H-DDCM: HADAMARD CODEBOOKS FOR FAST GENERATIVE IMAGE AND VIDEO COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Training-free diffusion-based compression methods repurpose the generative priors of a diffusion or a flow model to compress data without retraining them as task-specific codecs. Recent codebook-based approaches such as Turbo-DDCM achieve strong perceptual quality at ultra-low bitrates, but their reliance on dense Gaussian codebooks remains costly in both memory and computation, and scales poorly with the large latent dimensions of modern image and video generators. Their combinatorial bitstream encoding further becomes computationally heavy at this scale. We introduce H-DDCM, a scalable generative compression framework that replaces Gaussian codebooks with implicit randomized Hadamard dictionaries. The fast Walsh-Hadamard transform allows for implicitly synthesizing full-dimensional dictionaries with $O(D \log D)$ compute for projection and reconstruction and $O(D)$ memory, without materializing the full-dimensional dictionary. We further introduce an efficient table-free lexicographic coding scheme that makes combinatorial encoding practical at larger dimensions. Beyond images, we adapt the framework to an autoregressive video flow model and develop a streaming compression scheme that periodically re-encodes recently reconstructed frames into a fresh KV-cache prefix, preserving temporal context across segments at no additional transmitted bits. This enables long-form, close to real-time generative video compression on a single GPU. Across image and video benchmarks, H-DDCM achieves competitive rate-distortion-perception performance while substantially reducing the memory and computational overhead of codebook-based training-free generative compression. An anonymous website is available here.

1 INTRODUCTION

Generative priors have become increasingly important for low-bitrate image and video compression, where reconstructing perceptually plausible content can be preferable to preserving every pixel accurately. Early generative codecs were based on GANs (Goodfellow et al., 2020; Agustsson et al., 2019; Mentzer et al., 2020), while more recent methods exploit the stronger priors of diffusion (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2021b) and flow models (Lipman et al., 2023; Liu et al., 2023; Albergo et al., 2025). Most such approaches train or fine-tune the generative model specifically for compression (Yang & Mandt, 2023; Careil et al., 2024; Körber, 2024; Zhang & Wu, 2026; Mao et al., 2026), tying the codec to a particular backbone. In contrast, training-free diffusion compression leaves a pretrained generator frozen and encodes information directly into its sampling trajectory. Denoising Diffusion Codebook Models (DDCM) (Ohayon et al., 2025) realize this idea by replacing the random noise at each denoising step with a noise map selected from a Gaussian codebook shared by the encoder and decoder. The encoder selects the codeword that best aligns with the residual between the target and the model's current clean prediction and transmits only its index; the decoder regenerates the same codeword and thereby follows the encoded denoising trajectory. Turbo-DDCM (Vaisman et al., 2026) substantially accelerates this procedure by selecting and combining multiple codebook atoms in closed form at each step, producing a noise map better aligned with the residual and cutting the expensive denoising steps by over 90%.

Despite this acceleration, Turbo-DDCM remains difficult to scale to the high-dimensional latent spaces of modern generative models. Its Gaussian codebook must either be stored or regenerated, and residual matching requires dense operations whose cost grows with both the latent dimension and codebook size. Restricting the codebook reduces this cost but also limits the set of residual di-

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.

rections available to the encoder. A complete codebook becomes particularly problematic at modern scales, *i.e.*, for large image latents such as FLUX (Black Forest Labs, 2024), it already requires approximately 128 GiB of storage, while video latents such as those used by Causal-Forcing (CF) (Zhu et al., 2026) require roughly 167 GiB. These costs turn the codebook itself into a major computational and memory bottleneck, limiting the applicability of training-free generative compression precisely for the large image and fast video backbones where it is most attractive.

