EFFICIENT GENERATIVE IMAGE COMPRESSION WITH QUANTIZED LATENTS REFINEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative image compression (GIC) delivers strong perceptual quality at low bitrates. However, GIC methods with strong perceptual quality often rely on pretrained diffusion models with high latency, while recent lightweight GIC methods reduce latency but deliver lower perceptual quality. We observe that, for lightweight GIC methods at low bitrates, coarse deterministic quantization can collapse nearby encoder latents corresponding to different fine details to the same point, making these details difficult to recover. Motivated by this, we propose GR-GIC, an efficient GIC that delivers competitive reconstruction performance with fast inference. For better performance, we introduce Guided Refinement (GR), which combines latent refinement guided by entropy context with adaptive perturbations for training regularization, jointly improving reconstruction at low bitrates. For fast inference, we introduce reorganized projected residual vector quantization (P-RVQ), which equivalently reorganizes residual updates into low-dimensional lookups to expose greater parallelism and reduce sequential computation. Experiments on Kodak show that for performance, compared with OneDC, GR-GIC achieves BD-rate savings of 23.19%, 37.71%, 22.30%, and 22.14% for LPIPS, DISTS, PSNR, and FID, respectively. For fast inference, the highly optimized implementation of GR-GIC encodes and decodes in 4.60 and 5.98 ms on an NVIDIA A100. Specifically, the reorganized P-RVQ implementation achieves a $3.14\times$ average speedup over the original implementation under the same FP32 setting. On an NVIDIA Jetson Orin NX, GR-GIC achieves latency comparable to lightweight GIC methods.

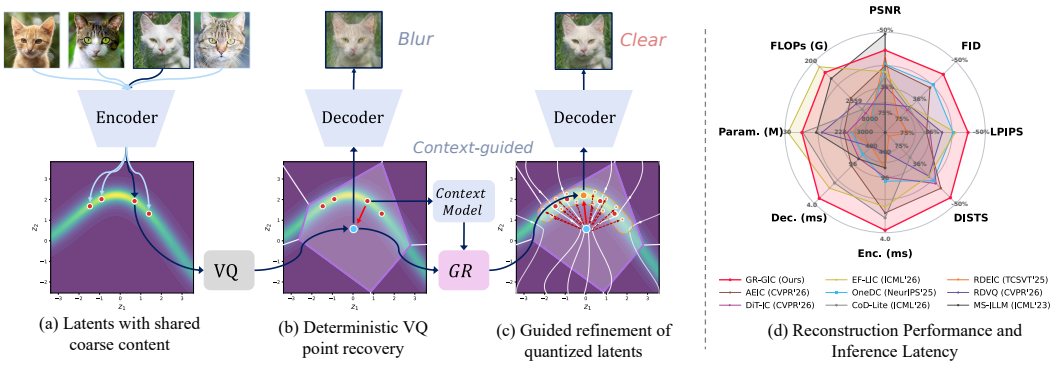
1 INTRODUCTION

Generative image compression (GIC) provides strong perceptual reconstruction at low bitrates. This is challenging for traditional codecs and learned image codecs optimized primarily for pixel fidelity, which tend to smooth textures and weaken fine details as the bitrate decreases (Wallace, 1992; Joint Video Experts Team (JVET), 2025; Ballé et al., 2018; Li et al., 2025b). GIC addresses this limitation with adversarial generators (Mentzer et al., 2020; Muckley et al., 2023), discrete generative representations (Qi et al., 2025; Xue et al., 2025b), or diffusion priors (Careil et al., 2024; Zhang et al., 2025; Xue et al., 2025a), whose learned priors can synthesize plausible details that are not fully specified by the compressed representation.

However, high-performing GIC methods often rely on pretrained diffusion models, whose large parameter counts and FLOPs hinder practical deployment (Guo et al., 2025; Zhang et al., 2025; Xue et al., 2025a; Shi et al., 2026). To address this issue, recent work has explored lightweight GIC methods with fewer parameters and faster inference (Cao et al., 2026b;a; Jia et al., 2026), but their reconstruction performance still leaves room for improvement compared with diffusion-based methods. This leaves a trade-off between reconstruction performance and fast inference.

We observe that, in lightweight GIC methods, images with similar coarse content but different fine details can be mapped to nearby points in the continuous latent space. At low bitrates, coarse deterministic quantization can collapse these distinct latents to the same quantized point, making their fine detail differences unavailable from the quantized representation, as illustrated in Figures 1a and 1b. By contrast, diffusion based methods introduce diversity through Gaussian noise and use strong generative priors to synthesize plausible details, providing a natural mechanism to compensate for this uncertainty.

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) Latents with shared coarse content

(b) Deterministic VQ point recovery

(c) Guided refinement of quantized latents

(d) Reconstruction Performance and Inference Latency

Figure 1: Motivation and overview of GR-GIC. (a) Images with similar coarse content but different fine details are mapped to nearby points in the continuous latent space. (b) At low bitrates, coarse deterministic quantization can collapse these distinct latents to the same quantized point, leaving their fine detail differences unspecified. (c) GR uses entropy context to define a Gaussian refinement distribution around the recovered latent and introduces adaptive perturbations during training. (d) GR-GIC combines competitive reconstruction performance with fast inference.

Motivated by this observation, we propose GR-GIC, an efficient GIC that combines competitive reconstruction performance with fast inference, as shown in Figure 1d. For better performance, we introduce Guided Refinement (GR), which uses entropy context to parameterize a Gaussian refinement distribution around the recovered latent. Its mean shifts the recovered latent toward a refined center, while its variance controls adaptive perturbations for training regularization, jointly improving reconstruction at low bitrates. For fast inference, we target projected residual vector quantization (P-RVQ), which is a major encoder latency bottleneck in EF-LIC (Cao et al., 2026a). We derive an algebraically equivalent reorganization that replaces serial high dimensional residual updates with low dimensional lookups, exposing greater parallelism and reducing sequential computation.

Experiments on Kodak show that (i) for performance, compared with OneDC (Xue et al., 2025a), GR-GIC achieves BD-rate (Bjøntegaard, 2001) savings of 23.19%, 37.71%, 22.30%, and 22.14% for LPIPS (Zhang et al., 2018), DISTS (Ding et al., 2022), PSNR, and FID (Heusel et al., 2017), respectively. (ii) For fast inference, the highly optimized implementation of GR-GIC encodes and decodes in 4.60 and 5.98 ms on an NVIDIA A100, achieving $16.73\times$ and $34.19\times$ speedups over the official OneDC implementation, respectively. Specifically, the reorganized P-RVQ implementation achieves a $3.14\times$ average speedup over the original implementation under the same FP32 setting. We further validate GR-GIC on an NVIDIA Jetson Orin NX, where it achieves latency comparable to existing lightweight GIC methods, supporting its deployment on edge devices.

Our contributions are summarized as follows:

- We introduce GR-GIC, an efficient GIC that combines GR and reorganized P-RVQ for competitive reconstruction performance and fast inference.
- GR uses entropy context to parameterize an adaptive refinement distribution around quantized latents. Reorganized P-RVQ replaces serial high dimensional residual updates with low dimensional lookups, reducing sequential computation and inference latency.
- For performance, experiments demonstrate competitive results across multiple reconstruction metrics. For fast inference, the highly optimized implementation achieves low latency on both GPU and edge platforms.

2 Related Work

Diffusion-based generative image compression. Diffusion models provide strong generative priors for reconstructing details that are not fully specified by compressed representations (Yang & Mandt, 2023; Careil et al., 2024). Recent GIC methods increasingly use pretrained diffusion models and improve their efficiency through fewer diffusion steps, one-step reconstruction, or lighter

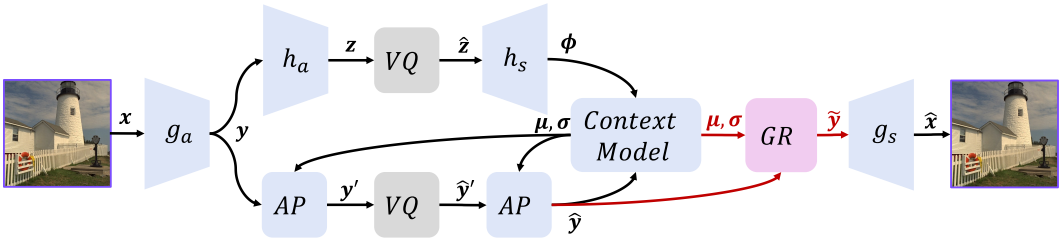


Figure 2: Overview of GR-GIC. The encoder g_a produces the latent $\boldsymbol{y}$, while the hyper encoder h_a and hyper decoder h_s provide the context feature $\boldsymbol{\phi}$. The context model predicts $(\boldsymbol{\mu}, \boldsymbol{\sigma})$ for the groupwise affine projections (APs) and P-RVQ. The GR head refines the recovered latent $\hat{\boldsymbol{y}}$ into $\tilde{\boldsymbol{y}}$ before the decoder g_s reconstructs $\hat{\boldsymbol{x}}$.

architectures (Li et al., 2025c; Zhang et al., 2025; Guo et al., 2025; Xue et al., 2025a; Jia et al., 2026). Despite these advances, their model size and computation remain substantial for efficient deployment.

VAE-based generative image compression. VAE-based GIC provides a more efficient alternative through adversarial priors, improved latent representations, and vector quantization (Mentzer et al., 2020; Muckley et al., 2023; Jia et al., 2024; Li et al., 2025a; Xue et al., 2025b; Jiang et al., 2026). Recent work further develops lightweight GIC. ProGIC combines a depthwise convolutional backbone with RVQ (Cao et al., 2026b), while EF-LIC models correlations among latents in the representation domain (Cao et al., 2026a). GR-GIC builds on EF-LIC, introducing GR for better reconstruction and reorganizing P-RVQ to move high dimensional reconstruction outside the sequential search path.

Latent refinement after quantization. Prior LIC methods have compensated quantization loss through latent residual prediction, post-quantization filtering, and adaptive quantization or correction (Minnen & Singh, 2020; Fu et al., 2023; Zhang et al., 2024; Ge et al., 2024). Recent work also studies latent and noise compensation (Yi et al., 2026). These methods mainly perform deterministic correction using decoder context, rely on guidance from the encoder for content-adaptive compensation, or model stochastic compensation without adapting it to context. In contrast, GR uses entropy context already available at the decoder to parameterize a refinement distribution that adapts to spatial location and image content, without additional payload, while adaptive perturbations provide training regularization for generative reconstruction.

3 Method

3.1 Overview

As illustrated in Figures 1a and 1b, at low bitrates, coarse deterministic quantization can collapse nearby encoder latents corresponding to different fine details to the same quantized point, making these details difficult to recover in lightweight GIC. GR-GIC builds on EF-LIC (Cao et al., 2026a), whose context model provides the recovered latent and entropy context without additional payload. GR uses the entropy context to parameterize a Gaussian refinement distribution around the recovered latent, with adaptive perturbations providing latent noise regularization during training. GR-GIC retains the analysis transform, hyperprior, groupwise context, and P-RVQ structure of EF-LIC, adds a GR head before synthesis as illustrated in Figure 2, and reorganizes P-RVQ execution in Figure 3.

Given an image $\boldsymbol{x} \in \mathbb{R}^{3\times H\times W}$, the encoder g_a produces the latent $\boldsymbol{y}$. The hyper encoder h_a maps $\boldsymbol{y}$ to the hyper latent $\boldsymbol{z}$, which the hyper latent RVQ $Q_{\boldsymbol{z}}$ quantizes to obtain $\hat{\boldsymbol{z}}$. The hyper decoder h_s reconstructs the context feature $\boldsymbol{\phi}$ from $\hat{\boldsymbol{z}}$:

$$\boldsymbol{y} = g_a(\boldsymbol{x}), \qquad \boldsymbol{z} = h_a(\boldsymbol{y}), \qquad \hat{\boldsymbol{z}} = Q_{\boldsymbol{z}}(\boldsymbol{z}), \qquad \boldsymbol{\phi} = h_s(\hat{\boldsymbol{z}}).$$

The latent $\boldsymbol{y}$ has a spatial downsampling factor of $f_y = 16$ and enters the groupwise affine projections. Following the quadtree context modeling of EF-LIC and DCVC-RT (Cao et al., 2026a; Jia et al., 2025), we partition $\boldsymbol{y}$ into $N = 4$ groups $\{\boldsymbol{y}_n\}_{n=1}^{N}$. For group n, the context model f_n combines $\boldsymbol{\phi}$ with the previously recovered groups to predict the mean $\boldsymbol{\mu}_n$ and scale $\boldsymbol{\sigma}_n$:

$$(\boldsymbol{\mu}_n, \boldsymbol{\sigma}_n) = f_n(\mathrm{concat}(\boldsymbol{\phi}, \hat{\boldsymbol{y}}_{<n})), \qquad \boldsymbol{\sigma}_n \geq \sigma_{\min}.$$

Before quantization, the affine projection (AP) centers and scales $\boldsymbol{y}_n$ using $\boldsymbol{\mu}_n$ and $\boldsymbol{\sigma}_n$. The inverse AP then reconstructs $\hat{\boldsymbol{y}}_n$ after quantization:

$$\boldsymbol{y}'_n = (\boldsymbol{y}_n - \boldsymbol{\mu}_n) \oslash \boldsymbol{\sigma}_n, \qquad \hat{\boldsymbol{y}}'_n = Q_n(\boldsymbol{y}'_n), \qquad \hat{\boldsymbol{y}}_n = \boldsymbol{\sigma}_n \odot \hat{\boldsymbol{y}}'_n + \boldsymbol{\mu}_n. \tag{1}$$

After all groups are recovered, their concatenation forms $\hat{\boldsymbol{y}}$. The recovered latent $\hat{\boldsymbol{y}}$ and the aligned context parameters $(\boldsymbol{\mu}, \boldsymbol{\sigma})$ are then fed to the GR head r_θ to produce $\tilde{\boldsymbol{y}}$, which the decoder g_s maps to the reconstruction $\hat{\boldsymbol{x}} = g_s(\tilde{\boldsymbol{y}})$.

