Enabling 8B Bitwise Autoregressive Image Generation on Edge GPUs – Implementation and Reproducibility Notes

Enrico Vezzali[1,2], Costantino Grana[1], and Federico Bolelli[1]

[1] AImageLab, University of Modena and Reggio Emilia, Italy
{name.surname}@unimore.it
[2] Datalogic, S.p.A., Bologna, Italy

Abstract. This paper presents the technical reproducibility protocol for the research presented in "Enabling 8B Bitwise Autoregressive Image Generation on Edge GPUs"[19]. We provide a comprehensive suite of tools, named `VAR-Compressor`, designed to replicate the W4A4 weight-activation quantization and the INT8 KV-cache compression strategy for the Infinity Visual Autoregressive (VAR) model family. Our implementation reduces the Infinity-8B footprint by 64% (37.1GB → 13.3GB), fitting it on the mid-range Orin NX, while retaining similar Image Reward and CLIP scores to the FP16 model. This report details how to set up the repository, quantize the model, and benchmark both the generative fidelity and the hardware efficiency results on NVIDIA Jetson architectures. The referenced code and trained models are available at https://github.com/Henvezz95/VAR-Compressor.

Keywords: Reproducible Research · Quantization · Edge AI · Visual Autoregressive Models · SVDQuant.

1 Introduction

Recent advances in Visual AutoRegressive (VAR) models [6,16,17] have challenged the dominance of diffusion models in high-resolution image synthesis. By treating visual generation as a sequence of discrete tokens across progressive scales, models like Infinity [6] achieve exceptional visual quality. Yet, the underlying autoregressive mechanism introduces a prohibitive computational trade-off: a monotonically expanding Key-Value (KV) cache. This characteristic sets up a massive memory wall that differentiates VAR from diffusion pipelines. For instance, synthesis of a 1024×1024 resolution image via an uncompressed 8B parameter framework demands upwards of 37 GB in FP16 format, pricing these systems out of commodity edge devices and anchoring them to data centers.

This memory barrier severely limits deployment across Edge AI domains like robotics [4,9], semantic compression [13,15], and privacy-critical consumer applications [5]. Consequently, executing flagship architectures on edge silicon like the 16 GB NVIDIA Jetson Orin NX remains a major challenge.

While quantization is well-established for LLMs, its application to VAR is nascent and faces distinct hurdles. Existing methods either suffer from quality loss at 4-bit precision [23] or rely on non-standard floating-point formats requiring custom FPGA co-design [21]. Moreover, token-centric KV-cache quantization methods like KIVI [12] fail to handle the channel-driven variance native to visual autoregressive networks.

To address these limitations, the foundational ICPR paper [19] introduced a specialized compression pipeline. By profiling Infinity's 8B architecture, we isolated massive activation outliers. We decoupled these outliers via an SVD-Quant [11] high-precision low-rank branch to enable stable W4A4 execution, while applying an Asymmetric Per-Channel INT8 quantization to accommodate the highly skewed distributions of the KV-cache. This hybrid approach compressed the Infinity-8B footprint by 64% down to 13.3 GB, allowing native execution on commodity edge platforms.

This report provides the reproducibility protocol and code for this pipeline via `VAR-Compressor`, a specialized fork of the `deepcompressor`[3] library Modified to support the structural nuances of VAR models and asymmetric KV-caches, this codebase is strictly additive and retains backward compatibility for diffusion models like Flux.1-dev [1], PixArt-Σ [2], Stable Diffusion XL [14], and SANA [22]. The consolidated replication workflow encompasses:

- **Post-Training Quantization (PTQ)**: Isolating Infinity 2B/8B activation outliers into low-rank branches.
- **KV-Cache Calibration**: Employing Golden-Section Search to optimize asymmetric INT8 scaling and zero-points.
- **Generative Evaluation**: Benchmarking quality metrics (ImageReward, CLIP-IQA) on MJHQ-30K and sDCI via fake-quantization.
- **Hardware Profiling**: Profiling real-world footprint and latency on the NVIDIA Jetson platform using compiled W4A4 kernels.