In this work, we address these limitations through three main contributions. First, we replace dense Gaussian codebooks with structured randomized Hadamard dictionaries. Turbo-DDCM only accesses its codebook to correlate candidate atoms with the current residual and to reconstruct the selected combination; both operations can therefore be replaced by fast matrix-vector products. A randomized Hadamard transform provides these operations implicitly, without materializing the dictionary, reducing the computation to $O(D \log D)$ and memory to $O(D)$. Moreover, while Turbo-DDCM relies on the approximate orthogonality of high-dimensional Gaussian atoms, the Hadamard basis is exactly orthogonal, making its top-M candidate thresholding rule optimal for the orthogonal Hadamard representation. We further extend this construction to an overcomplete structured dictionary, combining multiple independently randomized Hadamard bases to expose an even richer set of candidate directions while retaining the fast implicit projections.

Second, we address a separate computational bottleneck in the transmitted bitstream. Turbo-DDCM exploits the fact that the M selected codebook indices form an unordered subset and encodes this subset through a lexicographic rank, reducing the index cost from independently transmitting M indices to $\lceil \log_2 \binom{D}{M} \rceil$ bits. However, its implementation relies on a dense precomputed combinatorial table whose construction and memory requirements become impractical at the higher dimensions considered here. We develop a table-free lexicographic encoder and decoder, together with a grouping protocol that bounds the size of individual combinatorial operations. This retains the bit savings of the combinatorial coding while ensuring practical runtime at large latent dimensions.

Finally, we extend the framework to close to real-time autoregressive (AR) video compression. We adapt CF, a fast causal AR video flow model, into a streaming generative codec. To additionally compress beyond the model's native context length without periodically discarding temporal context, we introduce a re-encoded-tail cache refresh: at each continuation boundary, recently reconstructed frames are re-encoded into a fresh KV-cache prefix, providing the next segment with temporal context at no additional transmitted bits. Together, these components enable scalable streaming compression on a single GPU. On images, our structured dictionary matches the fidelity of complete-codebook Turbo-DDCM without materializing the corresponding codebook; on video, our streaming codec achieves competitive rate-distortion-perception performance while providing the highest measured throughput among the evaluated methods sharing the CF backbone.

2 Related work

Neural and generative compression. Learned codecs replace the transform and entropy model of classical compression with trained networks (Ballé et al., 2017; Toderici et al., 2017). These approaches were extended from images to video via learned motion estimation and residual coding (Lu et al., 2019; Li et al., 2023). The original methods targeted low distortion, but tended to produce over-smoothed reconstructions at low bitrates. Later generative codecs traded exact fidelity for plausibility, first with GANs (Agustsson et al., 2019; Mentzer et al., 2020) and more recently with diffusion priors trained from scratch (Yang & Mandt, 2023; Ghouse et al., 2023), fine-tuned for compression (Careil et al., 2024; Körber, 2024), or distilled into a single reverse step (Park et al., 2026; Xue et al., 2026). This framework has extended to video, where a pretrained video diffusion model refines the output of a separately designed codec (Mao et al., 2026). All of these require training or fine-tuning dedicated to compression, which ties the codec to a particular backbone and generally requires retraining when the backbone changes.

Training-free diffusion-based image compression. Training-free methods instead leave the generative model frozen and encode information through its sampling trajectory. DiffC (Theis et al., 2022) communicates samples of the forward process through reverse channel coding, which Vonderfecht & Liu (2025) made practical with a custom CUDA kernel. IPIC (Xu et al., 2024) builds on posterior sampling, while PSC (Elata et al., 2025) draws on compressed sensing (Donoho, 2006). DDCM (Ohayon et al., 2025) replaces the Gaussian noise at each reverse step with an atom selected from a reproducible codebook, and Turbo-DDCM (Vaisman et al., 2026) combines multiple atoms

per step in closed form, reducing the number of expensive denoising evaluations by over 90%. Our method belongs to this training-free setting, but replaces the explicit Gaussian codebooks with an implicit structured dictionary.

