DiffPro: Joint Timestep and Layer-Wise Precision Optimization for Efficient Diffusion Inference

Farhana Amin[1], Sabiha Afroz[1], Kanchon Gharami[2], Mona Moghadampanah[1], and Dimitrios S. Nikolopoulos[1]

[1] Virginia Tech, Blacksburg VA 24060, USA
[2] Embry Riddle Aeronautical University, Daytona Beach FL 32114, USA
{afarhana,sabihaafroz,monamp,dsn}@vt.edu,gharamik@my.erau.edu

Abstract. Diffusion models produce high-quality images, but their deployment remains expensive because generation requires many denoising steps through large transformer or U-Net backbones. Existing methods improve efficiency through post-training quantization or timestep reduction, but they optimize these two axes separately. Quantization methods usually assume a fixed precision, timestep reduction methods assume a fixed schedule, and both often rely on proxy costs such as BitOps instead of real hardware measurements. We introduce DiffPro, a post-training framework that jointly optimizes per-layer quantization precision and the denoising schedule under a unified budget defined by measured memory and latency on real hardware. DiffPro combines three key ideas: (i) a composite sensitivity metric that combines PCA dimensionality with diagonal Hessian curvature for bit allocation, (ii) Dynamic Activation Quantization (DAQ), which adapts activation scales across timesteps to handle temporal activation drift, and (iii) a drift-guided timestep selector that removes redundant steps while preserving the late refinement phase. On DiT-XL/2 (ImageNet 256×256), DiffPro achieves 50% step reduction, 7.9× compression, and 2.7× speedup at FID 5.89 (ΔFID 3.62), improving over uniform thinning by 2.5 FID at matched precision. On SDXL U-Net (COCO 2014 1024×1024), DiffPro achieves 5.6× compression with FID 26.50 (ΔFID 2.0) and 45% energy reduction, all without retraining.

Keywords: Diffusion models, quantization, timestep optimization

1 Introduction

Diffusion models are the dominant approach for high-fidelity image and video generation, surpassing GANs [12] and autoregressive methods [2,35] in quality and scalability [15, 25]. However, inference typically requires hundreds of sequential denoising steps through a large transformer or U-Net backbone [15], making deployment costly in latency, memory, and energy [3, 7, 34, 38].

Two common strategies reduce this cost. Post-training quantization (PTQ) lowers the numerical precision of weights and activations to reduce memory

and accelerate integer arithmetic [21]. Timestep reduction prunes the denoising schedule to decrease the number of forward passes [39]. Most prior work optimizes these axes independently: quantization ignores the schedule, while step reduction ignores how quantization noise varies across timesteps [36]. In addition, many methods rely on proxy metrics (e.g., FLOPs, BitOps) that can diverge from measured latency due to memory-bound kernels, dispatch overhead, and tensor-layout effects [7, 10, 40].

Diffusion Transformers (DiTs) [25] exacerbate these issues. Their activation statistics vary non-monotonically over timesteps, with early-step variance often far exceeding late-step variance. Static quantization scales calibrated on a single distribution therefore clip outliers elsewhere in the trajectory. Temporal drift compounds layer-wise heterogeneity: attention, MLP, and embedding layers differ substantially in quantization sensitivity, so a uniform bit-width allocation wastes precision on robust layers while under-protecting fragile ones. Moreover, removing steps reshapes the error landscape traversed by the quantized model, tightly coupling step reduction and precision allocation. Hardware further complicates the picture, as INT8 kernel latency depends on tensor shapes and dispatch paths rather than FLOPs alone [4, 25], making proxy cost models unreliable.

These gaps motivate DiffPro, a unified post-training framework that jointly optimizes quantization and scheduling under measured kernel latency. Our key contributions are:

- We present DiffPro, a unified pipeline that jointly optimizes timesteps and per-layer precision under measured hardware timing.
- We design a composite sensitivity metric that fuses PCA dimensionality, diagonal Hessian curvature, temporal drift, and quantization noise, improving layer-ranking correlation from 0.61 (best single signal) to 0.81 (Tab. 2).
- We introduce dynamic activation quantization (DAQ), which rescales activations per sample and per timestep to stabilize low-bit inference across the denoising trajectory.
- We propose a drift-guided timestep selector that improves FID by 2.5 points over uniform thinning at matched precision (Uni.500: 8.41→DiffPro: 5.89). The remaining 7.93-point gap over Q-DiT+Uni.500 reflects manifold-aware precision allocation.

2 Related Work

2.1 Background & Motivation

Diffusion models generate samples by iteratively denoising Gaussian noise with a timestep-conditioned Diffusion Transformer (DiT) [25] or U-Net [28]. DiTs have largely surpassed U-Nets in scalability and generation quality [4]. To characterize their activation dynamics, we profile per-layer activation standard deviations of DiT-XL/2 over 1,000 DDIM steps (Fig. 1). Attention layers exhibit the largest temporal variation, followed by MLP and embedding layers; shifts are strongest

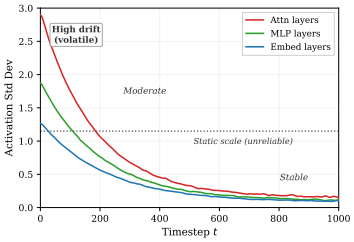


Fig. 1: Activation standard deviation across timesteps by layer type. Early volatility motivates dynamic scaling; a static (dotted) scale is unreliable.

early and stabilize during late-stage refinement. These results indicate that static, per-layer quantization scales are unreliable across the denoising trajectory.

These observations motivate three design choices in DiffPro. First, activation quantization must be dynamic (Sec. 4.3). Second, layers vary in their sensitivity to precision reduction, so DiffPro uses a composite sensitivity metric for per-layer bit allocation (Sec. 4.1). Third, timestep pruning alters the quantized model's error landscape, coupling schedule reduction with precision allocation. Moreover, INT8 kernel latency depends on tensor shapes and dispatch paths rather than FLOPs [7, 10]. Together, these factors motivate joint optimization of precision and scheduling under measured kernel timing (Sec. 4.5).

Diffusion Model Architectures Diffusion models [15] generate samples by reversing a Gaussian noising process. Diffusion Transformers (DiTs) [25] have largely replaced U-Net backbones [28] in many deployments, while latent diffusion models [27] reduce cost by operating in a compressed variational autoencoder (VAE) latent space. DiT activations exhibit non-monotonic temporal variation and channel imbalance, making them substantially harder to quantize than U-Net predecessors.

Post-Training Quantization and Timestep Scheduling Most prior work treats quantization and timestep reduction separately. PTQ4DM [30], Q-Diffusion [19], PTQD [14], PTQ4DiT [37], Q-DiT [7], and TRDQ [31] focus on post-training quantization, while DDIM [33], AutoDiffusion [17], DiffuseVAE [24], and OSS [29] reduce sampling steps under fixed precision. TMPQ-DM [34] considers both, but relies on proxy costs.