2 Environment and Installation

The reproducibility of our results involves two distinct hardware and software tiers, separated by their computational intensity and role in the pipeline.

2.1 Server-Side: Quantization and Calibration

The Post-Training Quantization (PTQ) and KV-cache calibration phases involve high-order optimizations and the caching of large activation tensors. These operations require server-class hardware:

- **Hardware Requirements**: A high-compute server environment (e.g., NVIDIA H100 or A100) is necessary to accommodate the peak memory overhead of the calibration process of the 8B model.
- **Software**: Standard Linux environments with CUDA 12.x and PyTorch 2.1+ are sufficient for these stages.

[3] https://github.com/mit-han-lab/deepcompressor

2.2 Edge-Side: Inference and Docker Infrastructure

The hardware efficiency and latency benchmarks can be conducted natively on edge hardware. In particular, the tests described in the original paper were conducted on an NVIDIA Jetson AGX Orin 64GB. To ensure a reproducible software environment specifically for the Jetson platform, we suggest a Docker-based deployment strategy intended solely for inference.

Rather than building the environment from scratch—which on ARM64 architectures is highly prone to dependency conflicts and compilation errors—we utilize the `dustynv/pytorch` base images (e.g., `dustynv/pytorch:2.6-r36-.4.0-cu128`). These images provide pre-installed, ARM64-optimized builds of PyTorch and CUDA. The provided `Dockerfile` extends this base to guarantee a seamless reproducibility pipeline. Key architectural requirements include:

- **L4T Compatibility**: The container's base image must strictly match the host's Linux for Tegra (L4T) release version.
- **Permission Synchronization**: The `Dockerfile` is configured to pass the host's UID and GID during the build phase. This prevents permission conflicts, ensuring that output artifacts (e.g., evaluation JSONs and generated images) are accessible on the host machine without root privileges.
- **GPU Visibility**: Before executing the hardware benchmarks, users must verify that the container correctly interfaces with the Jetson GPU runtime:

```
python3 -c "import torch; print(torch.cuda.is_available())"
```

2.3 Installation from Source

To initialize the `VAR-Compressor` suite on either platform:

1. **Clone the repository**:

```
git clone https://github.com/Henvezz95/VAR-Compressor.git
cd VAR-Compressor
```

2. **Install core dependencies**:

```
pip install -e .
```

3. **Initialize model-specific submodules**:

```
cd Infinity && pip install -r requirements.txt
```

2.4 Pre-Computed Artifacts and Weights

To guarantee exact bit-level reproducibility of the hardware benchmarks—and to allow reviewers to bypass the computationally expensive server-side Post-Training Quantization phase—all required artifacts are provided as frozen snapshots.

- **FP16 Golden Reference**: To prevent reproducibility failures caused by upstream model retraining or silent checkpoint updates, we provide a frozen snapshot of the exact original Infinity weights (FoundationVision/Infinity) used in our experiments. These must be downloaded from our repository links and placed strictly inside the `Infinity_rep/weights/` directory.
- **PTQ Artifacts**: The quantized INT4 weights (`model.pt`), SVD low-rank branches (`branch.pt`), and activation smoothing scales (`smooth.pt`) must be downloaded from the `VAR-Compressor` repository and extracted into the `./runs/diffusion/` directory.
- **KV-Cache Artifacts**: The optimized asymmetric cache scales (`kv_quant_calib.pt`) must be downloaded and placed into the `./runs/kv_scales/` directory.