Training-free generative video compression. Carrying the idea to video raises a difficulty absent from images, since temporal coherence is a property of the joint distribution across frames rather than of each frame separately. ZeroGVC (Gao et al., 2026) encodes the first frame of every group of pictures with an image codec and the remaining P-frames by codebook-guided AR latent compression, steering a few-step consistency trajectory toward each target frame, with an optional bidirectional mode that borrows context from the next I-frame at no extra rate. GVCC (Zeng et al., 2026) observes that modern video foundation models are sampled by a deterministic rectified-flow ODE, which leaves no per-step stochastic channel to encode into, and restores one by converting the sampler into an equivalent marginal-preserving SDE. Both retain an explicit Gaussian codebook, and video is where that choice is most expensive: a latent chunk is larger than an image latent, and the codebook grows with the product of its size and that dimension (App. F).

Autoregressive and streaming video diffusion. Recent advances in video generation increasingly utilize causal and AR formulations to support real-time generation. Diffusion Forcing (Chen et al., 2024) assigns independent noise levels per frame and thereby supports AR rollout, while CausVid (Yin et al., 2025) and Self-Forcing (Huang et al., 2026) distill bidirectional video diffusion models into few-step causal students. Addressing structural limits in this distillation, Zhu et al. (2026) demonstrate that the ODE-initialization stage requires frame-level injectivity – a condition violated by bidirectional teachers, and resolve this by distilling from an AR teacher, thereby achieving practical real-time streaming generation on a single GPU.

3 Background

Our method builds directly on Turbo-DDCM, which compresses data by replacing the random noise along a diffusion trajectory with combinations of atoms from a shared Gaussian codebook. We first review this formulation and its sparse noise-selection rule, and highlight the computational, memory and bitstream bottlenecks that emerge as latent dimension grows. We then introduce the randomized Hadamard transform and the properties that allow us to replace these dense codebook operations with fast implicit projections.

3.1 Denoising diffusion models

A diffusion model (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2021b) generates data by reversing a gradual noising process. Starting from $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$, the reverse step is given by

$$\mathbf{x}_{t-1} = \boldsymbol{\mu}_t(\mathbf{x}_t) + \sigma_t \mathbf{z}_t, \qquad \mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d), \tag{1}$$

Here, $\boldsymbol{\mu}_t$ is determined by the model's prediction at step t, while $\mathbf{z}_t$ provides the stochastic degree of freedom in the transition. Its exact parameterization depends on the sampler: we use DDPM sampling for images and a few-step consistency formulation for our causal video backbone, whose corresponding updates are detailed in App. D. DDCM exploits precisely this freedom in $\mathbf{z}_t$, replacing randomly sampled noise with information-carrying noise maps selected by the encoder.

3.2 Turbo-DDCM

DDCM (Ohayon et al., 2025) replaces the random noise in (1) with noise maps selected from a reproducible codebook $\boldsymbol{C}_t \in \mathbb{R}^{d \times K}$. Its columns $\mathbf{z}_t^{(1)}, \ldots, \mathbf{z}_t^{(K)}$ are i.i.d. Gaussian atoms generated from a seed shared by the encoder and decoder. At each step, the encoder selects the atom most aligned with the residual $\mathbf{r}_t := \mathbf{x}_0 - \hat{\mathbf{x}}_{0|t}$, where $\hat{\mathbf{x}}_{0|t}$ is the model's current clean prediction. Injecting a noise map correlated with $\mathbf{r}_t$ steers the sampling trajectory toward the target, while transmitting only its codebook index allows the decoder to reproduce the same trajectory.