The codec uses five RVQs, with $Q_{\boldsymbol{z}}$ quantizing the hyper latent $\boldsymbol{z}$ and Q_n quantizing each of the $N = 4$ main latent groups. At a given operating point, all five RVQs activate the same number $M \in \{1, 2, 3, 4\}$ of codebooks, and $m \in \{1, \dots, M\}$ indexes an active stage. With this shared RVQ depth and fixed width indices, the bit rate is $R_M = 0.0356M$ bits per pixel (BPP). Appendices B.3 and B.4 provide the exact rate calculation and boundary handling.

3.2 Guided Refinement of Quantized Latents

To improve low-bitrate reconstruction in lightweight GIC, we introduce Guided Refinement (GR) after RVQ recovery. GR uses entropy context to define a Gaussian refinement distribution around the recovered latent that adapts to spatial location and image content. During joint training, sampling from this distribution introduces adaptive perturbations that provide latent noise regularization.

We first formalize the effect of deterministic quantization that motivates this refinement. To express the mapping with the notation of Equation (1), we suppress group and stage indices, write Q for the deterministic quantizer, and denote its jth reconstruction point by $\hat{\boldsymbol{y}}'_j$. Its quantization cell is $\mathcal{C}_j = \{\boldsymbol{y}' : Q(\boldsymbol{y}') = \hat{\boldsymbol{y}}'_j\}$, and inverse projection gives $\hat{\boldsymbol{y}}_j = \boldsymbol{\sigma} \odot \hat{\boldsymbol{y}}'_j + \boldsymbol{\mu}$. The continuous latent distribution is transformed as

$$\begin{aligned} p_{\hat{\boldsymbol{Y}}}(\hat{\boldsymbol{y}} \mid \boldsymbol{\mu}, \boldsymbol{\sigma}) &= \int p_{\boldsymbol{Y}'}(\boldsymbol{y}' \mid \boldsymbol{\mu}, \boldsymbol{\sigma}) p_{\hat{\boldsymbol{Y}}}(\hat{\boldsymbol{y}} \mid \boldsymbol{Y}' = \boldsymbol{y}', \boldsymbol{\mu}, \boldsymbol{\sigma}) \, \mathrm{d}\boldsymbol{y}' \\ &= \sum_j \left(\int_{\mathcal{C}_j} p_{\boldsymbol{Y}'}(\boldsymbol{y}' \mid \boldsymbol{\mu}, \boldsymbol{\sigma}) \, \mathrm{d}\boldsymbol{y}' \right) \delta_{\mathrm{D}}(\hat{\boldsymbol{y}} - \hat{\boldsymbol{y}}_j). \end{aligned} \tag{2}$$

Here δ_{D} is the Dirac delta. Equation (2) shows that deterministic quantization assigns all latent realizations within one quantization cell to the same recovered point. The transmitted index therefore identifies the quantization cell, but does not preserve the continuous variation within that cell. At low bitrates, where the quantization cells are relatively coarse, this unresolved variation can correspond to different fine details that are difficult to determine from the recovered latent alone. This motivates learning a refinement distribution around the recovered point using information already provided by the entropy context.

We therefore use a Gaussian family conditioned on entropy context, as its simple mean-variance form separates the refinement center from adaptive perturbations without introducing additional latent correlations. Concatenating the groupwise reconstruction points in Equation (2) gives the recovered latent $\hat{\boldsymbol{y}}$. Let $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ denote a standard Gaussian variable for one latent realization. We model the refined latent distribution as

$$p_\theta(\tilde{\boldsymbol{y}} \mid \hat{\boldsymbol{y}}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \int p_{\boldsymbol{\epsilon}}(\boldsymbol{\epsilon}) \, p_\theta(\tilde{\boldsymbol{y}} \mid \boldsymbol{\epsilon}, \hat{\boldsymbol{y}}, \boldsymbol{\mu}, \boldsymbol{\sigma}) \, \mathrm{d}\boldsymbol{\epsilon} = \mathcal{N}\big(\tilde{\boldsymbol{y}} \mid \hat{\boldsymbol{y}} + \boldsymbol{\delta}, \mathrm{diag}(\boldsymbol{\beta}^2)\big) \, . \tag{3}$$

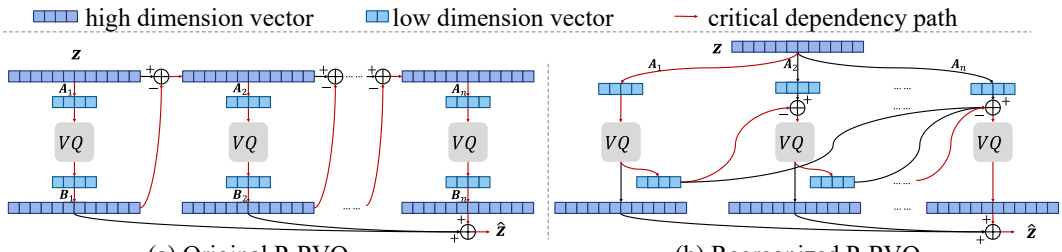
Here $\boldsymbol{\delta}$ shifts the recovered latent to the refinement center $\hat{\boldsymbol{y}} + \boldsymbol{\delta}$, while $\boldsymbol{\beta}$ controls the perturbation scale across spatial locations and image content. Both quantities are predicted from the recovered latent and entropy context parameters that are already available after decoding:

$$(\boldsymbol{\delta}, \tilde{\boldsymbol{\beta}}) = r_\theta(\mathrm{concat}(\hat{\boldsymbol{y}}, \boldsymbol{\mu}, \boldsymbol{\sigma})) \, , \qquad \boldsymbol{\beta} = \max(\tilde{\boldsymbol{\beta}}, \boldsymbol{0}). \tag{4}$$

The refined latent is then sampled as

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \qquad \tilde{\boldsymbol{y}} = \hat{\boldsymbol{y}} + \boldsymbol{\delta} + \boldsymbol{\beta} \odot \boldsymbol{\epsilon}. \tag{5}$$

All codec components are jointly optimized with one independent Gaussian draw per crop. The learned refinement center adjusts the recovered latent according to the entropy context, while the adaptive perturbation provides latent noise regularization during training. Section 4.3 further supports the important role of $\boldsymbol{\beta} \odot \boldsymbol{\epsilon}$ during training.



(a) Original P-RVQ (b) Reorganized P-RVQ

Figure 3: Reorganized P-RVQ. (a) Original P-RVQ alternates high dimensional residual updates, projections, and greedy search. (b) The equivalent form stacks online source projections and uses precomputed low dimensional lookups. Red arrows mark the critical dependency path.

3.3 Reorganized Projected Residual Vector Quantization

EF-LIC (Cao et al., 2026a) reported that P-RVQ takes 9.16 ms on Kodak, accounting for 52.0% of the measured 17.62 ms encoder latency and forming a major encoder latency bottleneck. For fast inference, we reorganize P-RVQ execution while preserving its greedy quantization process.

Within this subsection, $\boldsymbol{z} \in \mathbb{R}^D$ denotes an abstract input to a hyper or main latent P-RVQ. At stage m, $\boldsymbol{A}_m \in \mathbb{R}^{d\times D}$ projects the residual from high to low dimensions for search. The codebook $\boldsymbol{E}_m = [\boldsymbol{E}_m[1], \ldots, \boldsymbol{E}_m[K]] \in \mathbb{R}^{d\times K}$ stores K low dimensional codewords as columns, and $\boldsymbol{B}_m \in \mathbb{R}^{D\times d}$ maps them back to the reconstruction space, with projection biases suppressed. Starting from $\hat{\boldsymbol{z}}_0 = \boldsymbol{0}$, the partial reconstruction is defined for $m = 1, \ldots, M$ and returns $\hat{\boldsymbol{z}} = \hat{\boldsymbol{z}}_M$. As illustrated in Figure 3a, the original recursion carries a state in $\mathbb{R}^D$, projects the residual into $\mathbb{R}^d$ for search, and maps the selected codeword back to $\mathbb{R}^D$ before the next stage, so both projection directions remain on the critical dependency path:

$$\textsc{Original}: \quad \hat{\boldsymbol{z}}_m = \underbrace{\hat{\boldsymbol{z}}_{m-1} + \boldsymbol{B}_m \boldsymbol{E}_m \left[\arg\min_{1\le j\le K} \|\boldsymbol{A}_m(\boldsymbol{z} - \hat{\boldsymbol{z}}_{m-1}) - \boldsymbol{E}_m[j]\|_2^2 \right]}_{\text{critical dependency path with high dimensional updates: } \mathbb{R}^D\to\mathbb{R}^d\to\mathbb{R}^D}. \tag{6}$$

We derive an algebraically equivalent reorganization that stacks the source projections and expresses subsequent residual updates as low dimensional lookups, as illustrated in Figure 3b. Since the projection matrices and codebooks are fixed after training, each product $\boldsymbol{A}_m \boldsymbol{B}_\ell \boldsymbol{E}_\ell \in \mathbb{R}^{d\times K}$ can be fused offline into a new d dimensional codebook. Here, i_ℓ denotes the codeword index selected at an earlier stage ℓ:

$$\textsc{Reorganized}: \quad \hat{\boldsymbol{z}}_m = \hat{\boldsymbol{z}}_{m-1} + \boldsymbol{B}_m \boldsymbol{E}_m \left[\arg\min_{1\le j\le K} \left\| \boldsymbol{A}_m \boldsymbol{z} - \underbrace{\sum_{\ell<m} \boldsymbol{A}_m \boldsymbol{B}_\ell \boldsymbol{E}_\ell[i_\ell]}_{\text{critical dependency path using new } d \text{ dimensional codebooks fused offline}} - \boldsymbol{E}_m[j] \right\|_2^2 \right]. \tag{7}$$

The reorganized search evaluates all source projections together and updates later search states through lookups in $\mathbb{R}^d$. The high dimensional reconstruction is gathered from offline tables only after the codeword indices are selected. By linearity, the original and reorganized forms preserve the same search states, greedy assignments, and final reconstruction in real arithmetic. The reorganization therefore moves high dimensional reconstruction outside the critical dependency path, exposing greater parallelism and shortening the sequential computation path.

Under the same Torch FP32 backend and precision, the reorganized P-RVQ implementation achieves a $3.14\times$ average speedup over the original implementation across the 16 evaluated stage count and resolution settings. This matched comparison isolates the gain from the algebraic reorganization, which accounts for 23.54% of the total encoder latency reduction. Building on this reorganization, the CUDA FP16 implementation further increases the average speedup to $7.15\times$ relative to the original FP32 implementation. At the highest evaluated Kodak BPP, the compression-only P-RVQ path reaches a $18.80\times$ speedup, and the complete P-RVQ acceleration accounts for 31.28% of the total encoder latency reduction.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Implementation Details. Our implementation largely follows EF-LIC (Cao et al., 2026a), with a GR head added before synthesis and P-RVQ execution reorganized as described above. Appendix B provides the complete architecture and codebook settings. For fast inference, the highly optimized implementation further uses fused CUDA operators following DCVC-UF (Li et al., 2026).

Training Details. Each operating point is trained on ImageNet (Deng et al., 2009) in two 1M-iteration stages. Stage one samples 1% per epoch with 256×256 random crops at a learning rate of 10^{-4}. Stage two samples 10% of images exceeding 512 pixels in both dimensions with 512×512 random crops at 10^{-5}. Both stages apply random horizontal flipping and use AdamW (Loshchilov & Hutter, 2019) with weight decay 10^{-4}, $(\beta_1, \beta_2) = (0.9, 0.999)$, batch size 16, and one NVIDIA A100. Each training crop uses one standard Gaussian draw $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ to sample the refinement distribution conditioned on entropy context and estimate the expected image-space objective, which includes a generative adversarial network (GAN) term:

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{\epsilon}}[\lambda_{\mathrm{rec}}\mathcal{L}_2 + \lambda_{\mathrm{per}}\mathcal{L}_{\mathrm{LPIPS}} + \lambda_{\mathrm{adv}}\mathcal{L}_{\mathrm{GAN}}] + \lambda_{\mathrm{cb}}\mathcal{L}_{\mathrm{cb}} + \lambda_{\mathrm{com}}\mathcal{L}_{\mathrm{com}}. \tag{8}$$

Here $\mathcal{L}_2 = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2^2$ is averaged over pixels and channels, $\mathcal{L}_{\mathrm{GAN}}$ is the generator-side PatchGAN term, and $\mathcal{L}_{\mathrm{cb}}$ and $\mathcal{L}_{\mathrm{com}}$ are the codebook and commitment terms. We use $\lambda_{\mathrm{rec}} = 18$, $\lambda_{\mathrm{per}} = 2$, $\lambda_{\mathrm{cb}} = 1$, and $\lambda_{\mathrm{com}} = 1$, with adaptive λ_{adv} following VQGAN (Esser et al., 2021). All configurations in the study share the PatchGAN rule, training data, and losses. For the final codec, the adaptive perturbation provides latent noise regularization while all codec components are jointly optimized.

Evaluation Datasets and Metrics. We evaluate Kodak, Tecnick, DIV2K validation, and CLIC 2020 Professional (Eastman Kodak Company, 1993; Asuni & Giachetti, 2014; Agustsson & Timofte, 2017; Toderici et al., 2020). Metrics are LPIPS with a Visual Geometry Group (VGG) backbone, DISTS, PSNR, and patch-based FID (Zhang et al., 2018; Simonyan & Zisserman, 2015; Ding et al., 2022; Heusel et al., 2017; Jia et al., 2024). LPIPS and DISTS capture perceptual similarity, PSNR reports RGB pixel fidelity, and FID measures distribution-level alignment. All codecs use the same metric pipeline on decoded images for direct curves and pairwise BD-rate calculations. Appendix C gives dataset, metric, and pairwise PCHIP details (Bjøntegaard, 2001).

Compared Methods. The pretrained diffusion group comprises ResULIC, StableCodec, RDEIC, OneDC, OSCAR, DiT-IC, AEIC, and CoD-Lite (Ke et al., 2025; Zhang et al., 2025; Li et al., 2025c; Xue et al., 2025a; Guo et al., 2025; Shi et al., 2026; Zhang et al., 2026; Jia et al., 2026). The VAE-based group comprises MS-ILLM, Control-GIC, GLC, DLF, ProGIC, RDVQ, EF-LIC, and GR-GIC (Muckley et al., 2023; Li et al., 2025a; Jia et al., 2024; Xue et al., 2025b; Cao et al., 2026b; Jiang et al., 2026; Cao et al., 2026a). Runtime uses batch size one with 512×768 images on one NVIDIA A100. Latency follows the implementation and timing protocols in Appendix C.2, with parameters and forward FLOPs reported as additional model complexity measures. CoD-Lite uses our measurements of its official code, with its paper profile in Appendix C.3.