3 Diagnostic Analysis of Model Distributions

To verify the structural necessity of the proposed quantization strategies, the repository provides an auditing script, `activations_measurements.py`. This script hooks into the model's forward path using an instance of `InfinityCalibManager`, configured to capture hidden activations of the linear layers. Reviewers can run this profile via:

```
python activations_measurements.py  \
--config configs/models/infinity-8b.yaml
```

The tool aggregates layer-wise metrics across the multi-scale generation steps and serializes the complete distribution profiles into a structured JSON file (`infinity_stats_8b.json`) while printing summarized statistical invariants to the terminal.

3.1 Activation Outliers and Kurtosis

Large Transformer-based models often exhibit the "Massive Activation" phenomenon [3], where specific channels in linear layers develop extreme magnitude spikes. The diagnostic suite profiles these anomalies by tracking the Absolute Max-to-Median Ratio, defined mathematically as:

$$\eta = \frac{\max(|X|)}{\text{median}(|X|)}$$

Simultaneously, it measures the excess Kurtosis of the flattened layer tensors to quantify the extreme heavy-tailedness of these distributions.

Table 1. Activation Outlier Analysis. We report the Layer-wise Absolute Max-to-Median Ratio and Kurtosis profiled on 12K calibration tokens.

Layer Type	Infinity 2B				Infinity 8B			
	Max/Median		Kurtosis		Max/Median		Kurtosis	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
QKV Projection	30.6	47.2	5.55	10.89	37.2	59.4	6.59	21.97
Out Projection	23.8	128.0	2.72	24.67	27.6	195.4	3.40	23.90
FFN Up-Proj	24.4	36.5	2.99	7.99	24.0	46.6	2.10	12.24
FFN Down-Proj	**98.2**	**353.0**	**26.41**	**153.80**	**63.4**	**226.3**	**26.13**	**113.50**

As shown in Table 1, running this execution script isolates the FFN Down-Projection (`ffn.fc2`) as the primary structural bottleneck for stable quantization. For the Infinity-8B model, the extracted absolute Max-to-Median ratio reaches 226.3× with a mean Kurtosis peaking at 113.50 (surpassing 353× and 153.80 in the 2B variants). This massive tail anomaly confirms that conventional uniform weight-activation clipping would introduce catastrophic reconstruction noise, mathematically justifying the isolation of these outliers into the low-rank branch handled by SVDQuant [11].

3.2 KV-Cache Variance and Symmetry

The stability of our compressed attention infrastructure relies on characterising the multi-axis variance of the Dynamic Range ($R = \max(\mathbf{x}) - \min(\mathbf{x})$) within the sequential Key and Value caches. The profiling script isolates these dynamics using two metrics:

1. **Channel Dominance**: The script calculates the Coefficient of Variation ($CV = \sigma/\mu$) of the range R across individual channels (CV_{chan}) and token indices (CV_{tok}). The execution outputs yield $CV_{chan} \approx 0.31$, a value over 5× higher than the token-wise variation ($CV_{tok} \approx 0.06$). This demonstrates that dynamic range variance is a static property of concrete structural channels rather than an evolving artifact of input context, justifying a static per-channel scaling matrix.
2. **Asymmetric Skew**: To prevent variance cancellation effects where divergent channels average out, the function `analyze_symmetry` calculates the individual Fisher-Pearson skewness coefficient (γ) per column. While the global mean skewness is moderate (≈ 0.8), specific outlier channels exhibit severe directional asymmetry ($|\gamma_{max}| > 7$). Standard symmetric standard quantization grids would consequently waste valuable bit-width mapping empty numerical space, indicating a strict structural requirement for the asymmetric clipping bounds and zero-point shifts implemented in our calibration routine.

4 Quantization and Calibration

The quantization of the Infinity VAR models is performed in two stages: weight-activation quantization via SVD decoupling and asymmetric KV-cache calibration. These memory-intensive operations are conducted on server-side hardware (e.g., NVIDIA H100) to accommodate the activation caching required for the optimization process.

4.1 Weight and Activation Quantization

We utilize the `ptq_infinity.py` module to perform the Post-Training Quantization (PTQ). The framework employs SVDQuant [11] to isolate activation outliers into a high-precision low-rank branch while quantizing the remaining transformer weights to 4-bit.