Selecting a single atom provides limited residual correction per denoising step. Turbo-DDCM (Vaisman et al., 2026) instead approximates $\mathbf{r}_t$ using a sparse combination of M codebook atoms, allowing substantially more information to be transmitted per step, therefore reducing the number of expensive denoiser evaluations. Writing $\boldsymbol{\alpha}_t \in \mathbb{R}^K$ for the coefficient vector, the per-step problem is

$$\boldsymbol{\alpha}_t^\star = \underset{\boldsymbol{\alpha} \in \mathbb{R}^K}{\arg\min} \left\| \boldsymbol{C}_t \boldsymbol{\alpha} - \mathbf{r}_t \right\|_2^2 \quad \text{s.t.} \quad \|\boldsymbol{\alpha}\|_0 = M, \quad \alpha_i \in \mathcal{V} \cup \{0\} \ \ \forall i, \tag{2}$$

with $\mathcal{V}$ being a quantization set of 2^C levels, so each nonzero coefficient costs C bits. Problem (2) is combinatorial for a general dictionary. However, if the dictionary has orthonormal columns, it separates across coordinates and is solved exactly by thresholding its correlations with the residual (Elad, 2010). Gaussian atoms are not orthogonal, but become approximately so in high dimensions; Turbo-DDCM therefore applies the same thresholding rule. For the one-bit case $\mathcal{V} = \{-1, +1\}$,

$$(\boldsymbol{\alpha}_t^\star)_i = \begin{cases} \operatorname{sign}\big((\mathbf{u}_t)_i\big), & i \in \text{Top-M}(|\mathbf{u}_t|), \\ 0, & \text{otherwise}, \end{cases} \qquad \mathbf{u}_t = \boldsymbol{C}_t^\top \mathbf{r}_t. \tag{3}$$

For an orthogonal dictionary, this is exactly the minimizer of (2): the support collects the M largest correlations, and each selected coefficient is rounded to the closest value in $\mathcal{V}$. The resulting noise map is normalized, and injected in place of $\mathbf{z}_t$ in (1)

$$\mathbf{z}_t^\star = \frac{\boldsymbol{C}_t \boldsymbol{\alpha}_t^\star}{\operatorname{std}(\boldsymbol{C}_t \boldsymbol{\alpha}_t^\star)}. \tag{4}$$

Turbo-DDCM further reduces bitrate by terminating codebook selection early and completing the trajectory with $N+1$ deterministic DDIM steps (Song et al., 2021a). It also exploits the fact that the M selected indices form an unordered subset: rather than transmitting them independently, it sends their lexicographic rank using $\lceil \log_2 \binom{K}{M} \rceil + MC$ bits per coded step. This avoids the permutation redundancy of independently listing the selected indices.

Despite these improvements, two computational bottlenecks remain and become severe at large latent dimensions. First, computing $\boldsymbol{C}_t^\top \mathbf{r}_t$ and reconstructing $\boldsymbol{C}_t \boldsymbol{\alpha}_t^\star$ require access to a dense $d \times K$ Gaussian codebook, whose memory and computation grow with both d and K. Practical implementations must therefore restrict K as the latent dimension grows, reducing the set of directions available to represent the residual. Second, Turbo-DDCM's lexicographic rank and unrank operations rely on a dense precomputed table of binomial coefficients for each (K, M) pair, which becomes expensive to construct and store at large K. Our method targets both scaling bottlenecks directly.

3.3 Randomized Hadamard transform

The dense-codebook bottleneck above motivates a dictionary that is both orthogonal and admits fast implicit matrix-vector products. Let D be a power of two and $\mathbf{H} \in \mathbb{R}^{D \times D}$ denote the normalized Hadamard matrix, via Sylvester's recursion and scaled by $D^{-1/2}$. Its entries are $\pm D^{-1/2}$ and

$$\mathbf{H} = \mathbf{H}^\top, \qquad \mathbf{H}^\top \mathbf{H} = \mathbf{I}_D, \qquad \mathbf{H}^2 = \mathbf{I}_D. \tag{5}$$

Thus, the columns of $\mathbf{H}$ form an orthonormal basis of $\mathbb{R}^D$. Crucially, this basis never needs to be materialized: the fast Walsh-Hadamard transform (FWHT) (Fino & Algazi, 1976) computes $\mathbf{H}\mathbf{v}$, *i.e.*, the correlations of $\mathbf{v}$ with all Hadamard basis vectors simultaneously, using $O(D \log D)$ computation and $O(D)$ memory.