4.2 COMPARISON WITH PREVIOUS METHODS

Figure 4 and Table 1 compare the reconstruction performance and inference latency of GR-GIC with previous GIC methods. For performance, GR-GIC achieves bitrate savings over OneDC for LPIPS, DISTS, and FID across all four datasets. On Kodak, the OneDC-anchored BD-rate savings are 23.19%, 37.71%, 22.30%, and 22.14% for LPIPS, DISTS, PSNR, and FID, respectively.

For fast inference, the highly optimized implementation of GR-GIC encodes and decodes a 512×768 image in 4.60 and 5.98 ms on an NVIDIA A100. The official OneDC implementation takes 76.98 and 204.47 ms under the evaluated system settings, making the corresponding latency ratios $16.73\times$ and $34.19\times$. Appendix C.2 details the numerical precision, acceleration paths, and timing boundaries used for the compared systems.

Among the evaluated VAE-based methods, GR-GIC improves the rate-quality curves of previous lightweight codecs across the reported reconstruction metrics while retaining low inference latency.

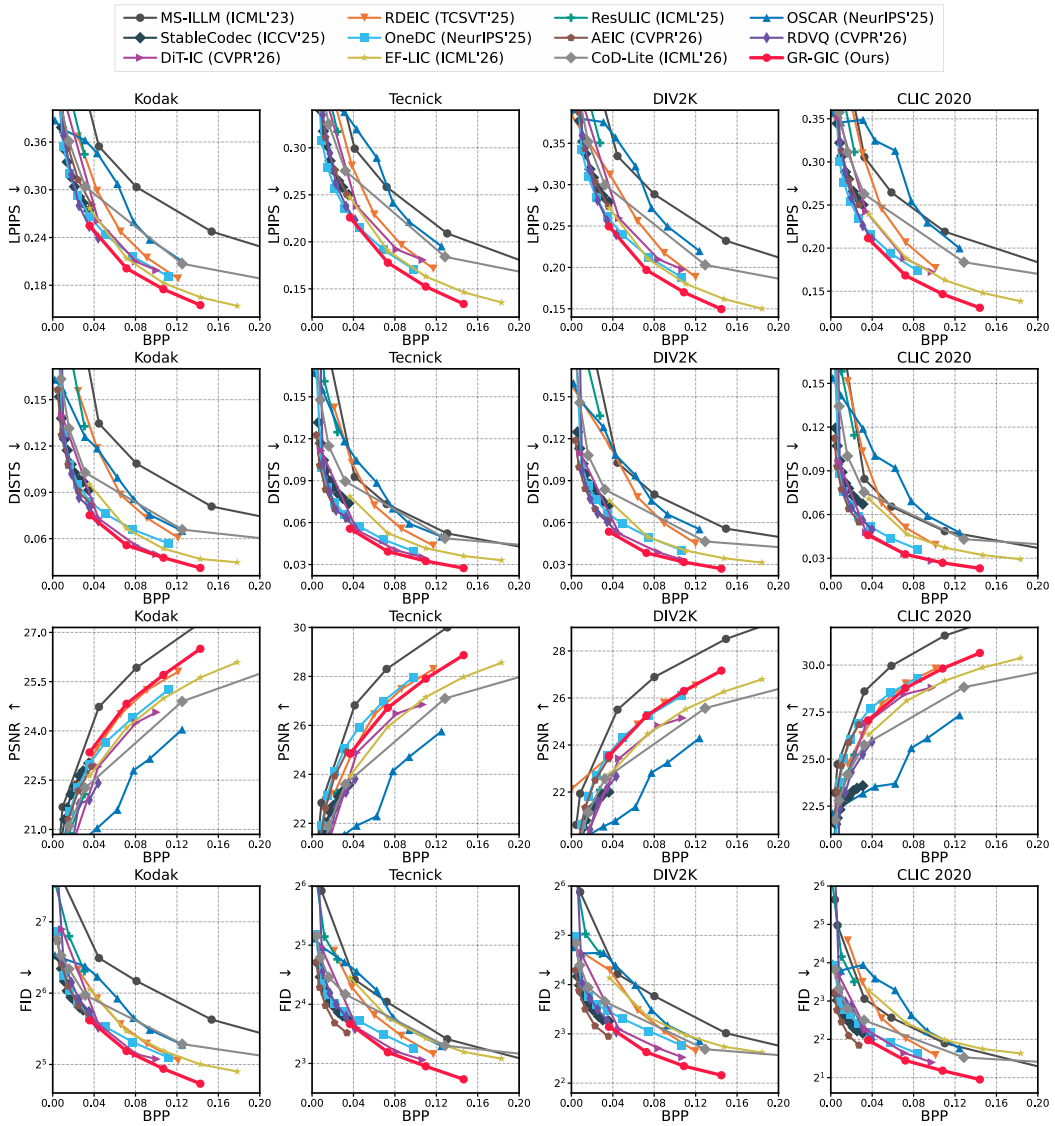


Figure 4: Rate and reconstruction quality across four datasets. Lower LPIPS, DISTS, and FID and higher PSNR indicate better quality. FID uses base-2 logarithmic vertical axes.

Taken together, these results show that GR-GIC combines gains across multiple reconstruction metrics with fast inference.

At similar rates, the qualitative comparisons in Figure 5 further illustrate the reconstruction differences. In the first row, GR-GIC preserves the thin eyeglass frame and nose pad. In the second row, it reconstructs the small circular fixture with coherent local geometry. These examples are consistent with the quantitative gains and show clearer reconstruction of local structures.

4.3 Guided Refinement Configuration Study

We compare five refinement configurations, each trained from scratch for 1M iterations at each rate under identical optimization conditions. M_0 bypasses refinement. M_1 predicts an offset from the recovered latent, while M_2 additionally uses entropy context. M_3 introduces stochastic perturbation with a channel-wise scale, and M_4 further adapts this scale to spatial location and image content. Both M_2 and M_3 match M_4 in parameter count and forward FLOPs, providing matched baselines that control for gains from increased model size or computation.

Table 1: Results on the Kodak dataset, with latency measured on one NVIDIA A100 GPU. Encoding and decoding latencies are in milliseconds (ms), FLOPs in billions (G), parameters in millions (M), and BD-rate in percent. OneDC is the BD-rate anchor, and lower values are better for every reported measure. Bold and underlined values indicate the best and second best results, respectively.

Family	Method	Complexity				BD-rate (%)			
		Enc.	Dec.	FLOPs	Params	LPIPS	DISTS	PSNR	FID
Pretrained diffusion	ResULIC	96.24	428.32	8549.01	1390.00	206.40	260.74	10.77	304.92
	StableCodec	100.18	200.03	6820.28	1065.81	8.03	6.77	-17.10	-4.45
	RDEIC	157.25	426.68	7767.46	1380.27	69.79	166.61	-12.14	80.26
	OneDC	76.98	204.47	8557.18	1406.42	0.00	0.00	0.00	0.00
	OSCAR	53.04	167.56	6485.61	1009.30	127.15	131.50	154.09	159.92
	DiT-IC	97.62	151.89	5743.15	1069.41	35.99	-6.44	33.24	56.47
	AEIC	12.39	82.98	2143.06	885.70	41.65	-16.71	2.11	7.16
	CoD-Lite	9.33	20.86	771.69	80.15	44.53	51.35	18.72	53.06
VAE-based	MS-ILLM	165.38	147.79	599.52	181.40	293.59	364.46	**-47.46**	468.10
	Control-GIC	103.56	436.26	5816.37	130.36	602.72	807.48	175.78	855.14
	GLC	34.75	48.26	2466.28	163.99	21.53	54.70	56.04	82.95
	DLF	168.78	240.63	5880.53	1286.01	20.87	20.19	42.73	70.80
	ProGIC	7.64	10.99	333.38	**33.11**	36.47	80.41	49.84	156.32
	RDVQ	408.24	412.44	3090.29	251.84	17.06	9.87	62.13	49.51
	EF-LIC	17.62	13.72	**279.61**	35.74	-3.33	10.05	11.49	49.01
	GR-GIC	**4.60**	**5.98**	456.55	79.39	**-23.19**	**-37.71**	-22.30	**-22.14**

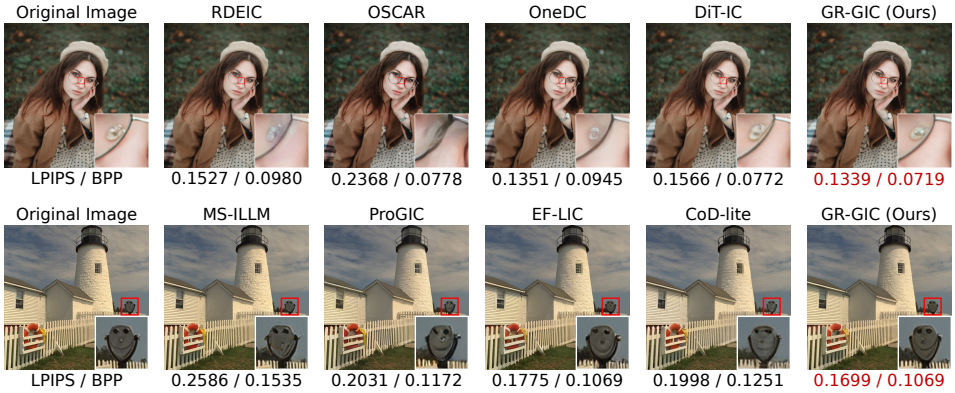


Figure 5: Qualitative comparisons of GR-GIC with high-capacity diffusion and lightweight codecs at similar rates. Boxes mark local details. Labels report LPIPS and BPP, where lower is better.

In Table 2, M_0 removes GR, retains the direct synthesis path of the EF-LIC backbone, and shows performance close to EF-LIC. Because M_0 and M_1–M_4 are trained under the same conditions, M_0 also provides a direct reference for comparing GR with the EF-LIC backbone. M_4 reduces FID by 4.2 to 7.8 relative to M_2 and by 1.5 to 4.9 relative to M_3, while achieving lower LPIPS at every evaluated rate. Since M_2 and M_3 match M_4 in parameter count and forward FLOPs, these results show that the complete GR configuration improves reconstruction performance on the lightweight EF-LIC backbone. We therefore select M_4 as the complete GR configuration.

To further understand the role of the perturbation, we set only the sampled term $\boldsymbol{\beta} \odot \boldsymbol{\epsilon}$ to zero during M_4 inference and observe only a small performance change. Together with the gains over deterministic M_2, this result supports an important role of adaptive perturbation as latent noise regularization during training. The gains over M_3 further show the benefit of adapting the perturbation scale to spatial location and image content. Appendix C.4 specifies the configuration training protocol, and Appendix D.7 reports the inference perturbation results.

Table 2: Guided Refinement configurations on Kodak at four nominal rates. The GR columns indicate a learned offset, decoder context, stochastic scale, and adaptive scale.

Model	GR configuration				BPP = 0.036		BPP = 0.071		BPP = 0.107		BPP = 0.143	
	Offset	Ctx.	Stoch.	Adapt.	LPIPS	FID	LPIPS	FID	LPIPS	FID	LPIPS	FID
M_0	✗	✗	✗	✗	0.277	66.2	0.214	51.2	0.184	43.6	0.165	38.1
M_1	✓	✗	✗	✗	0.271	63.6	0.212	46.8	0.184	39.2	0.164	35.4
M_2	✓	✓	✗	✗	0.264	62.4	0.208	41.2	0.182	36.5	0.162	31.9
M_3	✓	✓	✓	✗	0.264	59.5	0.207	37.9	0.182	33.8	0.162	29.5
M_4	✓	✓	✓	✓	**0.261**	**54.6**	**0.206**	**36.0**	**0.180**	**32.3**	**0.160**	**27.6**

Table 3: Encoding (Enc.) and decoding (Dec.) latency (ms) on an NVIDIA Jetson Orin NX. Bold and underline mark the two best reported values in each column.

Method	512×768		1080p		2K		4K	
	Enc.	Dec.	Enc.	Dec.	Enc.	Dec.	Enc.	Dec.
EF-LIC	**71.253**	<u>143.040</u>	<u>347.490</u>	<u>633.721</u>	<u>623.651</u>	<u>1137.985</u>	<u>1441.824</u>	<u>2628.537</u>
ProGIC	<u>83.204</u>	164.517	413.956	781.443	748.899	1412.819	1670.957	3182.561
AEIC	96.784	1290.483	–	> 10 s	–	> 10 s	OOM	
CoD-Lite	114.456	239.371	564.422	1348.999	974.139	2489.412	–	> 10 s
OneDC	773.388	1981.831	OOM		OOM		OOM	
GR-GIC	90.342	**99.850**	**265.745**	**428.361**	**465.576**	**754.362**	**1061.516**	**1739.863**

4.4 Edge-Device Deployment

For fast inference on edge hardware, we evaluate GR-GIC on a 16 GB NVIDIA Jetson Orin NX. Among the compared methods, GR-GIC achieves the lowest decoding latency at all four resolutions and the lowest encoding latency from 1080p to 4K in Table 3. Measurements use MAXN SUPER mode, batch size one, and synchronized wall-clock timing. GR-GIC remains deployable at 4K, where OneDC and AEIC run out of memory and CoD-Lite decoding exceeds 10 seconds. These results show that the low-latency implementation remains effective on edge hardware.

5 Limitations

Our evaluation spans 0.0356 to 0.1426 BPP, with the lowest operating point already below the 0.05 BPP ultra-low bitrate threshold (Zhang et al., 2025; 2026). The remaining challenge is extending lightweight GIC to even lower bitrates, where the transmitted representation specifies less fine detail and places greater demands on the generative capacity of the decoder. Future work can explore stronger lightweight priors and synthesis capacity while retaining fast inference. Our latency results characterize the complete deployed codec, including the proposed P-RVQ reorganization and CUDA acceleration, rather than the backbone alone. We therefore do not claim that the backbone itself achieves the fastest FP32 inference under identical backends. The compared methods retain their official or released implementations and available acceleration paths, so the reported latency reflects system-level performance under their respective implementations.