Configuration Hierarchy and Ablations The execution is managed via a positional configuration override hierarchy, where files passed later in the command override overlapping keys in earlier files. This modularity facilitates the reproduction of the ablation studies reported in the original paper:

- **Proposed SVDQuant (W4A4)** for the 8B Infinity model:

```
python -m deepcompressor.app.diffusion.ptq_infinity \
configs/models/infinity-8b.yaml \
configs/collect/qdiff.yaml \
configs/svdquant/int4.yaml \
--save-model
```

- **Proposed SVDQuant (W4A4)** for the 2B Infinity model: Utilizes `configs/models/infinity-2b.yaml` as the base model configuration.
- **SmoothQuant Baseline**: Utilizes `configs/models/infinity-2b-smoothquant.yaml` (or the 8B equivalent) to mitigate outliers via activation scaling only, without the low-rank branch.
- **Baseline Quantization**: Utilizes `configs/models/infinity-2b-naive.yaml` for standard 64-group block-wise quantization. For simpler per-channel quantization, the group size can be set to -1 in the configuration.

For rapid verification, users can append `configs/svdquant/fast.yaml` to the command to reduce the number of calibration steps. The primary output is the `model.pt` artifact (24 GB for the 8B model), alongside smoothing scales (`smooth.pt`) and SVD branches (`branch.pt`).

4.2 KV-Cache Calibration

To minimize the footprint of the monotonically growing KV-cache, we implement Asymmetric Per-Channel INT8 quantization. Unlike standard LLM methods

that often focus on token-wise outliers, our analysis indicates that VAR model variance is predominantly channel-driven.

Optimization via Golden-Section Search: the calibration routine, defined in the script `calibrate_cache_quantization.py`, optimizes the clipping bounds per channel to minimize the reconstruction Mean Squared Error (MSE). The optimization follows a two-stage coarse-to-fine process:

1. **Grid Search**: Scans a logarithmic grid of percentiles to identify a promising range for the clipping bounds.
2. **Refinement**: Employs a **Golden-Section Search** to refine the optimal bounds, accommodating the skewed distributions identified in Section 3.

The routine is executed as follows:

```
python -m deepcompressor.app.diffusion.calibrate_cache_quantization \
    configs/models/infinity-8b.yaml configs/collect/qdiff.yaml
```

The output is a `kv_quant_calib.pt` file containing the `scale` and `zero_point` parameters. These parameters enable the affine quantization mapping required to align the quantization grid with the axes of highest variance.

5 Evaluation and Verification

To comprehensively validate the proposed compression pipeline, the evaluation methodology is designed to assess the aesthetic impact of quantization noise and hardware-specific efficiency across three primary axes.

5.1 Experimental Setup and Proxy Methodology

Hardware Constraints. Reproducing the results of the 8B model requires a hardware strategy that distinguishes between the uncompressed baseline and the optimized targets. As described in the original work, the FP16 Infinity-8B baseline requires 37.1 GB, exceeding the capacity of standard 16 GB or 32 GB Jetson modules. We utilize a **proxy methodology**: benchmarking is conducted on an NVIDIA Jetson AGX Orin 64GB, allowing us to generate the "Golden Baseline" metrics without Out-Of-Memory (OOM) failures while strictly profiling the peak memory usage relative to the 16 GB (Orin NX) and 8 GB (Orin Nano) hardware envelopes.

Datasets and Sampling. The reproduction scripts support two distinct domains: MJHQ [10] for high-fidelity artistic generation and sDCI [18] for photorealistic prompt adherence. For verification, a fixed subset of 5,000 samples (use `-eval-num-samples 5000`) is enforced to ensure consistent FID [8] and CLIP-IQA [20] distributions.