Three gaps remain. First, PTQ4DiT [37] and Q-DiT [7] use limited signals for bit allocation and do not jointly model drift and PCA-based structure. Second, Q-DiT [7] uses fixed per-layer activation scales instead of per-sample, per-timestep adaptation. Third, TRDQ [31] and AutoDiffusion [17] do not account for how quantization alters timestep importance. DiffPro addresses these gaps jointly under measured INT8 kernel timing.

Layer Sensitivity and Dynamic Activation Quantization Bit-allocation methods typically assign per-layer precision using single-signal heuristics (e.g., Hessian traces [11]), which do not capture the geometric structure of the activation

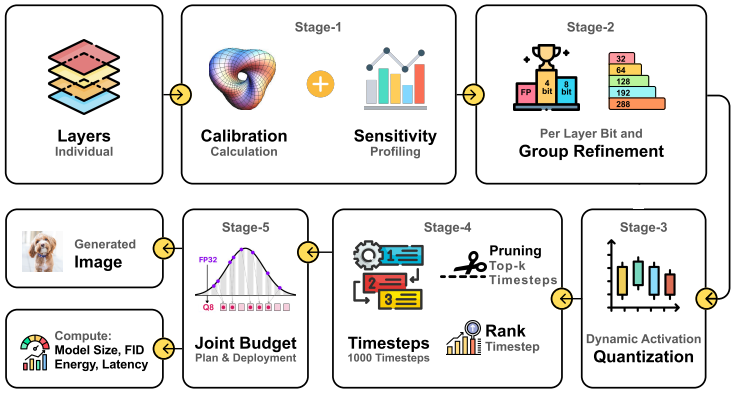


Fig. 2: DiffPro overview: the pipeline analyzes layers, derives manifold and PCA signals, ranks sensitivity to seed a bit plan, applies DAQ, prunes a 1,000-step schedule with a protected tail, and then quantizes and evaluates FID, latency, energy, and model size.

manifold [14, 37]. Static activation scales also degrade under timestep-dependent outliers, and existing dynamic methods lack student-aware calibration [7, 41]. DiffPro addresses these limitations by combining curvature energy with PCA rank for precision allocation and by using per-sample, per-timestep activation hooks calibrated against the quantized student.

Table 1 summarizes the limitations of prior work; DiffPro addresses all four gaps simultaneously.

Table 1: Comparison of Related Methods and Identified Research Gaps.

Method	Joint Opt.	HW Timing	DAQ	DiT
PTQ4DM, Q-Diffusion, PTQD [14, 19, 30]	×	×	×	×
Q-DiT, TRDQ [7, 31]	×	×	Partial	✓
AutoDiffusion, TMPQ-DM [17, 34]	×	×	×	×
DiffPro (Ours)	✓	✓	✓	✓

3 DiffPro Overview

3.1 Problem Statement

Let f_θ denote a pretrained diffusion denoiser with L linear layers and $\mathcal{T} = \{0, 1, \ldots, T-1\}$ the full denoising timestep set. A deployment plan $\Pi = (\mathbf{b}, \mathbf{g}, \mathcal{S})$ specifies: (i) per-layer bit widths $\mathbf{b} = (b_1, \ldots, b_L)$ with $b_\ell \in \{4, 8, 16\}$; (ii) per-layer group sizes $\mathbf{g} = (g_1, \ldots, g_L)$ with $g_\ell \in \{32, 64, 128, 192, 288\}$; and (iii) a retained timestep subset $\mathcal{S} \subseteq \mathcal{T}$ of size k. The bit width determines the numerical precision of each layer's weights, the group size controls the granularity of per-group quantization scales, and $\mathcal{S}$ defines which denoising steps are executed during inference.

The goal is to find Π^* that minimizes generation-quality degradation $\mathcal{D}$ (measured by FID between the quantized student $f_{\hat{\theta}}$ and the full-precision teacher f_θ), subject to hardware budgets on latency and memory (Eq. (1)).

$$\Pi^* = \arg\min_{\Pi} \ \mathcal{D}(f_{\hat{\theta}}^{\Pi}) \quad \text{s.t.} \quad C_{\text{lat}}(\Pi) \le B_{\text{lat}}, \ \ C_{\text{mem}}(\Pi) \le B_{\text{mem}} \tag{1}$$

where $C_{\text{lat}}(\Pi)$ and $C_{\text{mem}}(\Pi)$ denote the end-to-end inference latency and peak memory of plan Π, and B_{lat} and B_{mem} are the corresponding deployment budgets. All costs are grounded in measured INT8×INT8→INT32 kernel timing.

Modern DiTs operate in a compressed VAE latent space [25, 26], where precision and the denoising schedule jointly determine the quality–latency trade-off. DiffPro casts this as a post-training joint optimization problem over per-layer weight quantization, dynamic activation quantization, and timestep selection, guided by a full-precision teacher and measured hardware timing. Unlike prior methods that optimize these choices in isolation or rely on proxy costs [7, 17, 31, 34, 37], DiffPro optimizes them jointly under a unified budget. The pipeline has five stages: (1) calibration and sensitivity profiling, (2) per-layer bit and group-size refinement, (3) dynamic activation quantization, (4) drift-guided timestep pruning, and (5) budget-aware joint search, as shown in Fig. 2, formalized in Algorithms 1 and 2, and detailed in Sec. 4.

Algorithm 1 (DiffPro): End-to-End Training Free Optimization

Input: Pretrained denoiser f; compute budget B; calibration set; group sizes $G = \{32, 64, 128, 192, 288\}$.

1. *Calibrate:* collect per layer PCA statistics (95% variance); cache teacher outputs (x_t, t, y).
2. *Score:* for each layer ℓ, compute:
S_ℓ^x (curvature + PCA), S_ℓ^d (temporal drift), S_ℓ^k (sensitivity knee), S_ℓ^n (quantization noise);
fuse: $S_\ell^* = 0.4\, S_\ell^x + 0.2\, S_\ell^d + 0.25\, S_\ell^k + 0.15\, S_\ell^n$ (Eq. 3).
3. *Allocate:* tier layers into high/mid/low risk by S_ℓ^*; assign W4/W8/FP16 via ROI knapsack under B;
refine with GPTQ evolutionary search screened by $\mathcal{L}_{\text{drift}}$.
4. *DAQ:* enable per timestep quantile scaling with early/mid/late phase bins; manage outliers.
5. *Prune & Finalise:* remove low drift timesteps; protect tail $\mathcal{T}_{\text{tail}}$; apply bias/LN correction and adaptive rounding on top k fragile layers.