6 Conclusion

We introduced GR-GIC, an efficient GIC combining GR and reorganized P-RVQ for competitive reconstruction performance and fast inference. GR refines quantized latents using entropy context, while adaptive perturbations provide latent noise regularization during training. Reorganized P-RVQ replaces serial high dimensional residual updates with low dimensional lookups to reduce sequential computation. Experiments across four datasets show consistent gains in LPIPS, DISTS, and FID over OneDC, together with fast inference on both an NVIDIA A100 and Jetson Orin NX.

AI USE STATEMENT

Generative AI tools were used to assist with language editing and source-code organization. The authors reviewed all AI-assisted content and take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

Section 4 specifies the training setup, datasets, metrics, and compared methods. The appendix provides the complete P-RVQ derivation and complexity analysis, model architecture and bitstream format, evaluation and runtime protocols, ablation settings, and additional experimental results. For GR, the main evaluation uses a fixed decoder seed, with inference perturbation and seed stability further evaluated in the appendix. The source code will be released upon acceptance.

REFERENCES

Eirikur Agustsson and Radu Timofte. NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 126–135, 2017.

Nicola Asuni and Andrea Giachetti. TESTIMAGES: a Large-scale Archive for Testing Visual Devices and Basic Image Processing Algorithms. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*, pp. 63–70. The Eurographics Association, 2014.

Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations (ICLR)*, 2018.

Jean Bégaint, Fabien Racapé, Simon Feltman, and Akshay Pushparaja. CompressAI: a PyTorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*, 2020.

Gisle Bjøntegaard. Calculation of average PSNR differences between RD-curves. Technical Report VCEG-M33, ITU-T Video Coding Experts Group (VCEG), Austin, Texas, USA, April 2001.

Hao Cao, Wenqi Guo, Zhijin Qin, and Jungong Han. Efficient Learned Image Compression without Entropy Coding. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026a.

Hao Cao, Chengbin Liang, Wenqi Guo, Zhijin Qin, and Jungong Han. ProGIC: Progressive and Lightweight Generative Image Compression with Residual Vector Quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Findings*, pp. 2915–2925, June 2026b.

Marlène Careil, Matthew J. Muckley, Jakob Verbeek, and Stéphane Lathuilière. Towards image compression with perfect realism at ultra-low bitrates. In *The Twelfth International Conference on Learning Representations*, 2024.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255. IEEE, June 2009.

Nicki Skafte Detlefsen, Jiri Borovec, Justus Schock, Ananya Harsh Jha, Teddy Koker, Luca Di Liello, Daniel Stancl, Changsheng Quan, Maxim Grechkin, and William Falcon. TorchMetrics - Measuring Reproducibility in PyTorch. *Journal of Open Source Software*, 7(70):4101, 2022.

Keyan Ding, Kede Ma, Shiqi Wang, and Eero P. Simoncelli. Image Quality Assessment: Unifying Structure and Texture Similarity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(5):2567–2581, 2022.

Eastman Kodak Company. Kodak Lossless True Color Image Suite (PhotoCD PCD0992). `https://r0k.us/graphics/kodak/`, 1993.

Patrick Esser, Robin Rombach, and Björn Ommer. Taming Transformers for High-Resolution Image Synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12873–12883, 2021.

Frederick N. Fritsch and Judy Butland. A Method for Constructing Local Monotone Piecewise Cubic Interpolants. *SIAM Journal on Scientific and Statistical Computing*, 5(2):300–304, 1984.

Haisheng Fu, Feng Liang, Jie Liang, Binglin Li, Guohe Zhang, and Jingning Han. Asymmetric Learned Image Compression With Multi-Scale Residual Block, Importance Scaling, and Post-Quantization Filtering. *IEEE Transactions on Circuits and Systems for Video Technology*, 33(8): 4309–4321, 2023.

Ziqing Ge, Siwei Ma, Wen Gao, Jingshan Pan, and Chuanmin Jia. NLIC: Non-Uniform Quantization-Based Learned Image Compression. *IEEE Transactions on Circuits and Systems for Video Technology*, 34(10):9647–9663, 2024.

Robert M. Gray and David L. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 44 (6):2325–2383, 1998.

Jinpei Guo, Yifei Ji, Zheng Chen, Kai Liu, Min Liu, Wang Rao, Wenbo Li, Yong Guo, and Yulun Zhang. OSCAR: One-Step Diffusion Codec Across Multiple Bit-rates. In *Advances in Neural Information Processing Systems*, volume 38, pp. 85267–85286, 2025.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Advances in Neural Information Processing Systems*, volume 30, pp. 6626–6637. Curran Associates, Inc., 2017.

Zhaoyang Jia, Jiahao Li, Bin Li, Houqiang Li, and Yan Lu. Generative Latent Coding for Ultra-Low Bitrate Image Compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 26088–26098, 2024.

Zhaoyang Jia, Bin Li, Jiahao Li, Wenxuan Xie, Linfeng Qi, Houqiang Li, and Yan Lu. Towards Practical Real-Time Neural Video Compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12543–12552, 2025.

Zhaoyang Jia, Naifu Xue, Zihan Zheng, Jiahao Li, Bin Li, Xiaoyi Zhang, Zongyu Guo, Yuan Zhang, Houqiang Li, and Yan Lu. CoD-Lite: Real-Time Diffusion-Based Generative Image Compression. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Shiyin Jiang, Wei Long, Minghao Han, Zhenghao Chen, Ce Zhu, and Shuhang Gu. Differentiable Vector Quantization for Rate-Distortion Optimization of Generative Image Compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14440–14450, 2026.

Joint Video Experts Team (JVET). VVC Test Model (VTM), Version 23.10. `https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/-/tags/VTM-23.10`, 2025. Accessed: 2025-06-05.

Anle Ke, Xu Zhang, Tong Chen, Ming Lu, Chao Zhou, Jiawen Gu, and Zhan Ma. Ultra Lowrate Image Compression with Semantic Residual Coding and Compression-aware Diffusion. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 29626–29650. PMLR, 2025.

Anqi Li, Feng Li, Yuxi Liu, Runmin Cong, Yao Zhao, and Huihui Bai. Once-for-All: Controllable Generative Image Compression with Dynamic Granularity Adaptation. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025a.

Jiahao Li, Wenxuan Xie, Zhaoyang Jia, Bin Li, Zongyu Guo, Xiaoyi Zhang, and Yan Lu. Ultra-Fast Neural Video Compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 41311–41321, 2026.

Yuqi Li, Haotian Zhang, Li Li, and Dong Liu. Learned Image Compression with Hierarchical Progressive Context Modeling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 18834–18843, 2025b.

Zhiyuan Li, Yanhui Zhou, Hao Wei, Chenyang Ge, and Ajmal Mian. RDEIC: Accelerating Diffusion-Based Extreme Image Compression With Relay Residual Diffusion. *IEEE Transactions on Circuits and Systems for Video Technology*, 35(11):11540–11552, 2025c.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Fabian Mentzer, George D. Toderici, Michael Tschannen, and Eirikur Agustsson. High-Fidelity Generative Image Compression. In *Advances in Neural Information Processing Systems*, volume 33, pp. 11913–11924. Curran Associates, Inc., 2020.

David Minnen and Saurabh Singh. Channel-Wise Autoregressive Entropy Models for Learned Image Compression. In *2020 IEEE International Conference on Image Processing (ICIP)*, pp. 3339–3343, 2020.

Matthew J. Muckley, Alaaeldin El-Nouby, Karen Ullrich, Hervé Jégou, and Jakob Verbeek. Improving Statistical Fidelity for Neural Image Compression with Implicit Local Likelihood Models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 25426–25443. PMLR, 2023.

Linfeng Qi, Zhaoyang Jia, Jiahao Li, Bin Li, Houqiang Li, and Yan Lu. Generative Latent Coding for Ultra-Low Bitrate Image and Video Compression. *IEEE Transactions on Circuits and Systems for Video Technology*, 35(10):10500–10515, 2025.

Xihua Sheng, Lingyu Zhu, Tianyu Zhang, Dong Liu, Shiqi Wang, and Jing Wang. CADC: Content Adaptive Diffusion-Based Generative Image Compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 32936–32946, 2026.

Junqi Shi, Ming Lu, Xingchen Li, Anle Ke, Ruiqi Zhang, and Zhan Ma. DiT-IC: Aligned Diffusion Transformer for Efficient Image Compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 25719–25729, 2026.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

Kedar Tatwawadi, Parisa Rahimzadeh, Zhanghao Sun, Zhiqi Chen, Ziyun Yang, Sanjay Nair, Divija Hasteer, and Oren Rippel. What Matters in Practical Learned Image Compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12095–12105, June 2026.

George Toderici, Wenzhe Shi, Radu Timofte, Lucas Theis, Johannes Ballé, Eirikur Agustsson, Nick Johnston, and Fabian Mentzer. Workshop and Challenge on Learned Image Compression (CLIC 2020). CVPR Workshop and Challenge, 2020. URL `https://archive.compression.cc/`.

Gregory K. Wallace. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, 1992.

Naifu Xue, Zhaoyang Jia, Jiahao Li, Bin Li, Yuan Zhang, and Yan Lu. One-Step Diffusion-Based Image Compression with Semantic Distillation. In *Advances in Neural Information Processing Systems*, volume 38, pp. 37108–37144, 2025a.

Naifu Xue, Zhaoyang Jia, Jiahao Li, Bin Li, Yuan Zhang, and Yan Lu. DLF: Extreme Image Compression with Dual-generative Latent Fusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 19227–19236, 2025b.

Ruihan Yang and Stephan Mandt. Lossy Image Compression with Conditional Diffusion Models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 64971–64995. Curran Associates, Inc., 2023.

Fangzhou Yi, Zhicheng Gong, and Hui Zeng. Rethinking Latent Variable in Learned Image Compression. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 8531–8540, 2026.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 586–595, 2018.

Tianyu Zhang, Xin Luo, Li Li, and Dong Liu. StableCodec: Taming One-Step Diffusion for Extreme Image Compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17379–17389, October 2025.

Tianyu Zhang, Dong Liu, and Chang Wen Chen. Ultra-Low Bitrate Perceptual Image Compression with Shallow Encoder. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12118–12128, June 2026.

Yichi Zhang, Zhihao Duan, Ming Lu, Dandan Ding, Fengqing Zhu, and Zhan Ma. Another Way to the Top: Exploit Contextual Clustering in Learned Image Coding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 9377–9386, 2024.

APPENDIX

In the appendix, we provide the following:

- Appendix A derives the reorganized P-RVQ, analyzes its complexity, and reports implementation details and latency results.
- Appendix B describes the model architecture, bitstream packing, padding, and nominal rate.
- Appendix C defines the dataset, metric, runtime, baseline, and GR ablation evaluation protocols.
- Appendix D reports additional efficiency and quality comparisons, high-resolution results, and GR inference stability.
- Appendix E provides additional visual comparisons on all four evaluation datasets.

A ANALYSIS OF REORGANIZED P-RVQ

The main text presents the key reorganization of P-RVQ with projection biases omitted for clarity. This appendix restores the complete affine transformations, establishes the equivalence between the original and reorganized forms, analyzes their computation and storage, and reports the corresponding runtime results.

A.1 COMPLETE P-RVQ REORGANIZATION

We use the notation of Section 3.3. Let $\boldsymbol{z} \in \mathbb{R}^D$ denote the input to a P-RVQ module, covering both the hyper latent and the main latent groups. The following equations describe one input vector, while L denotes the number of input vectors when counting operations.

At stage m, the affine projection $(\boldsymbol{A}_m, \boldsymbol{a}_m)$ maps the current residual from $\mathbb{R}^D$ to the search space $\mathbb{R}^d$, where $\boldsymbol{A}_m \in \mathbb{R}^{d\times D}$ and $\boldsymbol{a}_m \in \mathbb{R}^d$. The codebook $\boldsymbol{E}_m = [\boldsymbol{E}_m[1], \ldots, \boldsymbol{E}_m[K]] \in \mathbb{R}^{d\times K}$ contains K codewords. The reconstruction projection $(\boldsymbol{B}_m, \boldsymbol{b}_m)$ maps the selected codeword back to $\mathbb{R}^D$, where $\boldsymbol{B}_m \in \mathbb{R}^{D\times d}$ and $\boldsymbol{b}_m \in \mathbb{R}^D$. The value of K may differ among the five P-RVQ modules.

Starting from $\hat{\boldsymbol{z}}_0 = \boldsymbol{0}$, let $\hat{\boldsymbol{z}}_m$ denote the partial reconstruction after stage m, and let $\hat{\boldsymbol{z}} = \hat{\boldsymbol{z}}_M$ denote the final reconstruction.

Original execution. The original P-RVQ projects the current residual for nearest neighbor search and immediately adds the selected reconstruction to the high dimensional state:

$$i_m = \arg\min_{1\le j\le K} \|\boldsymbol{A}_m(\boldsymbol{z} - \hat{\boldsymbol{z}}_{m-1}) + \boldsymbol{a}_m - \boldsymbol{E}_m[j]\|_2^2, \tag{9}$$

$$\hat{\boldsymbol{z}}_m = \hat{\boldsymbol{z}}_{m-1} + \boldsymbol{B}_m\boldsymbol{E}_m[i_m] + \boldsymbol{b}_m. \tag{10}$$

The final reconstruction is therefore

$$\hat{\boldsymbol{z}} = \hat{\boldsymbol{z}}_M = \sum_{m=1}^{M} \left(\boldsymbol{B}_m\boldsymbol{E}_m[i_m] + \boldsymbol{b}_m\right). \tag{11}$$

Before stage $m+1$, the selected codeword must pass through $\boldsymbol{B}_m$ and update the state in $\mathbb{R}^D$. The next stage then projects this updated residual back to $\mathbb{R}^d$. This produces the $\mathbb{R}^D \to \mathbb{R}^d \to \mathbb{R}^D$ dependency path shown in Equation (6).