5.2 Generative Quality (Fake-Quantization)

To verify the aesthetic impact of quantization independently of kernel-level performance, we utilize bit-accurate fake-quantization. The `benchmark_assembled_model.py` script simulates INT4/INT8 logic while executing in `bfloat16` to identify precisely how the noise impacts the ImageReward [24] and CLIP [7] scores.

Activation and Argument Mapping. The script supports modular activation of the quantization components, which is essential for replicating the ablation studies (e.g., assessing the 2B model's sensitivity to activation outliers). To execute the complete W4A4 + KV8 simulation, the following command fuses the SVD low-rank branches and activation scales generated during the PTQ phase:

```
python -m evaluation.benchmark_assembled_model \
    --config configs/models/infinity-8b.yaml \
    --configs/svdquant/int4.yaml \
    --ref-root ./evaluation_output/infinity_fp16_8b \
    --gen-root ./evaluation_output/infinity_w4a4_kv8_8b \
    --base-path ./runs/diffusion/int4_rank32_8b/ \
    --enable_weight_quant true \
    --enable_activation_quant true \
    --enable_kv_quant true \
    --eval-benchmarks MJHQ DCI \
    --eval-num-samples 5000 \
    --eval-gt-metrics clip_iqa clip_score fid image_reward psnr ssim lpips
```

The command-line arguments explicitly control the evaluation axes and quantization targets:

- `--base-path`: Directory containing the PTQ artifacts generated in Section 4 (`model.pt`, `smooth.pt`, and `branch.pt`).
- `--enable_*_quant`: Boolean flags (`weight`, `activation`, `kv`) to independently toggle fake-quantization logic. *Note: `--enable_kv_quant` automatically loads the scales from `runs/kv_scales/kv_quant_calib.pt`; thus, the cache calibration script must be executed prior to this evaluation.*
- `--gen-root`: Destination path for the generated images and the final evaluation `results.json`.
- `--ref-root`: Path to the FP16 ground-truth reference dataset.
- `--eval-gt-metrics`: Selects the evaluation axes. To measure *Generative Quality*, the repository supports ImageReward [24] to quantify alignment with human aesthetic preferences, CLIP Score [7] for semantic text-image consistency, CLIP-IQA [20] for perceptual fidelity, and FID [8] to measure distributional similarity with the reference dataset. To measure the pure quantization noise relative to the FP16 golden baseline (*Reference-Dependent Fidelity*), the script supports PSNR, SSIM, and LPIPS [25].

5.3 Hardware Efficiency (Real Quantization)

To verify the computational load and system efficiency, we measure the actual memory savings natively on the Jetson platform. This stage mandates the use of specialized hardware-accelerated kernels for 4-bit weight and 4-bit activation computation.

Nunchaku Framework Integration Our inference pipeline relies on the high-performance W4A4 integer kernels originally developed in the Nunchaku library [11]. To adapt these kernels for the Infinity architecture, we provide a specialized fork (https://github.com/Henvezz95/nunchaku-fork). Our primary contribution in this fork is the introduction of a custom C++ wrapper and a Python interface class designed to perfectly mirror the standard PyTorch `nn.Linear` API. This extension enables seamless "plug-and-play" integration of the SVDQuant logic into the autoregressive generation loop.

To compile and install this dependency on the Jetson environment, reviewers must execute the following commands before running the hardware benchmarks:

```
git clone https://github.com/Henvezz95/nunchaku-fork.git
cd nunchaku-fork
pip install -e .
```

Executing the Benchmark Once the specialized kernels are compiled, the `infinity_w4a4_test.py` script executes the hardware validation through the following steps:

1. **Model Transformation**: Swaps standard `torch.nn.Linear` layers for our custom `SVDQuantLinear` modules, excluding sensitive embeddings and transformer heads to maintain bitwise stability.
2. **Artifact Injection**: Loads the INT4 quantized weights (`model.pt`) and the high-precision low-rank branches (`branch.pt`) directly into the optimized modules, alongside the calibrated INT8 KV-cache scales.
3. **Footprint Profiling**: Reports the peak system memory via `torch.cuda.max_memory_allocated()`, confirming the 13.3 GB residency required for the Orin NX.

