Yang, Junxian Guo, Xiaozhe Yao, Yujun Lin, Yuxian Gu, Han Cai, Chuang Gan, Ana Klimovic, and Song Han. Taming the long-tail: Efficient reasoning rl training with adaptive drafter. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 1933–1948, 2026.

Hayate Iso, Tiyasa Mitra, Sudipta Mondal, Rasoul Shafipour, Venmugil Elango, Terry Kong, Yuki Huang, Seonjin Na, Izzy Putterman, Benjamin Chislett, et al. Accelerating rl post-training rollouts via system-integrated speculative decoding. *arXiv preprint arXiv:2604.26779*, 2026.

Doohyuk Jang, Sihwan Park, June Yong Yang, Yeonsung Jung, Jihun Yun, Souvik Kundu, Sung-Yub Kim, and Eunho Yang. Lantern: Accelerating visual autoregressive models with relaxed speculative decoding. In *International Conference on Learning Representations*, volume 2025, pp. 37871–37900, 2025.

Jialiang Kang, Han Shu, Wenshuo Li, Yingjie Zhai, and Xinghao Chen. Sjd-pac: Accelerating speculative jacobi decoding via proactive drafting and adaptive continuation. *arXiv preprint arXiv:2603.18599*, 2026.

Yoon Kim and Alexander M Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 1317–1327, 2016.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Haodong Lei, Hongsong Wang, Bingxuan Dai, and Pan Zhou. Pathrelax: Parallel-path relaxed speculative jacobi decoding for accelerating auto-regressive text-to-image generation. *arXiv preprint arXiv:2606.10492*, 2026.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International conference on machine learning*, pp. 19274–19286. PMLR, 2023.

Boya Liao, Ying Li, Siyong Jian, and Huan Wang. Parallel jacobi decoding for fast autoregressive image generation. *arXiv preprint arXiv:2606.05703*, 2026.

Bingshuai Liu, Ante Wang, Zijun Min, Liang Yao, Haibo Zhang, Yang Liu, Xu Han, Peng Li, Anxiang Zeng, and Jinsong Su. Spec-rl: Accelerating on-policy reinforcement learning with speculative rollouts. *arXiv preprint arXiv:2509.23232*, 2025.

Sihwan Park, Doohyuk Jang, Sungyub Kim, Souvik Kundu, and Eunho Yang. Lantern++: Enhancing relaxed speculative decoding with static tree drafting for visual auto-regressive models. *arXiv preprint arXiv:2502.06352*, 2025.

Elia Peruzzo, Aritra Bhowmik, Guillaume Sautiere, Yuki M Asano, and Amirhossein Habibian. Knowledge distillation for visual autoregressive models. *arXiv preprint arXiv:2606.06078*, 2026.

Haifeng Qian, Sujan Kumar Gonugondla, Sungsoo Ha, Mingyue Shang, Sanjay Krishna Gouda, Ramesh Nallapati, Sudipta Sengupta, Xiaofei Ma, and Anoop Deoras. Bass: Batched attention-optimized speculative sampling. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 8214–8224, 2024.

Bingqi Shan, Baoquan Zhang, Xiaochen Qi, Xutao Li, Yunming Ye, and Liqiang Nie. Sjd-vp: Speculative jacobi decoding with verification prediction for autoregressive image generation. *arXiv preprint arXiv:2603.27115*, 2026.

Junhyuk So, Hyunho Kook, Chaeyeon Jang, and Eunhyeok Park. Speculative coupled decoding for training-free lossless acceleration of autoregressive visual generation. *arXiv preprint arXiv:2510.24211*, 2025a.

Junhyuk So, Juncheol Shin, Hyunho Kook, and Eunhyeok Park. Grouped speculative decoding for autoregressive image generation. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15375–15384. IEEE, 2025b.

Peize Sun, Yi Jiang, Shoufa Chen, Shilong Zhang, Bingyue Peng, Ping Luo, and Zehuan Yuan. Autoregressive model beats diffusion: Llama for scalable image generation. *arXiv preprint arXiv:2406.06525*, 2024.