Reorganized execution. Expanding the partial reconstruction inside the search projection gives

$$\boldsymbol{A}_m(\boldsymbol{z} - \hat{\boldsymbol{z}}_{m-1}) + \boldsymbol{a}_m = \boldsymbol{A}_m\boldsymbol{z} + \boldsymbol{a}_m - \sum_{\ell<m} \boldsymbol{A}_m\left(\boldsymbol{B}_\ell\boldsymbol{E}_\ell[i_\ell] + \boldsymbol{b}_\ell\right).$$

All parameters and codewords in the terms from earlier stages are fixed after training. We therefore precompute the d dimensional lookup table

$$\boldsymbol{F}_{\ell\to m}[j] = \boldsymbol{A}_m\left(\boldsymbol{B}_\ell \boldsymbol{E}_\ell[j] + \boldsymbol{b}_\ell\right) = (\boldsymbol{A}_m\boldsymbol{B}_\ell)\boldsymbol{E}_\ell[j] + \boldsymbol{A}_m\boldsymbol{b}_\ell, \qquad \ell < m. \tag{12}$$

This restores the bias term omitted from the simplified expression in Equation (7). Using these tables, the reorganized recursion becomes

$$\begin{aligned} i_m &= \arg\min_{1\le j\le K} \left\| \boldsymbol{A}_m\boldsymbol{z} + \boldsymbol{a}_m - \sum_{\ell<m} \boldsymbol{F}_{\ell\to m}[i_\ell] - \boldsymbol{E}_m[j] \right\|_2^2, \\ \hat{\boldsymbol{z}}_m &= \hat{\boldsymbol{z}}_{m-1} + \boldsymbol{B}_m\boldsymbol{E}_m[i_m] + \boldsymbol{b}_m. \end{aligned} \tag{13}$$

The source projections for all stages are evaluated together:

$$\begin{bmatrix} \boldsymbol{s}_1^{(0)} \\ \vdots \\ \boldsymbol{s}_M^{(0)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_1 \\ \vdots \\ \boldsymbol{A}_M \end{bmatrix} \boldsymbol{z} + \begin{bmatrix} \boldsymbol{a}_1 \\ \vdots \\ \boldsymbol{a}_M \end{bmatrix}. \tag{14}$$

At stage m, the corresponding search state is

$$\boldsymbol{s}_m = \boldsymbol{s}_m^{(0)} - \sum_{\ell<m} \boldsymbol{F}_{\ell\to m}[i_\ell] = \boldsymbol{A}_m(\boldsymbol{z} - \hat{\boldsymbol{z}}_{m-1}) + \boldsymbol{a}_m. \tag{15}$$

After selecting i_ℓ, the later search states are updated through $\boldsymbol{F}_{\ell\to m}[i_\ell]$ for $m > \ell$. Equation 15 shows that the original and reorganized forms present the same search vector to every stage in real arithmetic. Starting from the first stage, the same nearest neighbor decision therefore produces the same update for all later stages. Induction over m preserves all greedy indices and the final reconstruction.

The reorganized execution does not need to materialize the high dimensional partial reconstruction between search stages. It maintains the low dimensional states $\boldsymbol{s}_m$ until all indices are selected, and then gathers and sums $\boldsymbol{B}_m\boldsymbol{E}_m[i_m] + \boldsymbol{b}_m$ to obtain Equation (11). The greedy stage order remains unchanged, while the high dimensional reconstruction is removed from the critical dependency path.

For nearest neighbor search, the part of the squared distance that depends on candidate j is

$$i_m = \arg\min_j \|\boldsymbol{s}_m - \boldsymbol{E}_m[j]\|_2^2 = \arg\min_j \left(\|\boldsymbol{E}_m[j]\|_2^2 - 2\boldsymbol{s}_m^\mathsf{T}\boldsymbol{E}_m[j] \right),$$

because $\|\boldsymbol{s}_m\|_2^2$ is constant across candidates. The codeword norms and codebook transposes can therefore be stored before inference.

A.2 Complexity and Parallelism

The reorganization preserves the projection used for search and the nearest neighbor computation. For L input vectors, the shared multiply-accumulate counts are

$$\begin{aligned} \mathcal{C}_{\text{proj}}^{\text{MAC}} &= L\sum_{m=1}^{M} Dd = LMDd, \\ \mathcal{C}_{\text{search}}^{\text{MAC}} &= L\sum_{m=1}^{M} Kd = LMKd, \\ \mathcal{C}_{\text{common}}^{\text{MAC}} &= LM(D+K)d. \end{aligned}$$

Both forms also require LMd additions for projection biases and $LM(K-1)$ candidate comparisons. In the evaluated codec, the four main latent P-RVQ modules use $D = 256$, $d = 16$, and $K \in \{1024, 512, 256, 128\}$. The hyper latent P-RVQ uses $D = 128$, $d = 8$, and $K = 1024$. Thus, all five modules satisfy $D = 16d$.

Reconstruction cost. The original execution evaluates $\boldsymbol{B}_m \boldsymbol{E}_m[i_m] + \boldsymbol{b}_m$ online for every selected codeword. The reorganized execution evaluates these projections for all codewords during checkpoint conversion and stores the resulting reconstruction tables. For one P-RVQ module,

$$\begin{aligned} \mathcal{C}^{\mathrm{MAC}}_{\mathrm{out,online}} &= LMDd, \\ \mathcal{C}^{\mathrm{MAC}}_{\mathrm{out,offline}} &= MKDd, \\ \mathcal{S}_{\mathrm{decode}} &= MKD. \end{aligned}$$

The original online path additionally performs LMD bias additions. The reorganized online path replaces the projections with table lookups and requires $L(M-1)D$ additions to sum the selected reconstruction vectors. Both forms produce the same reconstruction in Equation (11).

State update cost. The main reduction occurs in the sequential state updates. The corresponding addition counts and their ratio are

$$\mathcal{C}^{\mathrm{add}}_{\mathrm{high}} = L \sum_{m=2}^{M} D = L(M-1)D, \tag{16}$$

$$\mathcal{C}^{\mathrm{add}}_{\mathrm{low}} = L \sum_{m=2}^{M} (m-1)d = \frac{LM(M-1)d}{2}, \tag{17}$$

$$\frac{\mathcal{C}^{\mathrm{add}}_{\mathrm{high}}}{\mathcal{C}^{\mathrm{add}}_{\mathrm{low}}} = \frac{2D}{Md}. \tag{18}$$

For $M = 4$ and $D = 16d$, the original and reorganized state update counts become $48Ld$ and $6Ld$, respectively. The reorganization therefore uses $8\times$ fewer additions for state propagation. The projection MACs, nearest neighbor MACs, and number of ordered greedy decisions remain unchanged. This reduction characterizes the sequential state update path rather than the complete P-RVQ runtime.

Parallelism. The original schedule exposes one source projection at each stage. The reorganized schedule evaluates the M source projections together. After selecting i_ℓ, the lookup updates for all later stages can also be applied concurrently. The reorganized execution therefore exposes all M source projections and up to $M-1$ parallel low dimensional updates, while retaining the same M ordered nearest neighbor decisions.

Lookup storage. The cross stage tables are constructed once after training. Their offline computation and nonzero storage for one P-RVQ module are

$$\begin{aligned} \mathcal{C}^{\mathrm{MAC}}_{\mathrm{cross,offline}} &= KDd \sum_{m=2}^{M} (m-1) = \frac{M(M-1)KDd}{2}, \\ \mathcal{S}_{\mathrm{cross,nonzero}} &= Kd \sum_{m=2}^{M} (m-1) = \frac{M(M-1)Kd}{2}. \end{aligned}$$

For $M = 4$, six pairs satisfy $\ell < m$. Across all five P-RVQ modules, the corresponding nonzero content is

$$\mathcal{S}^{\text{all}}_{\text{nonzero}} = 6\,(1024{\cdot}16 + 512{\cdot}16 + 256{\cdot}16 + 128{\cdot}16 + 1024{\cdot}8) = 233{,}472.$$

The implementation uses a dense table with $M^2 = 16$ stage slots, giving

$$\mathcal{S}^{\text{all}}_{\text{dense}} = 16\,(1024{\cdot}16 + 512{\cdot}16 + 256{\cdot}16 + 128{\cdot}16 + 1024{\cdot}8) = 622{,}592.$$

The dense table has shape (M, K, Md) and occupies 1.19 MiB in FP16 or 2.38 MiB in FP32 across the five P-RVQ modules. This storage is prepared before inference and contains the static lookup values used by the reorganized search.

A.3 Implementation and Runtime Evaluation

CUDA implementation. Checkpoint conversion materializes the stacked search projection, codeword norms, codebook transposes, reconstruction tables, and the cross stage lookup tables in Equation (12). At runtime, the source projections in Equation (14) are evaluated with one matrix multiplication.

A fused CUDA kernel then processes the ordered P-RVQ stages. Candidate dot products, distance evaluation, parallel reduction for index selection, table lookup, and updates to the later low dimensional states are performed within one kernel launch. The kernel preserves the ordered greedy decisions while parallelizing candidate evaluation and the updates to future search states. This avoids separate search launches at every stage and avoids materializing complete distance and update tensors.

When reconstruction is required, a second kernel gathers the selected high dimensional vectors, sums them, and writes the reconstructed output. During decompression, these reconstruction operations are further combined with the surrounding codec operations. This codec integration lies outside the isolated P-RVQ timing reported below.

Benchmark protocol. We evaluate the original Torch FP32 implementation, the reorganized Torch implementations in FP32 and FP16, and the CUDA FP16 implementation on one NVIDIA A100 with batch size one. All implementations use weights converted from the same checkpoints.

We evaluate $M \in \{1, 2, 3, 4\}$ at latent sizes corresponding to 512×768, 1080p, 2K, and 4K images, giving 16 settings in total. Each latency is the median of 100 synchronized repetitions after 20 warmup runs.

The reported latency isolates P-RVQ execution. Complete codec latency is reported in Tables 1, 3 and 10.

Runtime results. Under the same Torch FP32 backend and precision, the reorganized P-RVQ achieves a $3.14\times$ average speedup over the original implementation across the 16 evaluated settings. The reorganized Torch FP16 implementation achieves a $2.97\times$ average speedup. These matched Torch results isolate the effect of the algebraic reorganization and numerical precision.

The CUDA FP16 implementation further increases the average speedup to $7.15\times$ relative to the original FP32 implementation. This number includes the algebraic reorganization, FP16 execution, and fused CUDA kernels. Table 4 reports the latency and speedup for every resolution and P-RVQ depth.

At the highest evaluated Kodak rate, the P-RVQ latency during compression is 11.2824 ms for the original FP32 implementation, 3.2435 ms for the reorganized Torch FP32 implementation, and 0.6001 ms for the CUDA FP16 implementation. The matched FP32 reorganization gives a $3.48\times$ speedup at this operating point. The CUDA FP16 implementation gives an $18.80\times$ speedup.

For complete encoder attribution, we construct an original P-RVQ reference by restoring the latency difference between the original and reorganized FP32 P-RVQ to the measured Torch FP32 encoder. Using the reduction from this reference to the final CUDA FP16 encoder as a common denominator, the algebraic reorganization accounts for 23.54% of the total encoder latency reduction. The complete P-RVQ acceleration, including the CUDA implementation, accounts for 31.28%.

Table 4: P-RVQ latency on an NVIDIA A100. Each entry reports the median CUDA event latency in ms followed by speedup over the corresponding original FP32 result. All implementations use weights converted from the same checkpoints.

Resolution	M	Original FP32	Reorganized FP32	Reorganized FP16	CUDA FP16
512×768	1	$3.84/1.00\times$	$1.54/2.50\times$	$1.54/2.49\times$	$0.93/4.12\times$
	2	$5.38/1.00\times$	$1.74/3.09\times$	$1.74/3.09\times$	$0.64/8.43\times$
	3	$10.80/1.00\times$	$3.29/3.29\times$	$3.33/3.24\times$	$0.87/12.38\times$
	4	$14.09/1.00\times$	$4.03/3.49\times$	$4.15/3.40\times$	$0.87/16.12\times$
1080p	1	$2.82/1.00\times$	$1.14/2.47\times$	$1.14/2.48\times$	$0.67/4.18\times$
	2	$5.42/1.00\times$	$1.77/3.07\times$	$1.83/2.97\times$	$0.66/8.26\times$
	3	$8.37/1.00\times$	$2.43/3.45\times$	$2.56/3.27\times$	$0.78/10.67\times$
	4	$10.59/1.00\times$	$3.03/3.49\times$	$3.20/3.31\times$	$0.95/11.16\times$
2K	1	$2.88/1.00\times$	$1.12/2.58\times$	$1.16/2.47\times$	$0.68/4.25\times$
	2	$5.40/1.00\times$	$1.74/3.09\times$	$1.96/2.76\times$	$0.87/6.23\times$
	3	$8.34/1.00\times$	$2.51/3.33\times$	$2.55/3.27\times$	$1.18/7.10\times$
	4	$10.83/1.00\times$	$3.06/3.55\times$	$3.27/3.31\times$	$1.48/7.32\times$
4K	1	$2.93/1.00\times$	$1.13/2.61\times$	$1.16/2.54\times$	$0.98/3.00\times$
	2	$5.97/1.00\times$	$1.92/3.11\times$	$3.00/1.99\times$	$1.67/3.58\times$
	3	$8.61/1.00\times$	$2.45/3.51\times$	$2.49/3.46\times$	$2.30/3.74\times$
	4	$11.42/1.00\times$	$3.09/3.69\times$	$3.24/3.52\times$	$2.99/3.82\times$
Mean speedup		$1.00\times$	$3.14\times$	$2.97\times$	$\mathbf{7.15}\times$

Numerical consistency. The original and reorganized Torch FP32 implementations select identical indices, with a maximum absolute reconstruction difference of 4.77×10^{-7}. Under FP16, the reorganized Torch and CUDA implementations differ on only 0.2293% of the evaluated indices. These differences mainly occur between nearby codewords and have little effect on the final reconstruction.

B Architecture and Bitstream Details

B.1 Backbone and Context Pathway

GR-GIC follows the lightweight backbone and context pathway of EF-LIC (Cao et al., 2026a). Table 5 gives the complete implemented topology.

For clarity, Section 3.1 absorbs the normalization used in the implementation into the analysis and synthesis transforms. The encoder first computes the latent before normalization,

$$\boldsymbol{v} = g_a(\boldsymbol{x}), \qquad \boldsymbol{z} = h_a(\boldsymbol{v}).$$

After quantizing the hyper latent, the hyper decoder produces the normalization scale and context feature,

$$(\boldsymbol{q}_{\mathrm{enc}}, \boldsymbol{\phi}) = h_s(\hat{\boldsymbol{z}}), \qquad \boldsymbol{y} = \boldsymbol{v} \oslash \boldsymbol{q}_{\mathrm{enc}}.$$

The normalized latent $\boldsymbol{y}$ is then processed by the groupwise affine projections and P-RVQ.