The benchmark is initiated via:

```
python infinity_w4a4_test.py configs/models/infinity-8b.yaml \
    --base-path ./runs/diffusion/int4_rank32_8b/ \
    --enable_kv_quant true \
    --seed 16 \
    --prompt <prompt>
```

Where `<prompt>` can be any string.

6 Qualitative Results

To complement the quantitative metrics, we provide a qualitative comparison across four distinct scenarios: Detailed Portrait, Architectural Geometry, Landscape Gradients, and Object Representation. As shown in Figure 1, the quantized Infinity 8B model (W4A4 + KV8) retains near-FP16 aesthetic quality, preserving fine-grained details such as skin textures and wood grain that are often lost in standard 4-bit quantization schemes.

6.1 Verification of Qualitative Stability

While the examples in Figure 1 represent individual samples from the large-scale evaluation, the reproducibility of these qualitative trends can be verified using the provided `infinity_w4a4_test.py` script.

Furthermore, when executing the `evaluation.benchmark_assembled_model` module, all generated images are automatically saved within the directory specified by the `-gen-root` flag. The specific samples presented in Figure 1 were directly extracted from these evaluation outputs, allowing reviewers to easily cross-reference the visual artifacts with the quantitative metrics computed for the MJHQ and sDCI datasets.

7 Conclusions

This report provides a comprehensive guide on how to utilize the `VAR-Compressor` framework and replicate the results presented in the ICPR paper "Enabling 8B Bitwise Autoregressive Image Generation on Edge GPUs" [19]. Specifically, it details the repository and dataset locations, the Docker-based setup instructions, and the precise execution procedures for both the Post-Training Quantization pipeline and the native hardware benchmarks.

By combining INT4 W4A4 SVDQuant for weights and activations with Asymmetric Per-Channel INT8 quantization for the KV-cache, the provided implementation successfully reduces the peak memory footprint of the Infinity-8B model by 64% (37.1 GB $\rightarrow$ 13.3 GB) without compromising generative fidelity.

The generative quality results and deterministic memory reductions achieved via the fake-quantization scripts should be replicable with high precision on any platform with sufficient VRAM. On the other hand, changes in the edge hardware used to run the native tests will strongly impact the latency and residency results. Achieving the exact 27.0-second inference speed and 13.3 GB memory footprint requires the specified NVIDIA Jetson Orin architecture and the successful compilation of the `nunchaku-fork` C++/CUDA kernels.