Output: Final deployable plan Π^*.

4 DiffPro Details

4.1 Stage 1: Calibration and Manifold-Aware Sensitivity

Algorithm 1 begins with a calibration pass (step 1) that collects per-layer principal component analysis (PCA) statistics (95% variance) and caches teacher outputs $\epsilon_\theta(x_t, t, y)$ from the full-precision model f_θ.

For each layer ℓ (step 2), we compute four signals: (i) S_ℓ^x, which fuses curvature energy with PCA sensitivity to capture geometric fragility; (ii) S_ℓ^d, which measures temporal drift across timestep phases; (iii) S_ℓ^k, which captures the sensitivity knee from bit-width and group-size sweeps; and (iv) S_ℓ^n, which models Jacobian-amplified quantization noise [22]. These signals are weakly correlated (mean $|r|$=0.31 across 113 layers; see supplementary), indicating that they provide complementary information. No single signal is sufficient: the best individual signal (S^x) yields FID 10.41, compared with 8.86 for the fused score (Tab. 2).

For each layer ℓ, we define a fragility score S_ℓ^x by combining curvature energy with PCA-based sensitivity. Specifically, $h_{\text{diag}} = \sum_i x_i^2$ denotes the activation energy, where x_i are the activations of layer ℓ. The PCA term s_{PCA} measures activation complexity using the number of principal components k_{95} required to explain 95% of the variance, the feature dimension d, and the residual spill, $\text{spill} = 1 - \text{cumvar}(k_{95})$. We compute $s_{\text{PCA}} = 0.5 \cdot \text{spill} + 0.5 \cdot \frac{k_{95}}{d}$ and fuse it with normalized curvature as $S_\ell^x = 0.5 \cdot s_{\text{PCA}} + 0.5 \cdot h_{\text{diag}}^{\text{norm}}$. We use equal weights as a neutral default; varying the split from 0.3/0.7 to 0.7/0.3 produces only small FID changes, indicating low sensitivity.

We also define a noise score S_ℓ^n that estimates quantization rounding noise from the layer step size Δ_ℓ and scales it by the layer Jacobian energy [22] to reflect downstream amplification. We further compute S_ℓ^d for temporal drift and S_ℓ^k for the sensitivity knee from bit-width and group-size sweeps. Together, these complement S_ℓ^x by capturing effects beyond geometric fragility.

The per-layer quality degradation is illustrated in Eq. (2), where $\Delta\text{FID}(\ell)$ is the predicted FID change for layer ℓ, $\mathbf{s}_\ell$ is the layer-signal vector, and $\mathbf{w}$ is the learned weight vector.

$$\Delta\text{FID}(\ell) \approx \mathbf{w}^\top \mathbf{s}_\ell, \tag{2}$$

where $\mathbf{s}_\ell = [S_\ell^x, S_\ell^d, S_\ell^k, S_\ell^n]^\top$ stacks fragility, drift, knee sensitivity, and quantization noise. We obtain ground truth by quantizing one layer at a time to W4 with all others at FP16 (113 passes). OLS regression achieves $R^2 = 0.76 \pm 0.04$ (5-fold cross-validation) and yields the composite weights in Eq. (3), which round to $0.4, 0.2, 0.25, 0.15$ and remain stable under ± 0.05 perturbations (FID < 9.62; Tab. 2).

$$S_\ell^* = 0.4\, S_\ell^x + 0.2\, S_\ell^d + 0.25\, S_\ell^k + 0.15\, S_\ell^n. \tag{3}$$

The weights refit to $(0.38, 0.22, 0.24, 0.16)$ on SDXL and $(0.39, 0.21, 0.25, 0.15)$ on FLUX, remaining within 0.3 FID of the original setting. Under ± 0.05 perturbation the worst FID shift is 0.76, small relative to the 1.55 FID gain of the fused score over the best single signal (Tab. 2).

Fig. 3a shows the OLS fit (R^2=0.76) against measured ΔFID; Figs. 3b to 3g show PCA variance curves, activation distributions, curvature vs. PCA sensitivity, score CDF, activation envelopes, and top-10 layer rankings, respectively.

Table 2: Sensitivity Weight Ablation (Bit Plan Only, Without DAQ or Timestep Pruning). Full DiffPro Achieves FID 5.89 (Tab. 7, DiffPro Row)

Config. (w_x, w_d, w_k, w_n)	Corr.	FID	Config. (w_x, w_d, w_k, w_n)	Corr.	FID
S^x only (1,0,0,0)	0.61	10.41	Equal (0.25 each)	0.72	9.83
S^d only (0,1,0,0)	0.55	11.22	PCA heavy (0.6,0.1,0.2,0.1)	0.75	9.54
S^k only (0,0,1,0)	0.58	10.84	Drift heavy (0.2,0.5,0.2,0.1)	0.70	9.97
S^n only (0,0,0,1)	0.49	13.07	**Ours (0.4,0.2,0.25,0.15)**	**0.81**	**8.86**
Pert. (0.35,0.25,0.25,0.15)	0.79	9.12	Pert. (0.40,0.20,0.20,0.20)	0.79	9.18
Pert. (0.45,0.15,0.25,0.15)	0.78	9.24	Pert. (0.40,0.20,0.30,0.10)	0.80	9.02
Pert. (0.35,0.25,0.20,0.20)	0.77	9.41	Pert. (0.45,0.15,0.30,0.10)	0.79	9.15
S^x (0.7 PCA, 0.3 curv.)	0.78	9.28	S^x (0.3 PCA, 0.7 curv.)	0.76	9.45

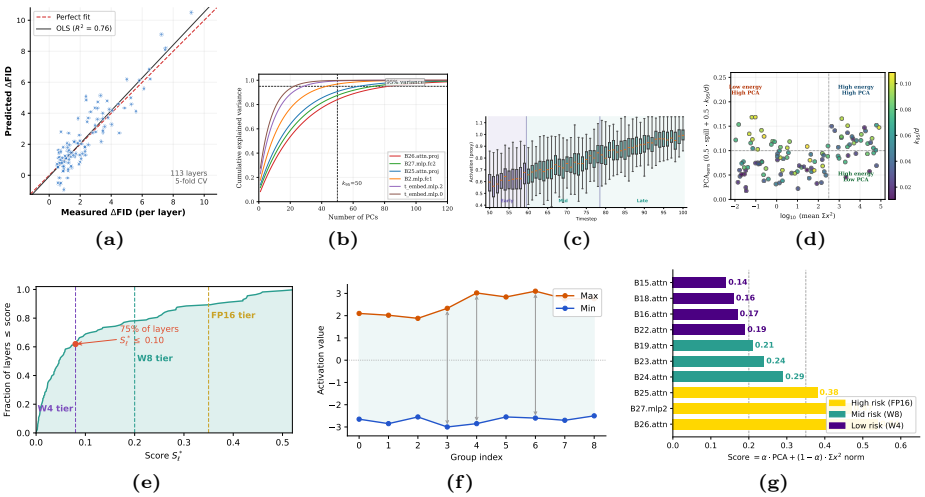
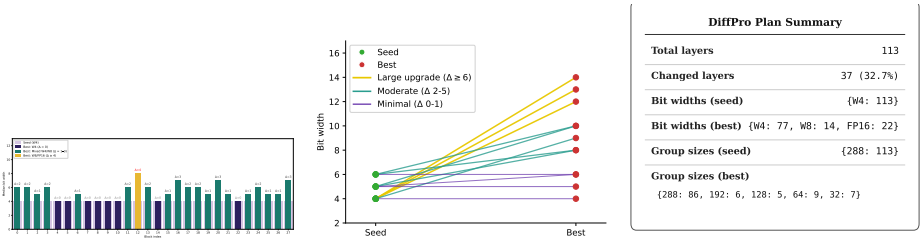