At the decoder, the same hyper decoder gives

$$(\boldsymbol{q}_{\mathrm{dec}}, \boldsymbol{\phi}) = h_s(\hat{\boldsymbol{z}}).$$

For a fixed checkpoint and bitstream, $\boldsymbol{q}_{\mathrm{enc}} = \boldsymbol{q}_{\mathrm{dec}}$, and reconstruction is obtained as

$$\hat{\boldsymbol{x}} = g_s(\boldsymbol{q}_{\mathrm{dec}} \odot \tilde{\boldsymbol{y}}).$$

The notation in Section 3.1 absorbs these deterministic scales into g_a and g_s. The normalization scale $\boldsymbol{q}_{\mathrm{dec}}$ and entropy context scale $\boldsymbol{\sigma}$ are lower bounded by 0.11.

The main latent uses the same four group quadtree context model as EF-LIC and DCVC-RT (Cao et al., 2026a; Jia et al., 2025). Each group contains one quarter of the spatial positions and is

Table 5: Detailed GR-GIC architecture. Conv, DCB, and SP denote convolution, DepthConvBlock, and subpixel upsampling, respectively. The analysis and synthesis depths count the repeated body DCBs, while the entry and final refinement blocks are listed separately.

Module	Implemented topology	Output or role
Analysis transform	$8\times$ pixel unshuffle, $192 \to 368$ entry DCB, six 368-channel body DCBs, 3×3 Conv with stride 2 to 256 channels	Main latent at $1/16$ resolution
Hyper analysis	1×1 Conv $256 \to 512$, two 512-channel DCBs, two residual downsampling stages with stride 2, final 128-channel projection	Hyper latent at $1/64$ resolution
Hyper synthesis	SP $128 \to 512$, one DCB, SP $512 \to 512$, two DCBs, 1×1 Conv to 512 channels	256-channel normalization scale $\boldsymbol{q}_{\mathrm{dec}}$ and 256-channel context feature $\boldsymbol{\phi}$
Context path	Quadtree support projection, projection of previously decoded groups, one group adaptor DCB, three shared 512-channel DCBs, final 1×1 Conv	256-channel entropy context mean $\boldsymbol{\mu}$ and scale $\boldsymbol{\sigma}$
GR head	1×1 Conv $768 \to 768$, three 768-channel DCBs, 1×1 Conv $768 \to 512$	256-channel offset $\boldsymbol{\delta}$ and adaptive perturbation scale $\boldsymbol{\beta}$
Synthesis transform	SP $256 \to 368$, twelve 368-channel body DCBs, one final refinement DCB, 1×1 Conv $368 \to 192$, $8\times$ pixel shuffle	RGB reconstruction
DCB	Pointwise mixing, weighted sigmoid linear unit (SiLU), 3×3 depthwise Conv, pointwise projection, fourfold feedforward expansion and projection	Residual feature transformation

Table 6: P-RVQ configuration. The four main latent quantizers operate on quadtree groups. The fifth quantizes the hyper latent. "Bits per stage" is the fixed index width.

Stream	y_1	y_2	y_3	y_4	z
Input dim.	256	256	256	256	128
Code dim.	16	16	16	16	8
Entries	1024	512	256	128	1024
Bits per stage	10	9	8	7	10

recovered using the previously decoded groups. The GR head receives the recovered latent and entropy context parameters $(\hat{\boldsymbol{y}}, \boldsymbol{\mu}, \boldsymbol{\sigma})$ and predicts the offset $\boldsymbol{\delta}$ and perturbation scale $\boldsymbol{\beta}$. Each P-RVQ depth $M \in \{1, 2, 3, 4\}$ defines one separately trained operating point with its corresponding GR head.

B.2 Fixed-Width Index Packing

Each power of two codebook in Table 6 uses a fixed index width according to the size of its quantization alphabet (Gray & Neuhoff, 1998). The hyper latent uses 10 bits per index, while the four main latent groups use 10, 9, 8, and 7 bits.

The multistage structure and low dimensional search spaces follow the P-RVQ configuration of EF-LIC (Cao et al., 2026a). Stored indices are flattened in $(\text{stage}, \text{batch}, \text{height}, \text{width})$ order and written with the least significant bit first. The five streams are ordered as $Q_{\boldsymbol{z}} \to Q_1 \to Q_2 \to Q_3 \to Q_4$.

A standalone bitstream stores the original image height and width as two unsigned 16 bit integers, followed by the five index streams. No additional side information is transmitted. We report payload BPP and exclude this 4 byte image size header. For a 512×768 image, the header contributes only

$$R_{\text{meta}} = \frac{32}{512 \cdot 768} = 8.14 \times 10^{-5} \text{ bpp}.$$

B.3 Padding and Cropping

The encoder pads the input on the right and bottom so that both spatial dimensions are multiples of 32. The hyper analysis path additionally pads the main latent to a multiple of four, while the main latent path retains its original logical shape. The quadtree groups, main latent P-RVQ modules, GR head, and synthesis transform therefore operate on this original latent shape. The decoder removes the corresponding padding and returns the reconstruction at the original image size.

B.4 Nominal Rate

The fixed index widths give an exact nominal payload rate before boundary padding. At each P-RVQ stage, the four main latent groups contribute

$$R_y = \frac{10 + 9 + 8 + 7}{4 \cdot 16^2} = 0.033203125 \text{ bpp},$$

while the hyper latent contributes

$$R_z = \frac{10}{64^2} = 0.00244140625 \text{ bpp}.$$

Thus, with M active P-RVQ stages,

$$R_M = M(R_y + R_z) = 0.03564453125M \text{ bpp}. \tag{19}$$

Boundary padding changes the number of transmitted indices and causes the small deviations from these nominal rates.

C Evaluation Protocol

C.1 Datasets and Objective Evaluation

Kodak, Tecnick, DIV2K validation, and CLIC 2020 Professional test contain 24, 100, 100, and 250 images, respectively. All reconstructed images are evaluated at their original image resolution.

Objective evaluation. We use standardized objective metrics as the primary quantitative evaluation. Subjective user studies can provide complementary perceptual evidence, but their results depend on the participant population, viewing conditions, presentation protocol, and selected test images, and naturally exhibit observer variability. These factors make subjective scores less suitable for reproducible comparisons across many methods, datasets, and operating points. Standardized objective metrics instead provide repeatable measurements under a fixed evaluation protocol.

This practice is also common in generative image compression. Representative methods including MS-ILLM, GLC, Control-GIC, DLF, OneDC, OSCAR, and RDVQ primarily evaluate reconstruction using objective metrics such as LPIPS, DISTS, and FID without reporting a subjective user study (Muckley et al., 2023; Jia et al., 2024; Li et al., 2025a; Xue et al., 2025b;a; Guo et al., 2025; Jiang et al., 2026). We therefore use LPIPS and DISTS to measure reference-based perceptual similarity, PSNR to measure pixel fidelity, and FID to measure distribution-level alignment, together with qualitative reconstructions for visual comparison.

LPIPS is computed with pretrained VGG features after mapping RGB inputs to $[-1, 1]$. DISTS receives RGB inputs in $[0, 1]$, and PSNR is computed in RGB.

FID. Following the patch evaluation protocol used for low-rate generative image compression (Jia et al., 2024), we compute FID from local image patches. Reference and reconstructed RGB images in $[0, 1]$ are rounded to unsigned 8-bit values and passed to the TorchMetrics implementation of FID (Detlefsen et al., 2022; Heusel et al., 2017).

Kodak uses 64×64 patches, while Tecnick, DIV2K, and CLIC 2020 use 256×256 patches. We extract one non-overlapping patch grid and a second grid shifted by half a patch along both spatial axes. This increases the number of Inception samples and improves the stability of distribution-level evaluation on the 24-image Kodak set. All compared methods use the same FID pipeline.

BD-rate. We use BD-rate (Bjøntegaard, 2001) to summarize the average rate difference between two rate-quality curves at matched reconstruction quality. For each metric, operating points are ordered by the metric value, and pairs without a valid common interval are left unreported.

Let q denote the metric value. We use monotone PCHIP interpolation (Fritsch & Butland, 1984) for the natural logarithm of bitrate. Let $f_{\mathrm{A}}(q)$ and $f_{\mathrm{T}}(q)$ denote the interpolated curves of the anchor and tested codec over the common interval $[q_{\min}, q_{\max}]$. The reported BD-rate is

$$\text{BD-rate} = 100 \left[\exp\left(\frac{1}{q_{\max} - q_{\min}} \int_{q_{\min}}^{q_{\max}} \left(f_{\mathrm{T}}(q) - f_{\mathrm{A}}(q) \right) \mathrm{d}q \right) - 1 \right]. \tag{20}$$

OneDC is the anchor for the main comparisons. Negative values indicate bitrate savings over the common interval. The numerical tables report pairwise BD-rate, while the rate-quality curves show the corresponding behavior over the evaluated operating points.

C.2 Runtime and Model Cost Protocol

All A100 codec runtimes use batch size one on one NVIDIA A100 GPU. For GR-GIC, CPU payload processing is performed on an AMD EPYC 7763.

The latency comparison retains the numerical precision, inference backend, and acceleration path provided by each available implementation. The reported values therefore characterize realized system latency rather than architecture-only speed under an identical backend. EF-LIC and ProGIC follow their published FP32 evaluation settings (Cao et al., 2026a;b), while the remaining methods retain their released inference settings.

The highly optimized implementation of GR-GIC combines reorganized P-RVQ with fused CUDA operators following DCVC-UF (Li et al., 2026). It uses FP16 execution and CUDA Graphs together with custom kernels for quantization and reconstruction. The remaining convolution operations use optimized ATen and cuDNN kernels.

The reported encoding and decoding times cover the complete codec path. Encoding includes index generation, transfer to the CPU, and bitstream packing. Decoding includes bitstream unpacking, transfer to the GPU, index recovery, one GR realization, and image synthesis. Timing starts after model loading, input construction, and CUDA Graph capture. We use two warmup runs followed by ten synchronized repetitions.

For GR-GIC, the reported latency and model cost are averaged over four independently trained operating points at 0.0356, 0.0713, 0.1069, and 0.1426 BPP. The same averaging convention is used for the A100 and Jetson results.

Parameter accounting across operating points. We report the parameter count of the complete codec deployed at one evaluated operating point. Each of the four independently trained GR-GIC checkpoints contains 79.389M parameters. The reported 79.389M therefore represents one deployed model rather than the combined storage of all four checkpoints.

This convention is also applied to methods that use separate checkpoints for different target rates. Among the compared methods, ResULIC, StableCodec, RDEIC, OneDC, MS-ILLM, DLF, DiT-IC, AEIC, and CoD-Lite use separately trained or fine-tuned model weights for different operating points (Ke et al., 2025; Zhang et al., 2025; Li et al., 2025c; Xue et al., 2025a; Muckley et al., 2023; Xue et al., 2025b; Shi et al., 2026; Zhang et al., 2026; Jia et al., 2026).

We use the same parameter accounting rule for both types of codec. The tables report the complete model required for the evaluated operating point and do not sum multiple checkpoints across the rate curve. The storage required to retain all supported operating points is a separate deployment property from the parameter count of one deployed codec.

FLOPs are measured for one encoder and decoder pass with batch size one on the corresponding PyTorch FP32 path under the profiler-supported operator scope. We count one multiply-accumulate operation as two FLOPs. The same pass definition and arithmetic convention are used for all methods. Peak memory denotes the maximum allocated GPU memory including the loaded model and active codec buffers.

C.3 Baseline Evaluation Protocol

Except where noted, we evaluate released checkpoints on the same NVIDIA A100 with batch size one. Each method follows the coding and inference procedure provided by its official implementation.

EF-LIC and ProGIC use their published FP32 settings (Cao et al., 2026a;b). OneDC follows its official path with BF16 autocast for the feedforward and codec networks while retaining the large VAE in FP32 (Xue et al., 2025a). CoD-Lite uses its official 0.0312 BPP checkpoint with BF16 inference (Jia et al., 2026). Under our full codec profiling scope, this checkpoint gives 771.69G FLOPs at 512×768. The CoD-Lite paper reports 535 kMAC per pixel, corresponding to 420.74G FLOPs under our arithmetic convention. We use our common profiling scope for the system comparison and retain the published value as a separate reference. Control-GIC uses its official granularity control, which we enumerate at intervals of 0.01, together with official tiling on DIV2K and CLIC 2020 (Li et al., 2025a).

For GLC (Jia et al., 2024), the released interface provides analysis and synthesis without a complete bitstream path. For the high-resolution codec comparison, we additionally use a CompressAI 1.2.6 integration (Bégaint et al., 2020) that produces and decodes the actual bitstream.

For OSCAR, rate and quality evaluation uses CPU offloading on high-resolution images when required, while the resource tables report the direct execution result (Guo et al., 2025).

CADC (Sheng et al., 2026) and PICO (Tatwawadi et al., 2026) do not provide the corresponding executable models used in our comparison. We therefore use their published curves or released numerical results only when the metric definition and evaluation range are compatible with ours.

For CoD-Lite, measurements from the released implementation differ from the latency values reported in its paper. At the 1024×1920 reference resolution, we measure 39.522 ms for encoding and 56.863 ms for decoding, compared with the reported 16 and 24 ms. We therefore use measurements from the released code under our common hardware and timing protocol rather than mixing measured and separately reported runtime values. The system comparisons use the official 0.0312 BPP checkpoint specified by the published speed evaluation.

C.4 GR Ablation Protocol

The configurations in Table 2 are trained independently from scratch at each bitrate. All configurations use the same training data, loss terms and weights, AdamW settings, learning-rate schedule, and update budget. Each ablation model is trained for 1M iterations. The main operating-point results use separately trained models with the complete 2M iteration schedule.

M_0 bypasses GR and directly synthesizes from the recovered quantized latent. It retains the direct synthesis path of the EF-LIC backbone and provides a reference under the same training conditions used for the GR configurations.