Yao Teng, Han Shi, Xian Liu, Xuefei Ning, Guohao Dai, Yu Wang, Zhenguo Li, and Xihui Liu. Accelerating auto-regressive text-to-image generation with training-free speculative jacobi decoding. *arXiv preprint arXiv:2410.01699*, 2024.

Yao Teng, Zhihuan Jiang, Han Shi, Xian Liu, Xuefei Ning, Guohao Dai, Yu Wang, Zhenguo Li, and Xihui Liu. Sjd++: Improved speculative jacobi decoding for training-free acceleration of discrete auto-regressive text-to-image generation. *arXiv preprint arXiv:2512.07503*, 2025.

Yao Teng, Fu-Yun Wang, Xian Liu, Zhekai Chen, Han Shi, Yu Wang, Zhenguo Li, Weiyang Liu, Difan Zou, and Xihui Liu. Speculative jacobi-denoising decoding for accelerating autoregressive text-to-image generation. *Advances in Neural Information Processing Systems*, 38:79670–79695, 2026.

Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. *Advances in neural information processing systems*, 37:84839–84865, 2024.

Zili Wang, Zheng Zhang, Kun Ding, Qi Yang, Fei Li, and Shiming Xiang. Continuous speculative decoding for autoregressive image generation. In *European Conference on Computer Vision*, pp. 558–574. Springer, 2026.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX symposium on operating systems design and implementation (OSDI 22)*, pp. 521–538, 2022.

Zhehao Yu, Baoquan Zhang, Bingqi Shan, Xinhao Liu, Dongliang Zhou, Guotao Liang, Guangming Ye, and Yunming Ye. Sjd-pv: Speculative jacobi decoding with phrase verification for autoregressive image generation. *arXiv preprint arXiv:2603.06666*, 2026.

Qianhao Yuan, Jie Lou, Xing Yu, Hongyu Lin, Le Sun, Xianpei Han, and Yaojie Lu. Vision-opd: Learning to see fine details for multimodal llms via on-policy self-distillation. *arXiv preprint arXiv:2605.18740*, 2026.

Baoquan Zhang, Bingqi Shan, Shihao Fang, Kenghong Lin, Xutao Li, and Yunming Ye. Sjd-sv: Speculative jacobi decoding with semantics verification for autoregressive image generation. *arXiv preprint arXiv:2609.13245*, 2026.

Ranran Haoran Zhang, Soumik Dey, Ashirbad Mishra, Hansi Wu, Binbin Li, and Rui Zhang. Batch speculative decoding done right. *arXiv preprint arXiv:2510.22876*, 2025.

A Appendix

A.1 Preliminaries

A.1.1 On-Policy Distillation

Let $\mathbf{x}_{1:T} = (x_1, \ldots, x_T)$ denote a token sequence. An autoregressive model factorizes its distribution as

$$p_\theta(\mathbf{x}_{1:T}) = \prod_{t=1}^{T} p_\theta(x_t \mid \mathbf{x}_{<t}). \tag{5}$$

We consider a fixed teacher p_{T} and a student $p_{\mathrm{S},\theta}$ defined over the same token vocabulary (Agarwal et al., 2024; Peruzzo et al., 2026). In our setting, $\mathbf{x}$ corresponds to a sequence of discrete image tokens produced by a visual tokenizer.

In on-policy distillation, part of the training sequence is generated by the current student rather than taken entirely from the data distribution or generated by the teacher. Given a data prefix $\mathbf{x}^{\mathrm{data}}_{1:k}$, the student generates the remaining suffix:

$$p^{k}_{\mathrm{mix},\theta}(\tilde{\mathbf{x}}_{1:T}) = p_{\mathrm{data}}(\tilde{\mathbf{x}}_{1:k}) \prod_{t=k+1}^{T} p_{\mathrm{S},\theta}(\tilde{x}_t \mid \tilde{\mathbf{x}}_{<t}). \tag{6}$$