Fig. 3: Sensitivity analysis (113 layers): (a) OLS fit (R^2=0.76), (b) PCA variance, (c) activation distribution, (d) curvature vs. PCA, (e) score CDF, (f) activation envelope, (g) top-10 layer ranking.

4.2 Stage 2: Per Layer Bit and Group Size Refinement

Step 3 of Algorithm 1 converts the sensitivity ranking into a concrete bit plan. Layers are tiered by S_ℓ^* into high-, mid-, and low-risk groups, with the highest-risk fraction frozen at full precision. A return-on-investment knapsack allocates W4/W8/FP16 under budget B, prioritizing layers where each additional bit yields the largest quality gain per unit cost, guided by each layer's knee point (b_ℓ^*, g_ℓ^*). The seed plan assigns W4 with coarse group sizes (192–288) to low-risk layers, W8 with default groups to mid-risk layers, and FP16 with small groups (32–64) to high-risk layers.

We refine this seed via evolutionary search using GPTQ [11]. GPTQ approximates the per-layer Hessian as $H \approx X^\top X$, solves via Cholesky decomposition [32], and quantizes weights group-wise to integer tensors with per-group scale and zero-point. Teacher outputs are cached from a single calibration pass; each gener-



(a) **(b)** **(c)**

Fig. 4: Step 2 analysis: (a) Median bit width per block, (b) Bit width changes from seed to best plan, (c) Summary of refinement.

ation mutates bit widths and group sizes while keeping frozen high-risk layers unchanged. Candidates are screened using teacher–student drift (Eq. (4)):

$$\mathcal{L}_{\text{drift}} = \mathbb{E}_{(x_t,t,y)} \left\| \epsilon_{\text{student}}(x_t, t, y) - \epsilon_{\text{teacher}}(x_t, t, y) \right\|_2^2 \tag{4}$$

Successive halving evaluates candidates on a few batches first and then re-evaluates top performers with the full setting. We skip duplicates via hashing and retain low-drift elites across generations until convergence.

Fig. 4a shows early blocks remain at W4 while late attention blocks upgrade to W8 or FP16, confirming sensitivity concentrates in deeper layers. Fig. 4b traces bit transitions from seed to best plan, with large upgrades targeting attention projections in blocks 19–27. Fig. 4c summarizes the outcome: 37 of 113 layers changed, yielding 77 W4, 14 W8, and 22 FP16.

4.3 Stage 3: Dynamic Activation Quantization

Step 4 of Algorithm 1 applies DAQ using quantile clipping and timestep phase bins. Because activation ranges in DiTs vary across the denoising trajectory (Fig. 1), fixed per layer scales are insufficient. For layer ℓ at timestep t, activations x_{btc} (sample b, channel c) are split into channel groups of size g, indexed by group k with channel set $\mathcal{C}_k$. We clip using the 99.9th percentile threshold $P_{99.9}$ (selected by sweeping 99.0–99.99 on the calibration set, see supplementary):

$$\hat{x}_{btc} = \text{clamp}(x_{btc},\, -P_{99.9},\, P_{99.9}), \tag{5}$$

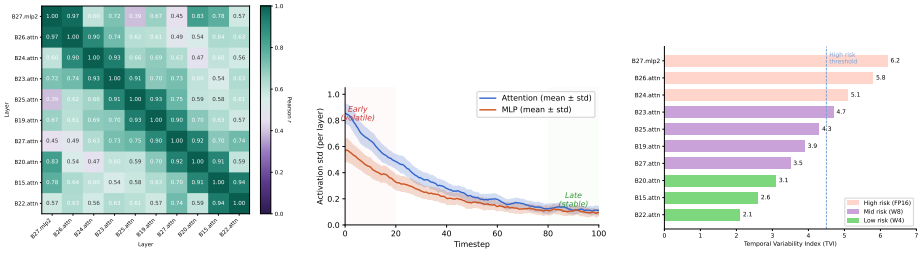
and compute a symmetric INT8 scale per sample, timestep, and group as

$$\tau_{btk} = \frac{\max_{c \in \mathcal{C}_k} |\hat{x}_{btc}|}{2^{b_\ell - 1} - 1}, \tag{6}$$

where b_ℓ is the activation bit width for layer ℓ and τ_{btk} is the group scale used to quantize activations. Lightweight pre-hooks compute τ_{btk} before each grouped linear. GPTQ packed weights [11] and grouped linears execute as INT8×INT8→INT32 GEMMs. DAQ covers Q, K, V, attention output projection, and both MLP layers; LayerNorm and residual paths remain in higher

Table 3: DAQ Runtime Overhead Per Inference Step.

Component	Time (ms/step)	% of Total
Percentile computation	0.12	1.8%
Scale calculation	0.04	0.6%
Requantization	0.08	1.2%
Total DAQ overhead	**0.24**	**3.6%**



(a) **(b)** **(c)**

caption DAQ analysis: (a) layer correlation, (b) attention vs. MLP volatility, (c) TVI ranking.

precision. Unlike Q-DiT, DiffPro computes scales per sample, per timestep, and per group, calibrated against the quantized student.

Fig. 5a reveals correlated temporal dynamics among late attention blocks, Fig. 5b shows attention layers are more volatile than MLP layers in early timesteps, and Fig. 5c confirms late block attention projections rank highest in temporal variability, validating DAQ's per-timestep scaling.