To randomize the basis, let $\boldsymbol{\varepsilon} \in \{-1, +1\}^D$ contain i.i.d. Rademacher samples, define $\mathbf{D} = \operatorname{diag}(\boldsymbol{\varepsilon})$, and consider the dictionary $\mathbf{DH}$ rather than $\mathbf{H}$. On the analysis side, multiplying a vector by the transposed matrix $\mathbf{HD}$ uniformly disperses its energy across all coordinates and reduces coherence with structured inputs (Tropp, 2011). Since $\mathbf{D}^\top \mathbf{D} = \mathbf{I}_D$,

$$(\mathbf{DH})^\top (\mathbf{DH}) = \mathbf{H}^\top \mathbf{D}^\top \mathbf{DH} = \mathbf{H}^\top \mathbf{H} = \mathbf{I}_D, \tag{6}$$

so that the randomized basis $\mathbf{DH}$ remains exactly orthonormal, while multiplication by either $\mathbf{DH}$ or its transpose requires only one FWHT and sign modulation. It can consequently be regenerated from a shared seed and applied with $O(D \log D)$ computation and $O(D)$ memory, without storing any of its D^2 entries.

4 Method

We address both scaling bottlenecks from Sec. 3: the dense Gaussian codebook and the combinatorial bitstream computation cost. First, we replace the Gaussian codebook with an implicit randomized Hadamard dictionary, enabling fast residual projection and reconstruction without materializing the dictionary (Sec. 4.1). Then, we introduce a scalable table-free, grouped lexicographic bitstream implementation (Sec. 4.2). Finally, we extend the structured dictionary with multiple randomized Hadamard bases to increase the available candidate directions in the low-rate regime (Sec. 4.3).

4.1 STRUCTURED HADAMARD DICTIONARY

Turbo-DDCM accesses its codebook $\boldsymbol{C}_t$ in two places – forming the correlations $\mathbf{u}_t = \boldsymbol{C}_t^\top \mathbf{r}_t$ and reconstructing $\boldsymbol{C}_t \boldsymbol{\alpha}_t^\star$. Both require only multiplication by the codebook and its transpose, which the Hadamard transform (Sec. 3.3) provides implicitly without materializing the dictionary.

For $\mathbf{r}_t \in \mathbb{R}^d$, we choose the smallest supported Hadamard dimension $D \geq d$ and zero-pad $\mathbf{r}_t$ to D coordinates; the reconstructed noise is cropped back to the original dimensions. We henceforth overload $\mathbf{r}_t$ to denote its zero-padded representation. At denoising step t, we generate a Rademacher diagonal matrix $\mathbf{D}_t$ from the seed shared by the encoder and decoder and define

$$\mathbf{A}_t := \mathbf{D}_t \mathbf{H} \in \mathbb{R}^{D \times D} \tag{7}$$

whose columns form an orthonormal basis by (6). A new $\mathbf{D}_t$ is generated at every step, providing a different randomized basis along the trajectory without storing or transmitting the dictionary. We approximate $\mathbf{r}_t$ in this basis by

$$\boldsymbol{\alpha}_t^\star = \underset{\boldsymbol{\alpha} \in \mathbb{R}^D}{\arg\min} \left\| \mathbf{r}_t - \mathbf{A}_t \boldsymbol{\alpha} \right\|_2^2 \quad \text{s.t.} \quad \|\boldsymbol{\alpha}\|_0 = M, \quad \alpha_i \in \mathcal{V} \cup \{0\} \ \ \forall i, \tag{8}$$

which is the sparse least-squares equation from (2) but with the Gaussian codebooks $\boldsymbol{C}_t$ replaced by randomized Hadamard entries $\mathbf{A}_t$, with $K = D$ (complete basis). Because $\mathbf{A}_t$ is an orthonormal basis, the objective decouples coordinate-wise via

$$\left\| \mathbf{r}_t - \mathbf{A}_t \boldsymbol{\alpha} \right\|_2^2 = \left\| \mathbf{A}_t (\mathbf{u}_t - \boldsymbol{\alpha}) \right\|_2^2 = \left\| \mathbf{u}_t - \boldsymbol{\alpha} \right\|_2^2, \qquad \mathbf{u}_t = \mathbf{A}_t^\top \mathbf{r}_t. \tag{9}$$