The resulting sequence is then used as context for teacher–student distillation. Following our training setting, we use bidirectional KL divergence between the teacher and student distributions:

$$\mathcal{L}_{\mathrm{OPD}} = \mathbb{E}_{\tilde{\mathbf{x}} \sim p^{k}_{\mathrm{mix},\theta}} \left[\frac{1}{T-k} \sum_{t=k+1}^{T} \mathcal{D}_{\mathrm{biKL}} \left(p_{\mathrm{S},\theta}(\cdot \mid \tilde{\mathbf{x}}_{<t}), p_{\mathrm{T}}(\cdot \mid \tilde{\mathbf{x}}_{<t}) \right) \right], \tag{7}$$

where $\mathcal{D}_{\mathrm{biKL}}(p, q) = \frac{1}{2}\mathrm{KL}(p\|q) + \frac{1}{2}\mathrm{KL}(q\|p)$. Since the student suffix must be generated online before distillation can proceed, conventional autoregressive rollout introduces $T - k$ sequential decoding steps into the training critical path.

A.1.2 Speculative Jacobi Decoding

Speculative decoding reduces autoregressive decoding steps by proposing multiple future tokens and verifying them in parallel (Leviathan et al., 2023; Chen et al., 2023). Speculative Jacobi Decoding (SJD) removes the auxiliary draft model by reusing predictions from previous Jacobi iterations as draft tokens (Teng et al., 2024). In each iteration, a Jacobi window of draft tokens is evaluated in one model forward and verified from left to right. Accepted tokens are appended to the generated sequence, while the remaining tokens are refined for the next iteration. By committing multiple tokens in some iterations, SJD reduces the number of sequential model forwards required for autoregressive generation. The original SJD is designed for single-sequence inference, while our work extends it to batched visual OPD rollouts.

A.2 Measurement of Shared-Boundary Waiting

We measure shared-boundary waiting using the same setup as the motivation experiment in Sec. 3.1. We use the final student checkpoint, a batch of 64 ImageNet samples, a 288-token prefix, and a 288-token student-generated suffix. The batch-synchronous speculative rollout uses a window size of $W = 16$ and a history offset of $H = 24$.

At each verifier round, we record the state of every sequence in the batch. A sequence is marked as *waiting* if it has already completed the current rollout window but cannot enter the next window because at least one other sequence has not yet reached the shared window boundary. We define the waiting ratio as

$$\rho_{\mathrm{wait}} = \frac{\sum_r \sum_b \mathbb{I}[s_{b,r} = \mathrm{waiting}]}{\sum_r \sum_b 1}, \tag{8}$$

where b indexes sequences and r indexes verifier rounds. Under the measured rollout, 46.15% of the recorded sequence states are waiting at shared window boundaries.

A.3 Implementation Details of Cross-Step Speculative Refill

Cross-Step Speculative Refill (CSR) reuses the idle batch positions created by asynchronous completion to advance the next student rollout. Here we provide the complete execution procedure used in our experiments. When CSR is enabled, we keep a fixed set of physical batch positions rather than switching between Full and Compact execution. Once a sequence from the current rollout finishes, its position becomes available. At the next scheduling point, a sample from the next rollout batch is prefetched into this position. After prefilling the real image prefix, the prefetched sequence participates in the same batched AOSR/SJD forward as the unfinished sequences of the current rollout.

For each prefetched sequence, CSR generates at most K future image tokens, where we use $K = 16$ by default. Once a sequence reaches this limit, its prefetched tokens are stored and the corresponding physical position can be reused by another sample from the next rollout batch. Importantly, prefetching is only auxiliary work performed during the tail of the current rollout. Therefore, CSR terminates immediately once all sequences of the current rollout have completed. A sequence that has generated fewer than K tokens retains its actual prefetched prefix, while samples that have not yet entered the batch simply have zero prefetched tokens. Prefetched results are associated with their sample indices in the next rollout batch rather than with physical batch positions.