4.4 Stage 4: Budgeted Timestep Selection (Pruning)

Step 5 prunes low-drift timesteps while protecting a tail set $\mathcal{T}_{\text{tail}}$ (ρ=0.2, i.e., the last 20% of steps, which becomes 40% of the retained schedule under 50% pruning). For each candidate t, drift $\delta(t)$ measures student–teacher output divergence on calibration samples:

$$\delta(t) = \mathbb{E}_{(x_t,y)} \, \|f_{\hat{\theta}}(x_t, t, y) - f_{\theta}(x_t, t, y)\|_2^2. \tag{7}$$

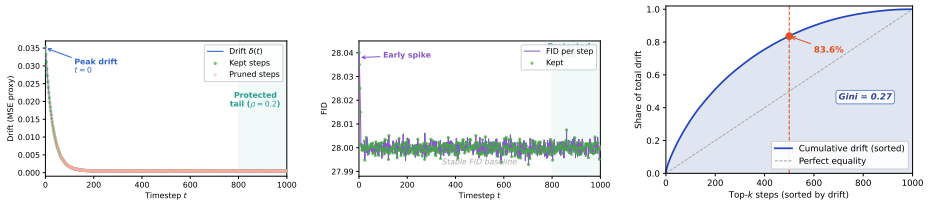
We protect the refinement phase by defining the tail set

$$\mathcal{T}_{\text{tail}} = \{t \in \mathcal{T} : t/T \geq 1 - \rho\}, \tag{8}$$

where $\mathcal{T}$ is the full timestep set. The final retained schedule keeps all tail steps and fills the remaining budget with the highest drift steps:

$$\mathcal{S} = \mathcal{T}_{\text{tail}} \cup \text{TopK}(\mathcal{T} \setminus \mathcal{T}_{\text{tail}}, \, \delta(\cdot), \, k - |\mathcal{T}_{\text{tail}}|), \quad |\mathcal{S}| = k, \tag{9}$$

where k is the target, and TopK selects the highest-drift steps, which are most fragile under quantization; low-drift steps are pruned with minimal quality loss.



(a) **(b)** **(c)**

Fig. 6: Timestep pruning: (a) drift, (b) per-step FID, (c) Lorenz curve (k=500).

Fig. 6a shows drift peaks in early steps with low values near late steps where the tail is protected, Fig. 6b confirms pruned steps have negligible FID impact, and Fig. 6c shows the top 500 steps capture 83.6% of total drift, justifying 50% pruning.

4.5 Stage 5: Joint, Budget-Aware Plan Search

Algorithm 2 merges the bit plan (Stage 2), DAQ policy (Stage 3), and pruned schedule (Stage 4) into a single deployable configuration. The tail set $\mathcal{T}_{\text{tail}}$ is always retained, and all layers start at maximum precision. Here, T is the original number of timesteps, ρ is the protected-tail fraction, and pruning to B steps yields an effective tail fraction $\rho T/B$. The algorithm maintains a drift-ranked queue of prunable steps $\delta(\cdot)$ and a max-heap of bit-reduction moves ranked by savings per estimated quality loss.

Each iteration chooses between pruning the current timestep $t^\star$ and applying the best bit move by comparing step efficiency $S_{\text{step}} = c_{\text{step}}(t^\star)/(\varepsilon + D_e(t^\star))$ against bit efficiency S_{bit}, where $c_{\text{step}}(t^\star)$ is the latency saved, $D_e(t^\star)$ is the estimated FID increase, and $\varepsilon = 10^{-8}$ avoids division by zero. The loop stops once the latency and memory budgets B_{lat} and B_{mem} are met.

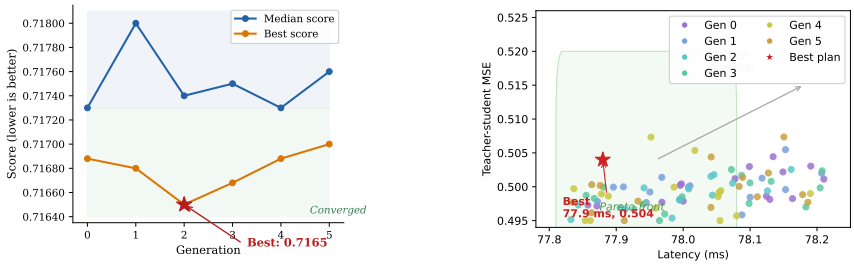
Candidates are scored by

$$\text{score} = \text{MSE} + \lambda \cdot \text{latency\_penalty} + \mu \cdot \text{bitops\_penalty}, \quad \lambda = \mu = 0.5, \quad (10)$$

where MSE is the student–teacher error and the penalties enforce the budgets; $\lambda = \mu = 0.5$ gives equal weight to latency and BitOps violations. Latency uses measured kernel timing from synchronized CUDA events. Step 5 (Algorithm 1) finalizes the plan with lightweight bias and LayerNorm correction and adaptive rounding. The supplementary shows that this greedy search is near-optimal at 3.2× lower search cost than simulated annealing or random search.

Fig. 7a confirms convergence within 1–2 generations, with the best plan at 77.9 ms and MSE ≈ 0.504.

The final plan Π^* wraps each linear layer with quantized kernels. During reverse denoising over $t \in \mathcal{K}^*$, DAQ selects activation bit widths from $\mathcal{P}_{\text{DAQ}}$, the model runs with quantized weights and activations, and the VAE decodes the output. Planning costs $O(T \log T + L \log L)$ due to drift sorting and heap-based layer selection, while inference scales linearly with $|\mathcal{K}^*|$.



(a) (b)

Fig. 7: Step 5 joint search: (a) Score evolution across generations, (b) MSE vs. latency Pareto front.

Algorithm 2 (Joint Planning): Budget-Aware Bit & Step Co-Optimization
Require: Per-step drift $\delta(t)$, per-step latency $c_{\text{step}}(t)$; per-layer bit reduction costs; budgets B_{lat}, B_{mem}.
1. Protect tail $\mathcal{T}_{\text{tail}}$; initialize all layers at max precision; keep all steps.
2. Build sorted prunable step queue R (by $\delta(\cdot)$ ascending); build max heap H_{bit} (by savings/quality-loss).
3. while $C_{\text{lat}} > B_{\text{lat}}$ **or** $C_{\text{mem}} > B_{\text{mem}}$: compute S_{step}, S_{bit}; apply higher-efficiency move; update costs.
Output: Optimized bit plan Π^* and kept steps $\mathcal{S}^*$.

5 Implementation

We evaluate class-conditioned DiT-XL/2 [25] on ImageNet [8] (256×256) and text-conditioned SDXL U-Net [27] on COCO 2014 [20] (1024×1024; 30K text prompts). DiT uses 1,000 DDIM steps, while SDXL uses 100. DiffPro retains 500 and 50 steps, respectively, with $\rho = 0.2$ yielding a 40% effective tail in both cases to protect late-stage refinement, where quantization noise is most perceptible. We report FID on 50K samples for DiT-XL/2 (cfg = 1.5) [9] and on 30K samples for SDXL (cfg = 3.0) [27]. All methods within each architecture group in Table 4 use identical protocols; batch-size 4 latency and per-seed FID breakdowns are provided in the supplementary material.