M_1 predicts an offset from the recovered latent. M_2 additionally uses entropy context for deterministic refinement. M_3 introduces Gaussian perturbation with one learned scale for each channel, shared across spatial locations. M_4 further adapts the perturbation scale to spatial location and image content.

Each crop for M_3 and M_4 uses one standard Gaussian draw to estimate the expected image objective. M_2 and M_3 have the same reported parameter count and forward FLOPs as M_4. They provide matched references for evaluating stochastic perturbation and its adaptation to image content and spatial location.

Table 7: Per operating point GR-GIC complexity, CUDA latency, and peak memory on one A100. FLOPs are reported in G for one encoder and decoder pass under the profiler-supported operator scope. Latency is in ms and peak GPU memory is in GiB.

Resolution	BPP	Params	FLOPs	Enc.	Dec.	Mem.
512×768	0.0356	79.389	456.491	4.475	5.949	0.183
	0.0713	79.389	456.531	4.523	5.947	0.185
	0.1069	79.389	456.571	4.717	6.028	0.187
	0.1426	79.389	456.610	4.674	6.009	0.189
1080p	0.0356	79.389	2425.111	13.817	21.428	0.250
	0.0713	79.389	2425.321	14.111	21.483	0.250
	0.1069	79.389	2425.532	14.137	21.389	0.252
	0.1426	79.389	2425.742	14.323	21.466	0.255
2K	0.0356	79.389	4293.353	22.447	35.987	0.312
	0.0713	79.389	4293.725	22.501	36.022	0.313
	0.1069	79.389	4294.097	22.880	35.897	0.315
	0.1426	79.389	4294.469	23.209	35.905	0.318

C.5 Stochastic Evaluation Protocol

The main results use a fixed decoder seed of 1926 and reconstruct each bitstream once. Stochastic sampling is applied only after deterministic RVQ index recovery, so the transmitted indices and payload do not depend on the decoder seed.

Inference perturbation ablation. To examine the role of the sampled perturbation after training, we use the same trained M_4 checkpoints as in Table 2. We retain the recovered indices, learned offset, and all model parameters, and set only $\boldsymbol{\beta} \odot \boldsymbol{\epsilon}$ in Equation (5) to zero during inference. The decoder therefore synthesizes from the learned refinement center

$$\tilde{\boldsymbol{y}} = \hat{\boldsymbol{y}} + \boldsymbol{\delta}.$$

The corresponding results are reported in the additional experiments.

Decoder seed stability. To measure sensitivity to the sampled Gaussian perturbation, we keep the checkpoint, transmitted indices, input images, and all deterministic computations fixed and vary only the decoder seed. We evaluate ten seeds on all Kodak images at each operating point. For every seed, metrics are first averaged over the dataset before variation across seeds is measured.

The resulting stability measurements are reported together with the additional GR experiments.

D Additional Experimental Results

This section reports additional experimental results under the evaluation protocols defined in Appendix C. We first examine the efficiency of different GR-GIC operating points and inference backends, then provide additional comparisons with existing codecs, and finally analyze the inference behavior of GR.

D.1 Efficiency across Operating Points

Table 7 reports the complexity and runtime of the four independently trained operating points at 512×768, 1080p, and 2K.

At 512×768, encoding latency ranges from 4.475 to 4.717 ms, while decoding latency ranges from 5.947 to 6.028 ms. At 1080p, the encoding and decoding ranges span at most 0.506 and 0.094 ms, respectively. At 2K, the corresponding ranges are 0.762 and 0.125 ms.

Table 8: GR-GIC backend comparison on one A100. Latency is in ms and peak GPU memory is in GiB. Online E2E denotes one consecutive pair of in-memory encode and decode calls.

Resolution	Backend	Enc.	Dec.	Online E2E	Mem.
512×768	PyTorch FP32	30.453	44.708	66.519	0.438
	PyTorch FP16	15.439	20.298	37.784	0.222
	CUDA FP16	4.597	5.983	10.353	0.186
1080p	PyTorch FP32	109.967	176.403	282.883	0.965
	PyTorch FP16	32.895	57.600	86.817	0.465
	CUDA FP16	14.097	21.442	35.567	0.252
2K	PyTorch FP32	185.468	312.454	495.019	1.446
	PyTorch FP16	55.968	95.262	145.070	0.691
	CUDA FP16	22.759	35.952	59.118	0.314
4K	PyTorch FP32	407.490	709.180	1114.092	2.909
	PyTorch FP16	120.631	221.758	327.342	1.373
	CUDA FP16	49.787	81.143	132.971	0.492

These results show that increasing the active P-RVQ depth changes runtime only modestly relative to the shared backbone.

D.2 GR-GIC Backend Comparison

We further compare three implementations of the same GR-GIC checkpoints to separate the effects of numerical precision and backend optimization.

At 512×768, the eager PyTorch FP32 implementation encodes in 30.453 ms and decodes in 44.708 ms. Relative to the official OneDC implementation, these times are $2.53\times$ and $4.57\times$ lower, respectively. The eager PyTorch FP32 implementation is also faster in both encoding and decoding than six of the eight displayed diffusion codecs, and this behavior remains consistent across the evaluated resolutions.

The CUDA FP16 implementation further reduces runtime relative to the PyTorch FP32 implementation. Across the evaluated resolutions, encoding is accelerated by $6.62\times$ to $8.18\times$, while decoding is accelerated by $7.47\times$ to $8.74\times$.

The eager FP32 results show that low latency is already present before the additional CUDA acceleration, while the optimized backend provides a further reduction in runtime.

D.3 Additional Baseline Comparisons

CADC and PICO do not provide the corresponding executable models used in our comparison. Following the protocol in Appendix C, we therefore compare against their published curves or released numerical results where the metric definitions and rate ranges are compatible.

CADC. At the shared lowest Kodak rate of 0.03564 BPP, interpolation of the published CADC curves gives approximately 23.15 dB PSNR and 0.087 DISTS (Sheng et al., 2026). At the same rate, GR-GIC achieves 23.35 dB PSNR and 0.075 DISTS. This corresponds to a PSNR increase of approximately 0.20 dB and a DISTS reduction of approximately 0.012.

CADC reports LPIPS using AlexNet features, while our main evaluation uses VGG features. We therefore restrict this comparison to the aligned PSNR and DISTS metrics.

PICO. PICO reports VGG-based LPIPS on CLIC 2020 (Tatwawadi et al., 2026). Over the common quality interval, GR-GIC obtains a PICO-anchored LPIPS BD-rate of -54.01%. This corresponds to a 54.01% average bitrate reduction at matched LPIPS over the evaluated common interval.

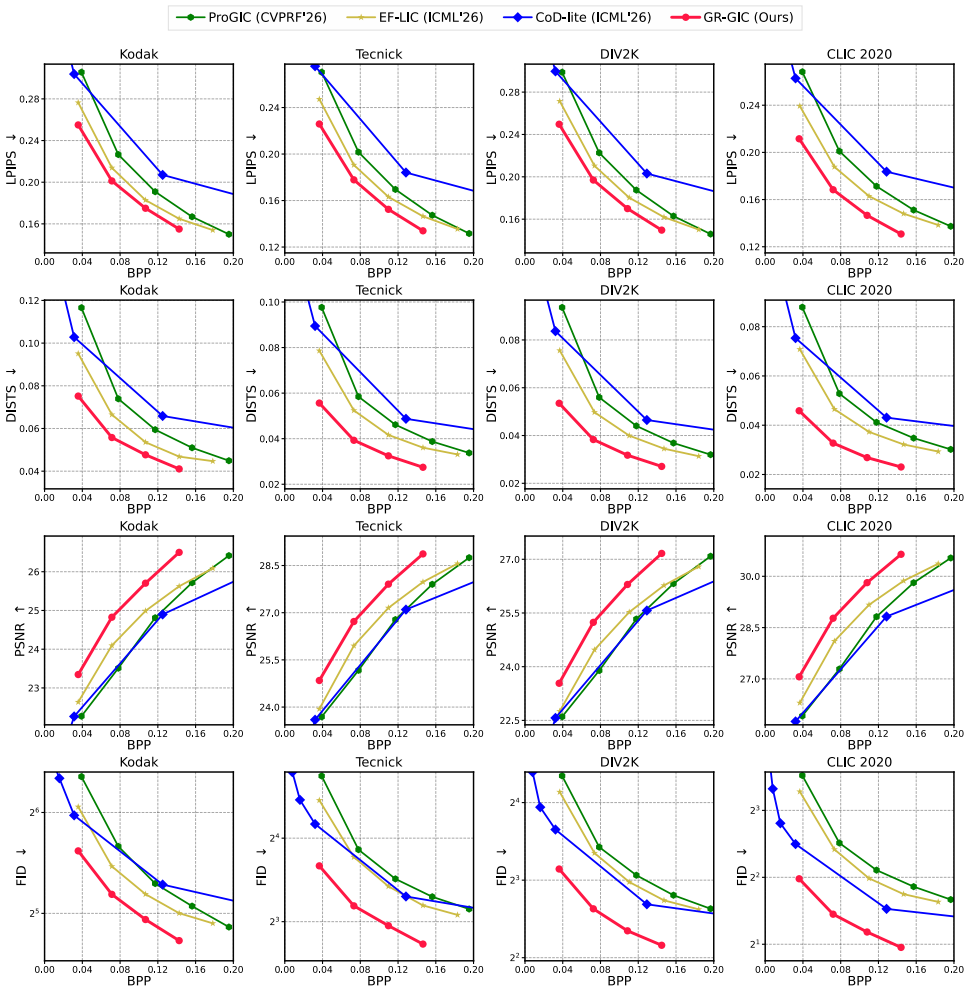


Figure 6: Rate and reconstruction quality of GR-GIC and lightweight codecs on four datasets. GR-GIC uses four operating points, while EF-LIC uses five. Lower LPIPS, DISTS, and FID and higher PSNR indicate better reconstruction. FID uses a base-2 logarithmic vertical axis.

D.4 Comparison with Lightweight Codecs

Figure 6 compares GR-GIC with ProGIC, EF-LIC, and CoD-Lite across all four datasets and metrics. At comparable rates, GR-GIC provides the strongest displayed LPIPS, DISTS, PSNR, and FID curves among these lightweight codecs.

The four GR-GIC operating points span approximately 0.036 to 0.143 BPP. The highest-rate GR-GIC point at 0.143 BPP uses a lower bitrate than the fifth EF-LIC point while achieving higher PSNR and lower DISTS and FID on every dataset. Its LPIPS is comparable or lower. The four GR-GIC operating points therefore cover the shared comparison range without requiring an additional higher-rate point.

D.5 Additional Rate and Quality Results

Table 9 reports OneDC-anchored BD-rate for LPIPS, DISTS, PSNR, and FID on Kodak, Tecnick, DIV2K, and CLIC 2020. Entries without a valid monotone common interval are left unavailable according to the protocol in Appendix C. CoD-Lite and MS-ILLM are already included in the main Kodak and high-resolution comparisons, so their full entries are not repeated across all four metric blocks in this table.

Table 9: Pairwise OneDC-anchored BD-rate (%). Bold and underline mark the two lowest displayed values per dataset.

Metric	Family	Method	Kodak	Tecnick	DIV2K	CLIC 2020
LPIPS	Diffusion	OneDC	0.00	0.00	0.00	0.00
		StableCodec	8.03	32.28	27.06	36.13
		AEIC	41.65	34.58	25.27	32.18
		DiT-IC	35.99	61.79	49.18	52.00
		OSCAR	127.15	225.78	201.57	–
	VAE-based	GLC	21.53	47.87	33.19	39.53
		DLF	20.87	54.78	47.02	45.66
		ProGIC	36.47	56.48	42.04	84.07
		EF-LIC	-3.33	15.12	3.51	24.62
		RDVQ	17.06	47.25	26.35	33.11
		GR-GIC	**-23.19**	**-11.27**	**-20.24**	**-18.39**
DISTS	Diffusion	OneDC	0.00	0.00	0.00	0.00
		StableCodec	6.77	31.27	9.38	24.27
		AEIC	-16.71	-20.02	-30.19	-21.29
		DiT-IC	-6.44	-0.28	-16.61	-10.82
		OSCAR	131.50	240.68	223.40	339.84
	VAE-based	GLC	54.70	72.21	50.63	70.55
		DLF	20.19	62.90	56.62	64.56
		ProGIC	80.41	126.30	78.51	172.95
		EF-LIC	10.05	44.93	13.25	69.31
		RDVQ	9.87	27.16	9.10	27.14
		GR-GIC	**-37.71**	**-28.12**	**-42.46**	**-31.53**
PSNR	Diffusion	OneDC	0.00	**0.00**	0.00	0.00
		StableCodec	-17.10	41.35	31.65	–
		AEIC	2.11	1.78	**-4.44**	**-12.35**
		DiT-IC	33.24	50.41	40.19	26.63
		OSCAR	154.09	248.22	195.61	338.47
	VAE-based	GLC	56.04	72.28	61.07	79.51
		DLF	42.73	78.89	63.48	67.35
		ProGIC	49.84	105.02	72.35	116.69
		EF-LIC	11.49	61.07	39.83	59.45
		RDVQ	62.13	70.23	60.92	70.88
		GR-GIC	**-22.30**	19.60	-0.41	16.48
FID	Diffusion	OneDC	0.00	0.00	0.00	0.00
		StableCodec	-4.45	-5.58	-26.77	-28.75
		AEIC	7.16	**-38.95**	**-49.80**	**-55.10**
		DiT-IC	56.47	41.17	30.56	44.53
		OSCAR	159.92	222.65	–	–
	VAE-based	GLC	82.95	54.94	50.70	90.98
		DLF	70.80	73.87	66.43	66.97
		ProGIC	156.32	267.15	225.46	389.82
		EF-LIC	49.01	143.61	138.95	257.90
		RDVQ	49.51	55.05	40.18	59.78
		GR-GIC	**-22.14**	-32.07	-45.68	-34.26

Across the twelve LPIPS, DISTS, and FID entries, GR-GIC achieves a negative OneDC-relative BD-rate on every dataset. The corresponding rate and quality curves in Figures 4 and 6 show the behavior over the evaluated operating ranges.

Table 10: High-resolution encoding and decoding latency (ms) and peak GPU memory (GiB) on one A100. Bold and underline mark the two best results.