After the current rollout is completed, the usual OPD update produces a new student parameterized by θ_{t+1}. Since the prefetched tokens were generated by the previous student θ_t, they are not directly committed. Instead, they are treated as speculative candidates and re-verified using θ_{t+1}. Under the probabilistic sampling used in our main experiments, CSR re-verifies the prefetched candidates against the updated student's generated tokens from left to right. All consecutive matching tokens before the first mismatch are accepted. At the first mismatch, the candidate is replaced by a token sampled from the updated student's predictive distribution. The remaining prefetched tokens are retained only as drafts for subsequent SJD++ refinement and verification, rather than being directly committed. The greedy variant follows the same re-verification procedure but selects the highest-probability token instead of sampling from the distribution. If all prefetched tokens match, the entire prefetched prefix is retained; if no tokens were prefetched, generation starts normally from the real prefix.

The KV-cache produced before the student update is not reused across parameter updates. Re-verification is performed with the updated student, after which each sequence resumes AOSR from its own committed decoding position. CSR therefore moves part of the next rollout computation into otherwise underutilized positions of the current rollout, while post-update re-verification ensures that the committed sequence remains consistent with the updated student.

A.4 Ablation Details

A.4.1 Configurations of Main-Text Ablations

We summarize the configurations and measurement scopes of the main-text experiments in Table 9. Unless otherwise specified, the image-generation experiments use BF16, classifier-free guidance with scale 2.0, and a Jacobi window size of $W = 16$. Each rollout benchmark fixes the student checkpoint and input workload across the compared methods, except for the component being evaluated. Training comparisons use the teacher–student pairs and objectives specified in Sec. 4.1.

AOSR components and batching baselines. Table 4 first adds the SJD++ draft refinement to batch-synchronous SJD, then enables sequence-wise asynchronous advancement and completion-aware batch adaptation. This ordering evaluates the two execution designs on top of the same refined verifier. Table 8 compares batch-synchronous, regrouping-based, and AOSR execution with SJD++ under a matched rollout configuration. The regrouping baseline adapts the sequence-reorganization principle rather than reproducing an entire serving system. Its grouping rule and KV-cache handling are kept fixed throughout the comparison.

CSR prefetching and re-verification. Table 5 varies the maximum prefetched prefix length. The retained-token metric reports the average number of prefetched tokens accepted after the student update, rather than an acceptance percentage. Its end-to-end speedup is relative to AOSR without CSR. The no-re-verification experiment in Table 7 uses LlamaGen-XL as the teacher and LlamaGen-

Algorithm 1 Cross-Step Speculative Refill

Require: Current rollout batch $\mathcal{B}_t$, next batch $\mathcal{B}_{t+1}$, student θ_t, prefetch length K

Initialize all samples in $\mathcal{B}_t$ in fixed batch positions
Initialize the prefetch length of each sample in $\mathcal{B}_{t+1}$ to zero
while some sequence in $\mathcal{B}_t$ is unfinished **do**
    for each available batch position **do**
        if an unprocessed sample exists in $\mathcal{B}_{t+1}$ **then**
            Prefill its real prefix and insert it into the available position
        end if
    end for
    Advance unfinished current sequences and active prefetched sequences together using one batched AOSR/SJD step
    for each prefetched sequence that reaches K generated tokens **do**
        Store its prefetched tokens and release its batch position
    end for
    Release positions of newly completed sequences in $\mathcal{B}_t$
end while
Stop prefetching and retain all partially prefetched sequences
Update the student: $\theta_t \rightarrow \theta_{t+1}$
for each sample in $\mathcal{B}_{t+1}$ **do**
    if prefetched tokens are available **then**
        Re-verify the prefetched tokens from left to right using θ_{t+1}
        Commit the consecutive matching prefix
        if the first mismatch exists **then**
            Generate and commit a replacement token from θ_{t+1} using the configured sampling rule
            Keep the remaining candidates only as SJD++ drafts
        end if
    end if
end for
Continue the next rollout with AOSR from the resulting sequence-wise positions

Table 9: Configurations of the main-text experiments. Batch sizes are per GPU. The global batch is 256 for the four-GPU LlamaGen training runs and 64 for Janus-Pro.