Consequently, the thresholding rule of (3), which was only an approximation for the nearly orthogonal Gaussian atoms of Turbo-DDCM, solves (8) exactly: the support is $\mathcal{I}_t := \text{Top-M}(|\mathbf{u}_t|)$ and each selected entry is rounded to the nearest level in $\mathcal{V}$ (Vaisman et al., 2026). The selected combination is rescaled and injected in place of $\mathbf{z}_t$ in (1), by substituting $\mathbf{A}_t$ instead of $\boldsymbol{C}_t$ in (4). Encoding and decoding algorithms are provided in App. H.

The encoder requires one FWHT and top-M selection; the decoder requires one FWHT to reconstruct the selected noise. Both run in $O(D \log D)$ time and $O(D)$ memory without storing a dictionary, compared with the $O(Kd)$ Gaussian codebook of Turbo-DDCM (App. F). Note that although residual-dependent selection means the resulting noise is not itself Gaussian, the same is true of Turbo-DDCM's selected Gaussian combinations – Sec. 5 evaluates reconstruction quality directly.

4.2 BITSTREAM OPTIMIZATION

Turbo-DDCM encodes the M selected indices through their lexicographic rank, requiring $\lceil \log_2 \binom{K}{M} \rceil$ bits for the selected index set. However, its implementation relies on a large precomputed table of binomial coefficients whose memory and construction costs scale poorly with dictionary size K, becoming impractical in high-dimensional regimes (we refer to App. E for further discussion). Instead, we propose a table-free lexicographic coding scheme that is tractable at large dimensions, while preserving the same combinatorial coding principle.

We propose two table-free mechanisms (App. E): (i) an incremental recurrence advances the required binomial coefficients using exact Pascal-rule updates, avoiding their repeated computation and storage, and (ii) a hockey-stick-identity (Jones, 1996) formulation expresses each selected index's contribution using directly evaluated binomial coefficients and unranks via binary search. The former is effective for dense selections, while the latter is particularly efficient when $M \ll D$.

Furthermore, for a dictionary of size K, we divide the candidate atoms into G groups of size $K_G := K/G$ and encode each group independently. We first transmit the composition $(m_1, \ldots, m_G)$ of the M selected indices using a stars-and-bars code (Feller, 1950), followed by the local lexicographic rank and coefficient bits within each group. This trades a small rate overhead for G smaller combinatorial operations that can be evaluated independently and in parallel. The resulting bitrate is

$$\text{BPP} = \frac{S\Big(L_{\text{counts}} + \sum_{g=1}^{G} \Big(\Big\lceil \log_2 \binom{K_G}{m_g} \Big\rceil + m_g C \Big)\Big)}{\text{number of pixels}}, \qquad L_{\text{counts}} = \Big\lceil \log_2 \binom{M+G-1}{G-1} \Big\rceil, \tag{10}$$

where $\sum_{g=1}^{G} m_g = M$ and S is the number of bit-carrying sampling steps, which is $T - N - 1$ for DDPM and T for consistency-models (see App. D). The single Hadamard dictionary has $K = D$ atoms, giving $K_G = D/G$. Setting $G = 1$ recovers a single flat lexicographic rank.