Method	1080p			2K			4K		
	Enc.	Dec.	Mem.	Enc.	Dec.	Mem.	Enc.	Dec.	Mem.
ResULIC	481.75	3702.45	20.38	–	> 10 s	–	–	> 10 s	–
StableCodec	641.76	1657.43	21.80	1085.39	3161.54	30.65	–	> 10 s	–
RDEIC	683.20	2296.58	13.55	1436.00	5963.77	20.17	–	> 10 s	–
OneDC	303.33	742.06	12.81	585.72	1554.82	18.86	1542.18	4846.63	35.38
OSCAR	513.20	1123.38	24.75	2382.55	2719.79	79.44	–	–	> 80
DiT-IC	480.22	750.38	10.98	718.17	1111.59	14.47	1581.91	2448.49	25.41
AEIC	25.15	682.50	6.95	<u>30.69</u>	1943.15	9.87	<u>62.70</u>	9303.96	17.83
CoD-Lite	42.37	60.47	2.45	73.61	105.09	3.97	163.43	236.05	8.33
MS-ILLM	350.85	379.01	2.99	516.47	601.18	4.87	1305.93	1613.82	9.94
Control-GIC	610.76	2186.30	69.99	–	–	> 80	–	–	> 80
GLC	340.746	475.738	5.66	638.989	826.327	9.71	1456.258	1816.005	20.76
DLF	915.42	1416.49	10.00	1380.00	2153.39	12.61	3001.08	5122.21	22.35
ProGIC	28.40	55.00	1.07	38.63	80.38	1.46	78.26	145.24	2.58
RDVQ	704.64	809.99	6.01	953.80	1130.17	10.08	8575.63	8655.58	20.07
EF-LIC	<u>19.53</u>	<u>35.62</u>	<u>0.65</u>	31.58	<u>55.50</u>	<u>1.04</u>	65.24	<u>116.28</u>	<u>2.10</u>
GR-GIC	**14.10**	**21.44**	**0.25**	**22.76**	**35.95**	**0.31**	**49.79**	**81.14**	**0.49**

The table also places these reconstruction results in the context of model complexity. AEIC uses approximately $11.2\times$ the parameters and $4.7\times$ the FLOPs of GR-GIC. StableCodec uses approximately $13.4\times$ the parameters and $14.9\times$ the FLOPs. EF-LIC and ProGIC use fewer parameters and FLOPs than GR-GIC, while GR-GIC provides lower DISTS and FID over the reported comparisons.

D.6 High-Resolution Runtime and Memory

Table 10 extends the runtime comparison to 1080p, 2K, and 4K images on one NVIDIA A100.

The released GLC interface reports analysis and synthesis without a complete bitstream path. Its forward execution takes $(159.460, 259.762)$ ms at 1080p, $(291.436, 488.817)$ ms at 2K, and $(659.937, 1101.531)$ ms at 4K for analysis and synthesis, respectively. Under the complete CompressAI integration defined in Appendix C, the corresponding encoding and decoding times are $(340.746, 475.738)$ ms, $(638.989, 826.327)$ ms, and $(1456.258, 1816.005)$ ms. Peak memory is 5.66, 9.71, and 20.76 GiB at the three resolutions.

Among the available implementations in Table 10, GR-GIC achieves the lowest encoding latency, decoding latency, and peak memory at all three resolutions. Relative to the second-lowest values, the reductions correspond to factors of $1.26\times$ to $1.39\times$ for encoding, $1.43\times$ to $1.66\times$ for decoding, and $2.58\times$ to $4.27\times$ for peak memory.

EF-LIC provides the second-lowest decoding latency and peak memory across the three resolutions. AEIC provides the second-lowest encoding latency at 2K and 4K. These results show that the fast inference of GR-GIC extends from Kodak resolution to high-resolution images.

D.7 GR Inference Stability

Inference perturbation. We first evaluate the effect of the sampled perturbation at inference using the protocol in Appendix C.5. With the perturbation term removed, the decoder uses only the learned refinement center $\hat{\boldsymbol{y}} + \boldsymbol{\delta}$.

In the same rate order as Table 2, the resulting LPIPS and FID pairs are $(0.262, 54.8)$, $(0.206, 36.4)$, $(0.181, 32.5)$, and $(0.162, 27.9)$.

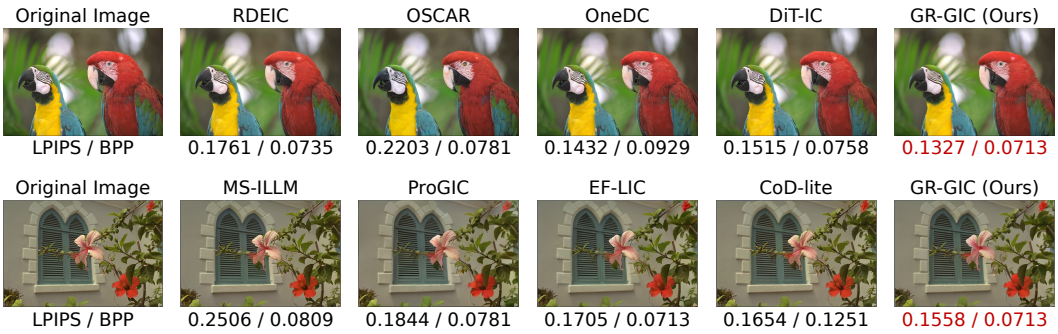


Figure 7: Additional Kodak visuals with LPIPS and BPP labels, where lower is better.

Relative to the complete M_4 results, removing the sampled perturbation changes LPIPS by only 0.000 to 0.002 and increases FID by 0.2 to 0.4. Most of the gain of the complete GR configuration therefore remains when inference uses only the learned refinement center.

Without sampled perturbation at inference, M_4 retains FID reductions of 7.6, 4.8, 4.0, and 4.0 relative to deterministic M_2. It also retains reductions of 4.7, 1.5, 1.3, and 1.6 relative to M_3, which uses a channel-wise perturbation scale. LPIPS remains equal to or lower than both configurations at every evaluated rate.

These results support the interpretation in Section 4.3 that the adaptive perturbation primarily contributes through training regularization rather than through the particular random realization used at inference.

Decoder seed stability. We next vary only the decoder seed according to the protocol in Appendix C.5. Across ten seeds on Kodak, aggregate LPIPS varies by less than 10^{-4} and FID by less than 10^{-2}.

The transmitted indices and payload are identical across these runs. The small metric variation shows that the reported aggregate reconstruction performance is stable across the evaluated GR realizations.

FID repeatability. We also repeat the Kodak FID computation four times with the reconstructed images and patch extraction fixed. All four runs produce identical FID values. This confirms that the variation measured above comes from the GR realization rather than repeated execution of the FID estimator.

E Additional Visual Comparisons

This section provides additional visual comparisons of GR-GIC on all four evaluation datasets. Each figure places comparisons with high-capacity diffusion codecs above those with lightweight baselines and reports LPIPS and BPP for the displayed reconstructions. Two further panels extend the Tecnick and CLIC 2020 coverage under the same LPIPS and BPP annotation protocol.

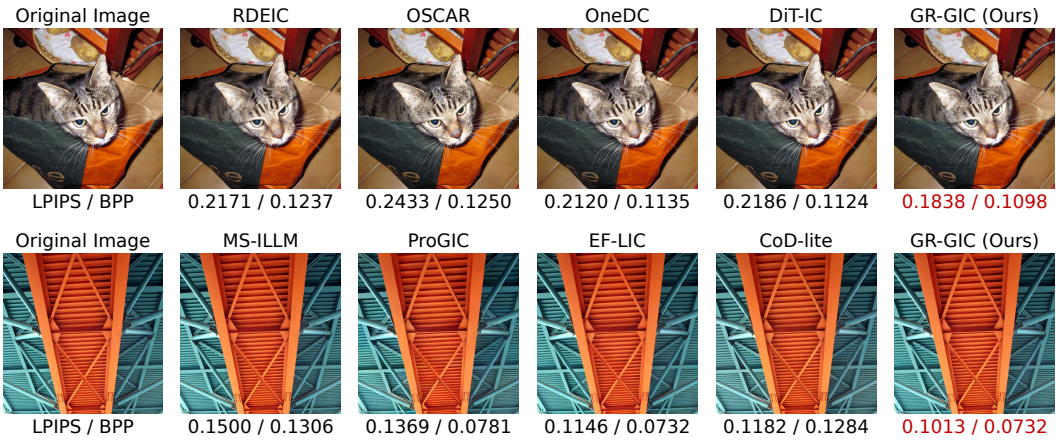


Figure 8: Additional Tecnick visuals with LPIPS and BPP labels, where lower is better.

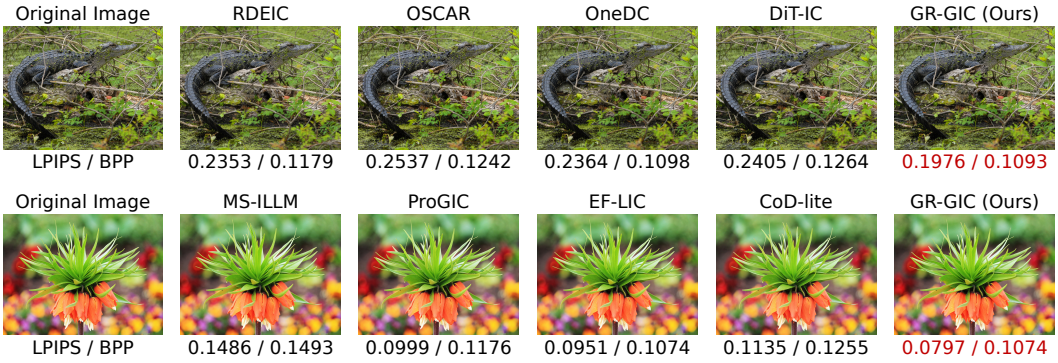


Figure 9: Additional DIV2K visuals with LPIPS and BPP labels, where lower is better.

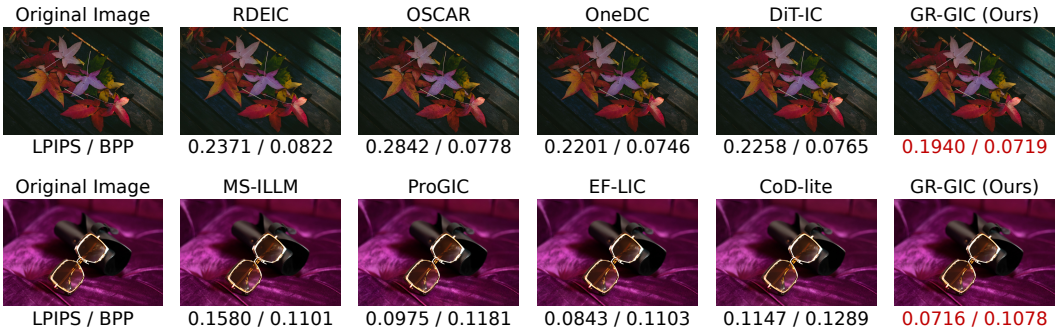


Figure 10: Additional CLIC 2020 visuals with LPIPS and BPP labels, where lower is better.

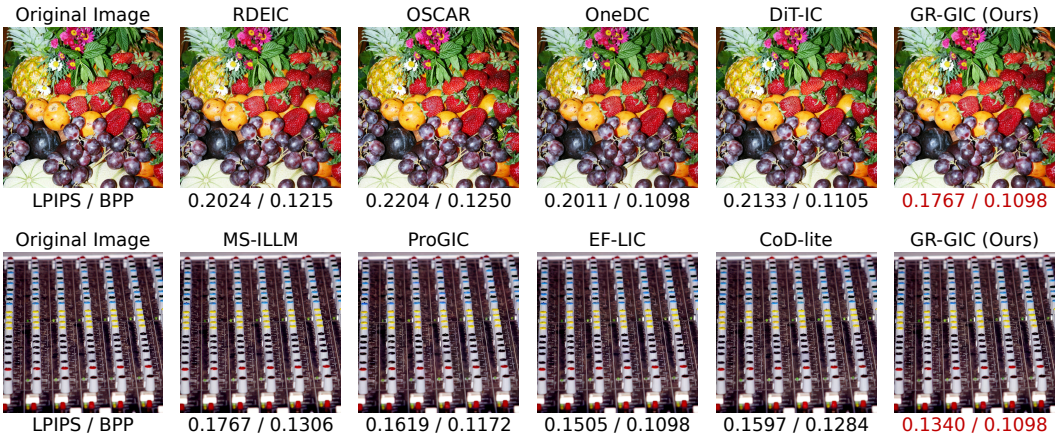


Figure 11: Additional Tecnick visuals with LPIPS and BPP labels, where lower is better.

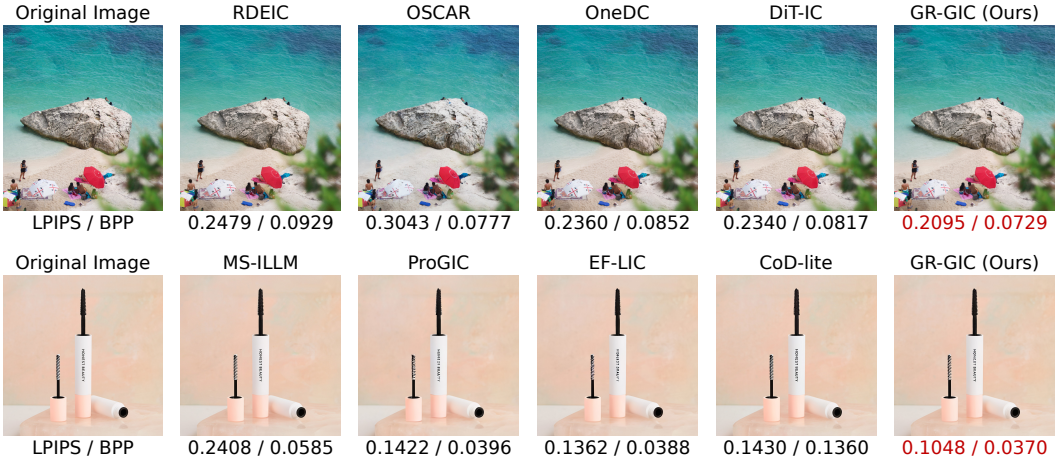


Figure 12: Additional CLIC 2020 visuals with LPIPS and BPP labels, where lower is better.