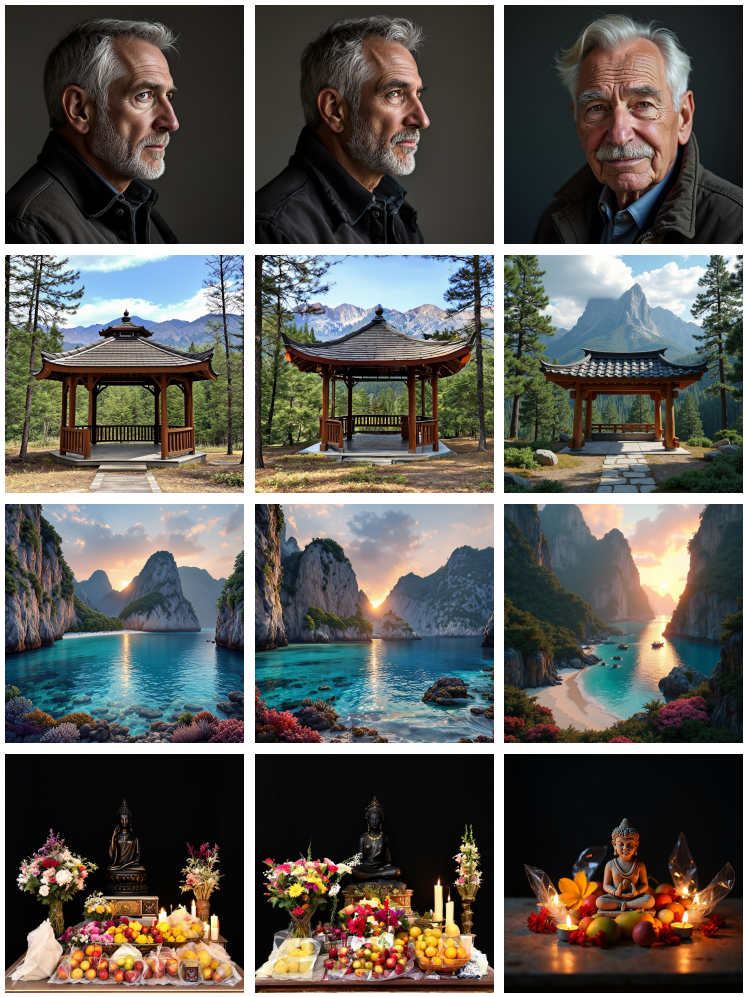

Fig. 1. Qualitative Comparison of Infinity-8B Quantization. Columns compare the Infinity-8B baseline (FP16), our W4A4 INT4 pipeline, and Flux.1-dev (SVDQuant W4A4) as a reference. **Results:** The quantized 8B model preserves near-FP16 fidelity and global composition, showing similar or even superior quality compared to the larger Flux baseline.

Prompts: (1) "Portrait, photograph, canon 5d... middle aged man, realistic" [MJHQ]; (2) "Intricate wooden gazebo, Asian-style tiled roof, mountains, 8k" [sDCI]; (3) "Photorealistic, hidden beach, rock formations, sunset, 4k" [sDCI]; (4) "Close-up of a small shrine, stone statue, flowers, cinematic" [MJHQ]. A fixed seed is used across all models to ensure qualitative parity.