Experiments run on NVIDIA A30 24 GB GPUs (CUDA 12.1.1, PyTorch 2.1.2), using INT8×INT8→INT32 GEMMs via cuBLASLt (and CUTLASS for selected grouped kernels). Weights are GPTQ-packed [11], and DAQ applies per-sample, per-timestep activation scales calibrated on 512 examples. Latency is measured end-to-end with synchronized CUDA events and averaged over 100 runs at batch size 1. Energy is measured with NVML [23] (idle power subtracted), averaged over 100 runs.

Planning takes ∼53 minutes as a one-time cost (108 minutes on FLUX). During inference, DAQ adds 0.24 ms per step (3.6% overhead). Search cost scales as $O(T \log T + L \log L)$ and is amortized after approximately 1,200 generated images; it does not affect inference latency because the final bit allocation and timestep schedule are fixed before deployment. Further planning details are in the supplementary material.

6 Evaluation

6.1 Comparison with Baselines

Tab. 4 compares DiffPro with existing PTQ and scheduling methods on DiT-XL/2 (ImageNet 256×256) and SDXL U-Net (COCO 2014 1024×1024) under identical A30/A100 conditions.

On DiT-XL/2, DiffPro achieves FID 5.89 with 7.9× compression and 2.7× speedup, outperforming Q-DiT (6.40) and PTQ4DiT (7.75) at lower latency. The 7.93-point gap over Q-DiT+Uni.500 (13.82) reflects the combined benefit of drift-guided pruning and manifold-aware precision allocation. TQ-DiT achieves FID 4.91 but at 2× more steps and higher latency (3.60 vs. 2.55 s). Our W8A8 variant (FID 5.12) nearly matches it at 25% lower latency.

On SDXL, architecture-specific methods (Q-DiT, TQ-DiT) do not directly transfer because SDXL uses a heterogeneous convolution–attention U-Net. Using GPTQ W8/A8 and W4/A8 as baselines, DiffPro (FID 26.50) outperforms both (28.82 and 29.47) with 5.6× compression (12.9 GB→2.3 GB). Extended baselines, including TMPQ-DM (27.91) and uniform mixed W4A8 (28.15), confirm this advantage.

Tab. 5 reports perceptual metrics, with LPIPS computed relative to full precision. DiffPro incurs less than a 0.005 CLIP-score drop and achieves LPIPS below 0.12 on both architectures, indicating that compression preserves text–image alignment and perceptual fidelity. Fig. 8a shows the per-layer bit assignment, and Fig. 8b shows the FID–latency Pareto front.

6.2 Generalization to Modern Architectures

Tab. 6 evaluates DiffPro on FLUX.1-dev [13] and PixArt-α [6], comparing against SVDQuant [18] and ViDiT-Q [41] under the same A30 GPU and latency protocol as our main experiments, evaluated on COCO.

On FLUX.1-dev, DiffPro achieves FID 20.5 and IR 0.943, outperforming SVDQuant (FID 20.7, IR 0.935) and ViDiT-Q (FID 21.2, IR 0.921) while remaining close to full precision (FID 20.3, IR 0.953). This result highlights that quantization-only methods reduce model cost without accounting for which denoising steps become most sensitive after compression; joint optimization of timesteps and precision captures this interaction.

On PixArt-α, DiffPro achieves the best FID (73.54), IR (0.901), and LPIPS (0.323) among the compressed methods. ViDiT-Q W4/A8 degrades substantially on IR (0.573), and SVDQuant incurs a large FID increase (79.39) despite competitive LPIPS. DiffPro's FID and IR remain within 0.20 and 0.010 of full precision, respectively, demonstrating generalization beyond DiT-XL/2 to architectures with different design choices.

Table 4: Baseline Comparison On DiT-XL/2 (FID-50K, CFG=1.5 [9]) And SDXL (FID-30K, CFG=3.0 [27]). All Latencies Are Measured On A30/A100 GPUs With Batch Size 1.

Arch.	Method	W/A	FID↓	Size↓	A30	A100
DiT-XL/2	Full Precision	16/16	2.27	2575	6.88	3.28
	PTQ4DM [30]	8/8	20.91	698	4.24	2.00
	Q-Diffusion [19]	8/8	14.78	674	6.29	2.99
	PTQD [14]	4/8	26.64	335	5.44	2.59
	GPTQ [11]	4/8	15.76	358	4.10	1.95
	PTQ4DiT [37]	4/8	7.75	339	3.85	1.83
	Q-DiT [7]	4/8	6.40	348	3.21	1.53
	TQ-DiT [16]	8/8	4.91	392	3.60	1.71
	Q-DiT+Uni.500 [7]	4/8	13.82	348	1.85	0.88
	DiffPro (W8A8)	8/8	5.12	392	2.68	1.27
	DiffPro	Mixed	**5.89**	**327**	**2.55**	**1.21**
SDXL	Full Precision	16/16	24.50	12877	9.84	4.68
	GPTQ [11]	8/8	28.82	6439	7.21	3.43
	GPTQ [11]	4/8	29.47	3568	6.54	3.11
	Uniform Mixed	4/8	28.15	3210	5.89	2.80
	TMPQ-DM† [34]	Mixed	27.91	2854	3.12	1.48
	DiffPro	Mixed	**26.50**	2317	2.35	1.12

Size in MB; A30/A100 latency in s/img.
† TMPQ-DM covers latent diffusion models only; DiT results and latency comparisons are not available from the original paper.

Table 5: Perceptual Quality Metrics on DiT-XL/2 (50K Images) and SDXL (30K Images).

Arch.	Method	FID↓	CLIP↑	LPIPS↓
DiT-XL/2	Full Precision	2.27	0.286	–
	DiffPro (Mixed)	5.89	0.281	0.11
SDXL	Full Precision	24.50	0.314	–
	DiffPro (Mixed)	26.50	0.312	0.08

Table 6: Generalization to Modern Architectures on COCO. IR denotes image reward. Bold marks the best compressed method.