4.3 Extension: overcomplete Hadamard dictionary

At small M, *i.e.*, in the low-bitrate regime, reconstruction quality becomes particularly sensitive to the alignment between $\mathbf{r}_t$ and the available basis directions. We therefore extend the single Hadamard basis into an overcomplete dictionary by drawing B independent Rademacher diagonals $\mathbf{D}_{t,1}, \ldots, \mathbf{D}_{t,B}$ from the shared seed. This gives B independently randomized orthogonal bases $\mathbf{A}_{t,b} := \mathbf{D}_{t,b}\mathbf{H}$, which we concatenate as

$$\mathbf{A}_t^{\mathrm{oc}} := [\mathbf{A}_{t,1} \ \cdots \ \mathbf{A}_{t,B}] \in \mathbb{R}^{D \times BD}. \tag{11}$$

The encoder can therefore select from BD candidate directions rather than D of a single basis.

Each $\mathbf{A}_{t,b}$ is individually orthogonal by (6), but atoms belonging to different randomized bases are not mutually orthogonal to each other: $\mathbf{A}_{t,b}^\top \mathbf{A}_{t,b'}$ is a dense matrix for $b \neq b'$, not the identity. Global top-M thresholding is therefore no longer an exact solution of (8); instead, we use the matched-filter approximation also employed by Turbo-DDCM for its approximately orthogonal Gaussian atoms (Sec. 3.2). Each basis contributes its own scores $\mathbf{u}_{t,b} := \mathbf{A}_{t,b}^\top \mathbf{r}_t$, at the cost of one FWHT, and the encoder selects the M largest-magnitude coefficients across all B score vectors. On the synthesis side, once the support is selected, the least-squares coefficients on the cross-basis support can be recovered through a pseudo-inverse. In practice, however, we find that directly using the matched-filter coefficients produces an almost identical noise direction (the comparison is provided in App. C.6).

An index now identifies one of BD atoms, which we encode using the grouped protocol of Sec. 4.2 by treating each randomized basis as an individual group ($G = B$). Because each of the B bases contains D atoms, the per-group size matches the basis dimension ($K_G = D$). The richer dictionary therefore incurs additional transform and index costs; at a fixed bit budget the extension must select fewer atoms than H-DDCM. Appendix G gives the construction in full and an exact and efficient way to compute the global top-M.

5 Application to image compression

At high latent dimensions, Turbo-DDCM faces a practical trade-off – a complete codebook provides the full set of candidate directions but incurs large memory and computational costs, while capping the codebook reduces these costs at the expense of fidelity due to restricted subspace span. Here, we demonstrate that H-DDCM avoids this trade-off at FLUX scale ($d = 2^{18}$), matching the quality of the complete codebook while retaining the efficiency of the capped variant.

Implementation details. We evaluate two backbones: SD 2.1 at 512×512 resolution, where the latent dimension is $d = 2^{14}$, and FLUX at 1024×1024 resolution, where $d = 2^{18}$. For both, we run $T = 30$ steps, $C = 1$, $G = 1$ groups for SD 2.1, $G = 8$ for FLUX, and sweep M to trace out a full rate-distortion-perception (Blau & Michaeli, 2019) curve – 22 points for SD 2.1, 14 for FLUX. Appendix A lists additional implementation details and every method's exact hyperparameters.

Evaluation. For SD 2.1 we report on Kodak24 and additionally evaluate DIV2K and CLIC2020 in App. C; all three datasets are resized and center-cropped to 512×512. For FLUX we evaluate on DIV2K, resized and center-cropped to 1024×1024. We report round-trip (compression and decompression) time for every method on a single NVIDIA A6000 GPU. Distortion is measured by PSNR and perceptual quality by FID (Heusel et al., 2017), computed on 64×64 patches following Mentzer et al. (2020). For training-free diffusion-based methods, we further report the incremental memory required for storage and computations, as well as disk storage required by Turbo-DDCM's precomputed bitstream tables. Appendix C additionally reports SSIM (Wang et al., 2004), LPIPS (Zhang et al., 2018), and separate encoding and decoding times.