References

1. Black Forest Labs: Flux.1. https://blackforestlabs.ai/ (2024)
2. Chen, J., Ge, C., Xie, E., Wu, Y., Yao, L., Ren, X., Wang, Z., Luo, P., Lu, H., Li, Z.: Pixart-σ: Weak-to-strong training of diffusion transformer for 4k text-to-image generation. In: European Conference on Computer Vision. pp. 74–91. Springer (2024)
3. Dettmers, T., Lewis, M., Belkada, Y., Zettlemoyer, L.: Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. Advances in neural information processing systems **35**, 30318–30332 (2022)
4. Firoozi, R., Tucker, J., Tian, S., Majumdar, A., Sun, J., Liu, W., Zhu, Y., Song, S., Kapoor, A., Hausman, K., et al.: Foundation models in robotics: Applications, challenges, and the future. The International Journal of Robotics Research **44**(5), 701–739 (2025)
5. Golda, A., Mekonen, K., Pandey, A., Singh, A., Hassija, V., Chamola, V., Sikdar, B.: Privacy and security concerns in generative ai: a comprehensive survey. IEEE Access **12**, 48126–48144 (2024)
6. Han, J., Liu, J., Jiang, Y., Yan, B., Zhang, Y., Yuan, Z., Peng, B., Liu, X.: Infinity: Scaling bitwise autoregressive modeling for high-resolution image synthesis. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 15733–15744 (2025)
7. Hessel, J., Holtzman, A., Forbes, M., Le Bras, R., Choi, Y.: Clipscore: A reference-free evaluation metric for image captioning. In: Proceedings of the 2021 conference on empirical methods in natural language processing. pp. 7514–7528 (2021)
8. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. Advances in neural information processing systems **30** (2017)
9. Kapelyukh, I., Vosylius, V., Johns, E.: Dall-e-bot: Introducing web-scale diffusion models to robotics. IEEE Robotics and Automation Letters **8**(7), 3956–3963 (2023)
10. Li, D., Kamko, A., Akhgari, E., Sabet, A., Xu, L., Doshi, S.: Playground v2. 5: Three insights towards enhancing aesthetic quality in text-to-image generation. arXiv preprint arXiv:2402.17245 (2024)
11. Li, M., Lin, Y., Zhang, Z., Cai, T., Li, X., Guo, J., Xie, E., Meng, C., Zhu, J.Y., Han, S.: Svdquant: Absorbing outliers by low-rank components for 4-bit diffusion models. arXiv preprint arXiv:2411.05007 (2024)
12. Liu, Z., Yuan, J., Jin, H., Zhong, S., Xu, Z., Braverman, V., Chen, B., Hu, X.: Kivi: A tuning-free asymmetric 2bit quantization for kv cache. arXiv preprint arXiv:2402.02750 (2024)
13. Mentzer, F., Toderici, G.D., Tschannen, M., Agustsson, E.: High-fidelity generative image compression. Advances in neural information processing systems **33**, 11913–11924 (2020)
14. Podell, D., English, Z., Lacey, K., Blattmann, A., Dockhorn, T., Müller, J., Penna, J., Rombach, R.: Sdxl: Improving latent diffusion models for high-resolution image synthesis. arXiv preprint arXiv:2307.01952 (2023)
15. Qin, Z., Tao, X., Lu, J., Tong, W., Li, G.Y.: Semantic communications: Principles and challenges. arXiv preprint arXiv:2201.01389 (2021)
16. Tang, H., Wu, Y., Yang, S., Xie, E., Chen, J., Chen, J., Zhang, Z., Cai, H., Lu, Y., Han, S.: Hart: Efficient visual generation with hybrid autoregressive transformer
17. Tian, K., Jiang, Y., Yuan, Z., Peng, B., Wang, L.: Visual autoregressive modeling: Scalable image generation via next-scale prediction. Advances in neural information processing systems **37**, 84839–84865 (2024)

18. Urbanek, J., Bordes, F., Astolfi, P., Williamson, M., Sharma, V., Romero-Soriano, A.: A picture is worth more than 77 text tokens: Evaluating clip-style models on dense captions. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 26700–26709 (2024)
19. Vezzali, E., Bolelli, F., Grana, C., Benini, L., Yawei, L., et al.: Enabling 8B Bitwise Autoregressive Image Generation on Edge GPUs. In: 28th International Conference on Pattern Recognition (2026)
20. Wang, J., Chan, K.C., Loy, C.C.: Exploring clip for assessing the look and feel of images. In: Proceedings of the AAAI conference on artificial intelligence. vol. 37, pp. 2555–2563 (2023)
21. Wei, R., Xu, S., Guo, Q., Li, M.: Fpqvar: Floating point quantization for visual autoregressive model with fpga hardware co-design. arXiv preprint arXiv:2505.16335 (2025)
22. Xie, E., Chen, J., Chen, J., Cai, H., Tang, H., Lin, Y., Zhang, Z., Li, M., Zhu, L., Lu, Y., et al.: Sana: Efficient high-resolution image synthesis with linear diffusion transformers. arXiv preprint arXiv:2410.10629 (2024)
23. Xie, R., Zhao, T., Yuan, Z., Wan, R., Gao, W., Zhu, Z., Ning, X., Wang, Y.: Litevar: Compressing visual autoregressive modelling with efficient attention and quantization. arXiv preprint arXiv:2411.17178 (2024)
24. Xu, J., Liu, X., Wu, Y., Tong, Y., Li, Q., Ding, M., Tang, J., Dong, Y.: Imagereward: Learning and evaluating human preferences for text-to-image generation. Advances in Neural Information Processing Systems **36**, 15903–15935 (2023)
25. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 586–595 (2018)