Model	Method	W/A	FID↓	IR↑	LPIPS↓
FLUX.1-dev	Full Precision	16/16	20.3	0.953	–
	SVDQuant [18]	4/4	20.7	0.935	0.223
	ViDiT-Q [41]	8/8	21.2	0.921	0.089
	DiffPro	Mixed	**20.5**	**0.943**	**0.062**
PixArt-α	Full Precision	16/16	73.34	0.911	–
	Q-DiT [7]	8/8	73.60	0.854	0.723
	SVDQuant [18]	4/4	79.39	0.878	0.369
	ViDiT-Q [41]	8/8	75.61	0.897	0.569
	ViDiT-Q [41]	4/8	74.33	0.573	0.611
	DiffPro	Mixed	**73.54**	**0.901**	**0.323**

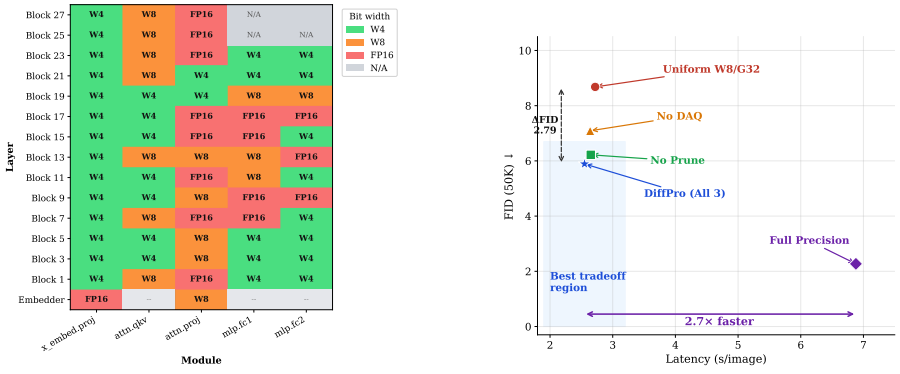


Fig. 8: (a) Final bit-width assignment across layers. (b) FID-latency tradeoff for ablation variants.

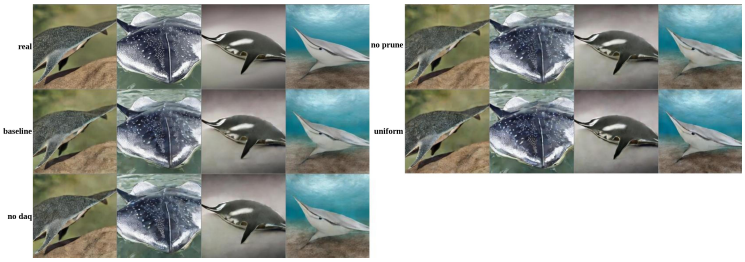


Fig. 9: Generated images across ablation variants. Left: full precision reference, DiffPro (all three components), No DAQ. Right: No Prune, Uniform W8 quantization.

6.3 Ablation Study

We ablate DiffPro on DiT-XL/2 and SDXL (Tab. 7) using five variants: (i) Uniform W8/G32, (ii) No DAQ, (iii) No Prune, (iv) Uni.50%, which retains DiffPro's bit allocation and DAQ but replaces drift-guided pruning with uniform 50% step thinning to the same number of retained steps, and (v) Sequential, which optimizes bit allocation, DAQ, and timestep pruning independently without a final joint search.

All ablated variants worsen FID relative to DiffPro (5.89), as shown in Tab. 7. The Uni.50% gap (8.41 vs. 5.89) isolates drift-guided pruning; the Sequential gap (6.74 vs. 5.89) confirms that joint optimization captures cross-component dependencies that stage-wise tuning misses.

The same trend holds on SDXL: Sequential gives FID 27.38 versus 26.50 for DiffPro, both at 2.35 s latency, with 5.6× compression and 45% energy reduction relative to full precision. As reflected in the table, pruning affects latency and energy but not model size.

Pruning ratio sweeps (25%, 50%, 75%) in the supplementary (Appendix 5) confirm that 50% achieves the best quality–latency tradeoff on both architectures; 75% degrades FID as high-drift steps are removed.

Table 7: Ablation Study On DiT-XL/2 (ImageNet 256 × 256) And SDXL U-Net (COCO 2014 1024 × 1024). Uni.50% Uses DiffPro Bits+DAQ With Uniform 50% Step Thinning. No Prune, Uni.50%, And DiffPro Share The Same Model Size.

Arch.	Method	FID↓	Lat. (s)	Energy (J)	Size (MB)↓
DiT-XL/2	Full Precision	2.27	6.88	660.94	2575.42
	Uniform W8/G32	8.68	2.72	389.61	429.95
	No DAQ	7.09	2.64	359.64	327.24
	No Prune	6.22	2.65	361.51	327.24
	Uni.50% (DiffPro bits+DAQ)	8.41	2.55	358.72	327.24
	Sequential (bits→DAQ→prune)	6.74	2.55	361.28	327.24
	DiffPro	**5.89**	**2.55**	**360.05**	**327.24**
SDXL	Full Precision	24.50	9.84	675.92	12877.1
	Uniform W8/G32	29.41	3.94	378.29	2568.12
	No DAQ	28.80	3.58	368.18	2317.87
	No Prune	28.50	3.62	370.52	2317.87
	Uni.50% (DiffPro bits+DAQ)	27.85	2.95	367.41	2317.87
	Sequential	27.38	2.35	370.15	2317.87
	DiffPro	**26.50**	**2.35**	**369.24**	**2317.87**

7 Conclusion

We present DiffPro, a post-training framework that jointly optimizes per-layer precision, dynamic activation quantization, and timestep pruning under measured hardware timing. On DiT-XL/2, DiffPro achieves a 2.7× speedup and 45% energy reduction with FID 5.89; on SDXL, it achieves FID 26.50 with 5.6× compression. The gain from joint optimization increases from 0.85 to 1.4 FID under tighter budgets, indicating that component interactions become more critical as constraints tighten. Future work includes stronger per-block quantization, support for video diffusion, and lightweight fine-tuning to close the remaining FID gap.

Acknowledgements

This work was supported in part by NSF Award No. 2315851 (SWEET), a Sony Faculty Research Award (SCALE-R project), and the Virginia Tech ICTAS and IAC Institutes. Computational resources were provided by Advanced Research Computing at Virginia Tech [1] and the Chameleon testbed [5] supported by the National Science Foundation.