Experiment	Data	Batch/GPU	Protocol
LlamaGen main results	ImageNet-1K	64	4 GPUs; training
Janus-Pro	COCO2017	16	4 GPUs; training
EAGLE	DAPO-Math-17k	32	1024 responses/update
AOSR components	Fixed ImageNet inputs	64	4 GPUs; rollout only
CSR prefetch length	ImageNet-1K	64	4 GPUs; training
Time and memory	Fixed ImageNet inputs	64	4 GPUs; rollout only
CSR re-verification	ImageNet-1K	64	4 GPUs; 22K steps
Regrouping baseline	Fixed ImageNet inputs	–	Rollout only

B initialized from an existing KD checkpoint as the student. It uses GKD on ImageNet, a prefetch length of $K = 16$, and 22K training steps with 64 samples per GPU on four GPUs. This variant directly retains the prefetched tokens after the update instead of re-verifying them with the updated student.

Decoding and sampling policy. In the main experiments, Cached AR and the speculative rollout methods use the same probabilistic token-sampling policy. CSR re-verifies prefetched candidates from left to right after the student update, retaining the consecutive matching prefix. At the first mismatch, a replacement token is generated from the updated student. Under probabilistic sampling, this token is sampled from the predictive distribution, while the greedy variant selects the highest-probability token. Thus, the two settings differ only in token selection and use the same CSR re-verification and execution procedure.

Table 10: Generation quality and efficiency with probabilistic and greedy token sampling. Cached AR is included as a common reference, while the speculative rows evaluate sensitivity to the sampling strategy.

Method	Sampling	FID↓	IS↑	Time (ms)↓	Roll.↑	E2E↑
GKD (AR)	–	5.77	157.68	4122	1.00×	1.00×
+ AOSR	Probabilistic	5.78	160.42	2179	1.89×	1.35×
+ AOSR + CSR	Probabilistic	5.80	157.65	1808	2.27×	1.49×
+ AOSR	Greedy	5.80	162.50	2169	1.90×	1.33×
+ AOSR + CSR	Greedy	5.79	158.38	1800	2.29×	1.48×

Resource usage. In Table 6, rollout latency and peak GPU memory are measured over the student-generation region under the same workload. The relative memory change is measured against Cached AR. These measurements characterize rollout resource usage and are separate from the end-to-end training-step measurements.

Janus-Pro and EAGLE evaluation. The Janus-Pro experiment uses COCO2017 and reports FID and CLIP score. For EAGLE, we use Qwen3-4B-Instruct-2507 as the teacher, Qwen3-1.7B as the student, and a fixed EAGLE3 drafter corresponding to the student. Training uses DAPO-Math-17k, with 1024 responses collected per update. Task performance is evaluated on AIME2024 using Pass@8: a problem is counted as solved if at least one of eight sampled responses is correct. All compared methods use the same evaluation sampling settings and response budget.

Table 11: Additional generation-quality metrics for the ImageNet settings in Table 1. The student is LlamaGen-B in every setting.

Teacher	Objective	Rollout	Precision↑	Recall↑
LlamaGen-L	GKD	Cached AR	0.80	0.48
		AOSR	0.80	0.47
		AOSR + CSR	0.79	0.49
	VarKD	Cached AR	0.81	0.47
		AOSR	0.83	0.47
		AOSR + CSR	0.82	0.48
LlamaGen-XL	GKD	Cached AR	0.79	0.49
		AOSR	0.79	0.49
		AOSR + CSR	0.79	0.48
	VarKD	Cached AR	0.81	0.47
		AOSR	0.82	0.44
		AOSR + CSR	0.82	0.46

A.4.2 Additional Ablation Results

We provide additional comparisons of verification strategies and generation-quality metrics. We also report precision and recall for the ImageNet training settings in the main text. The experiments below complement the main-text component, prefetch-length, resource, re-verification, and batching comparisons without repeating them.

How do probabilistic and greedy token sampling compare in quality and efficiency? We compare probabilistic and greedy token sampling for both AOSR and AOSR + CSR. Table 10 reports generation quality together with rollout latency and end-to-end step speedup. Cached AR is included as a common reference, while the speculative rows evaluate whether the observed quality and efficiency are sensitive to the token-sampling strategy.

Do precision and recall support the generation-quality comparison? We report precision and recall for the teacher–student pairs and distillation objectives in Table 1. Table 11 complements the main-text FID and IS evaluation using the same generation-quality evaluation protocol.