References

1. Advanced Research Computing, Virginia Tech: Advanced research computing. `https://arc.vt.edu/` (2024), accessed: 2026-02-28

2. Afroz, S., Khan, R.I.S., Albahar, H., Han, J., Butt, A.R.: 10Cache: Heterogeneous resource-aware tensor caching and migration for llm training. In: Proceedings of the 2025 ACM Symposium on Cloud Computing. pp. 320–333 (2025)
3. Afroz, S., Ramanan, B., Khan, M., Butt, A.R.: Treecnn and nilmtk unite: Illuminating energy efficiency in real-world scenarios. In: 2024 IEEE International Conference on Big Data (BigData). pp. 6884–6893 (2024)
4. Bao, F., Nie, S., Xue, K., Li, C., Pu, S., Wang, Y., Yue, G., Cao, Y., Su, H., Zhu, J.: One transformer fits all distributions in multi-modal diffusion at scale. In: ICML. pp. 1692–1717 (2023)
5. Chameleon Cloud: Chameleon: A large-scale reconfigurable experimental environment for cloud research. `https://chameleoncloud.org/` (2024), accessed: 2026-02-28
6. Chen, J., Yu, J., Ge, C., Yao, L., Xie, E., Wang, Z., Kwok, J., Luo, P., Lu, H., Li, Z.: Pixart-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. In: International Conference on Learning Representations (2024)
7. Chen, L., Meng, Y., Tang, C., Ma, X., Jiang, J., Wang, X., Wang, Z., Zhu, W.: Q-DiT: Accurate post-training quantization for diffusion transformers. In: CVPR. pp. 28306–28315 (2025)
8. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A large-scale hierarchical image database. In: CVPR. pp. 248–255 (2009)
9. Dhariwal, P., Nichol, A.: Diffusion models beat GANs on image synthesis. In: NeurIPS. vol. 34, pp. 8780–8794 (2021)
10. Ding, N., Han, J., Tian, Y., Xu, C., Han, K., Tang, Y.: Post-training quantization for diffusion transformer via hierarchical timestep grouping. arXiv preprint arXiv:2503.06930 (2025)
11. Frantar, E., Ashkboos, S., Hoefler, T., Alistarh, D.: GPTQ: Accurate post-training quantization for generative pre-trained transformers. arXiv preprint arXiv:2210.17323 (2022)
12. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. Communications of the ACM **63**(11), 139–144 (2020)
13. Greenberg, O.: Demystifying flux architecture. arXiv preprint arXiv:2507.09595 (2025)
14. He, Y., Liu, L., Liu, J., Wu, W., Zhou, H., Zhuang, B.: PTQD: Accurate post-training quantization for diffusion models. arXiv preprint arXiv:2305.10657 (2023)
15. Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. In: NeurIPS. vol. 33, pp. 6840–6851 (2020)
16. Hwang, Y., Lee, H., Kang, J.: TQ-DiT: Efficient time-aware quantization for diffusion transformers. arXiv preprint arXiv:2502.04056 (2025)
17. Li, L., Li, H., Zheng, X., Wu, J., Xiao, X., Wang, R., Zheng, M., Pan, X., Chao, F., Ji, R.: AutoDiffusion: Training-free optimization of time steps and architectures for automated diffusion model acceleration. In: ICCV. pp. 7105–7114 (2023)
18. 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)
19. Li, X., Liu, Y., Lian, L., Yang, H., Dong, Z., Kang, D., Zhang, S., Keutzer, K.: Q-diffusion: Quantizing diffusion models. In: ICCV. pp. 17535–17545 (2023)
20. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: Common objects in context. In: ECCV. pp. 740–755 (2014)

21. Liu, Z., Wang, Y., Han, K., Zhang, W., Ma, S., Gao, W.: Post-training quantization for vision transformer. Advances in Neural Information Processing Systems **34**, 28092–28103 (2021)
22. Nagel, M., Amjad, R.A., Van Baalen, M., Louizos, C., Blankevoort, T.: A white paper on neural network quantization. arXiv preprint arXiv:2106.08295 (2021)
23. NVIDIA Corporation: NVIDIA Management Library (NVML) (2024), `https://developer.nvidia.com/management-library-nvml`, accessed: 2026-02-28
24. Pandey, K., Mukherjee, A., Rai, P., Kumar, A.: DiffuseVAE: Efficient, controllable and high-fidelity generation from low-dimensional latents. arXiv preprint arXiv:2201.00308 (2022)
25. Peebles, W., Xie, S.: Scalable diffusion models with transformers. In: ICCV. pp. 4195–4205 (2023)
26. Pinheiro Cinelli, L., Araujo Marins, M., Barros da Silva, E.A., Lima Netto, S.: Variational autoencoder. In: Variational Methods for Machine Learning with Applications to Deep Networks, pp. 111–149. Springer (2021)
27. 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)
28. Ronneberger, O., Fischer, P., Brox, T.: U-Net: Convolutional networks for biomedical image segmentation. In: MICCAI. pp. 234–241 (2015)
29. Sabour, A., Fidler, S., Kreis, K.: Align your steps: Optimizing sampling schedules in diffusion models. arXiv preprint arXiv:2404.14507 (2024)
30. Shang, Y., Yuan, Z., Xie, B., Wu, B., Yan, Y.: Post-training quantization on diffusion models. In: CVPR. pp. 1972–1981 (2023)
31. Shao, Y., Lin, D., Zeng, F., Yan, M., Zhang, M., Chen, S., Fan, Y., Yan, Z., Wang, H., Guo, J., et al.: TRDQ: Time-rotation diffusion quantization. arXiv preprint arXiv:2503.06564 (2025)
32. Smith, S.P.: Differentiation of the Cholesky algorithm. J. Computational and Graphical Statistics **4**(2), 134–147 (1995)
33. Song, J., Meng, C., Ermon, S.: Denoising diffusion implicit models. arXiv preprint arXiv:2010.02502 (2020)
34. Sun, H., Tang, C., Wang, Z., Meng, Y., Jiang, J., Ma, X., Zhu, W.: TMPQ-DM: Joint timestep reduction and quantization precision selection for efficient diffusion models. CoRR (2024)
35. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, Llion, e.a.: Attention is all you need. In: Advances in neural information processing systems (2017)
36. Wang, C., Wang, Z., Xu, X., Tang, Y., Zhou, J., Lu, J.: Towards accurate post-training quantization for diffusion models. In: CVPR. pp. 16026–16035 (2024)
37. Wu, J., Wang, H., Shang, Y., Shah, M., Yan, Y.: PTQ4DiT: Post-training quantization for diffusion transformers. In: NeurIPS. vol. 37, pp. 62732–62755 (2024)
38. Ye, J., Wang, Z., Jiang, L.: PQD: Post-training quantization for efficient diffusion models. In: WACV. pp. 150–156 (2025)
39. Yue, Z., Wang, J., Sun, Q., Ji, L., Chang, E.I., Zhang, H., et al.: Exploring diffusion time-steps for unsupervised representation learning. arXiv preprint arXiv:2401.11430 (2024)
40. Zhang, L., Han, S., Wei, J., Zheng, N., Cao, T., Yang, Y., Liu, Y.: nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices. In: MobiSys. pp. 81–93 (2021)
41. Zhao, T., Fang, T., Huang, H., Liu, E., Wan, R., Soedarmadji, W., Li, S., Lin, Z., Dai, G., Yan, S., et al.: ViDiT-Q: Efficient and accurate quantization of diffusion transformers for image and video generation. arXiv preprint arXiv:2406.02540 (2024)