Competing methods. We compare against every training-free diffusion-based codec that shares our exact backbone and requires no training: DiffC (Theis et al., 2022; Vonderfecht & Liu, 2025), DDCM, Turbo-DDCM utilizing a capped codebook ($K = 2^{14}$) as well as a complete one ($K = d$), and PSC (Elata et al., 2025) in its distortion- and perception-oriented variants (PSC-D, PSC-P). We additionally compare against codecs trained or fine-tuned specifically for compression – DiffEIC (Li et al., 2024), PerCo (Körber, 2024), ILLM (Muckley et al., 2023), CRDR (Iwai et al., 2024) (its -D and -R variants), StableCodec (Zhang et al., 2025), and HiFiC (Mentzer et al., 2020). We also compare against the classical codec BPG (Bellard, 2018). For FLUX we compare to training-free methods only: DiffC and Turbo-DDCM using both codebook sizes. Note that PSC is omitted from the runtime comparisons (Figs. 1 and S2) due to its slow runtime.

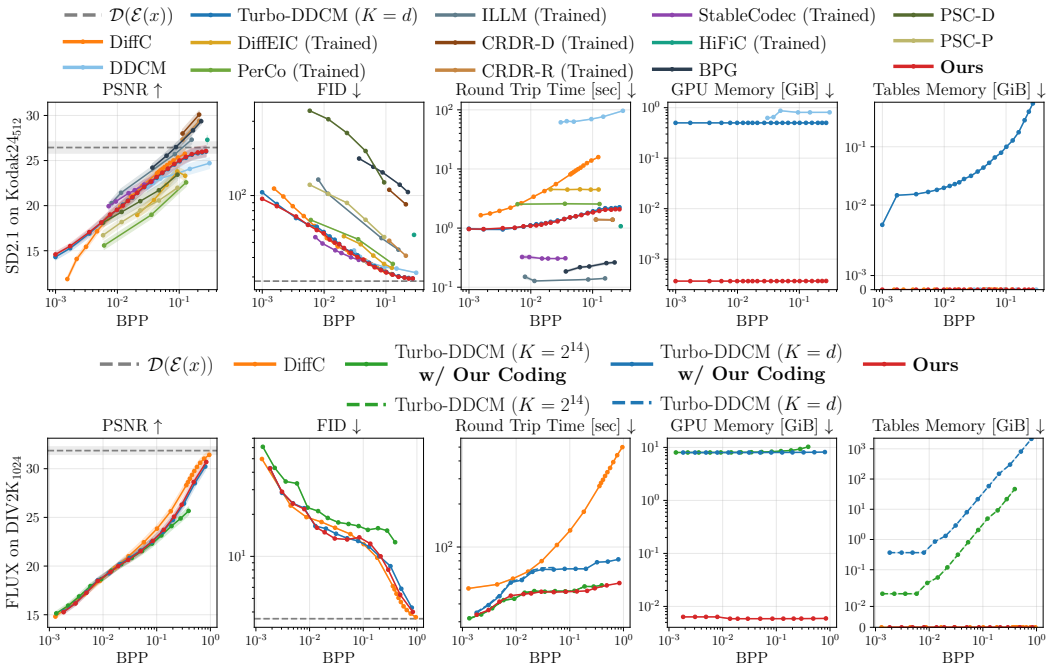

Figure 1: **Quantitative comparison of image compression methods.** We compare H-DDCM with training-free diffusion-based and other compression methods in terms of rate, distortion, and perception. Distortion is measured by PSNR, perceptual quality by FID (presented in log-scale), and runtime by round-trip encoding and decoding time. GPU memory denotes incremental memory used by training-free methods for storage and computation, while tables memory refers to disk storage required by Turbo-DDCM's precomputed combinatorial tables. The first row corresponds to training-free methods utilizing the SD 2.1 backbone (512×512, $d = 2^{14}$) and other compression methods. The second row corresponds to FLUX backbone (1024×1024, $d = 2^{18}$). The dashed gray vertical line denotes the VAE encoder-decoder bound. H-DDCM achieves competitive results on the rate-distortion-perception tradeoff, all the while being the fastest training-free method, avoiding explicit codebook materialization. Error bars represent the standard error of the mean (SEM).

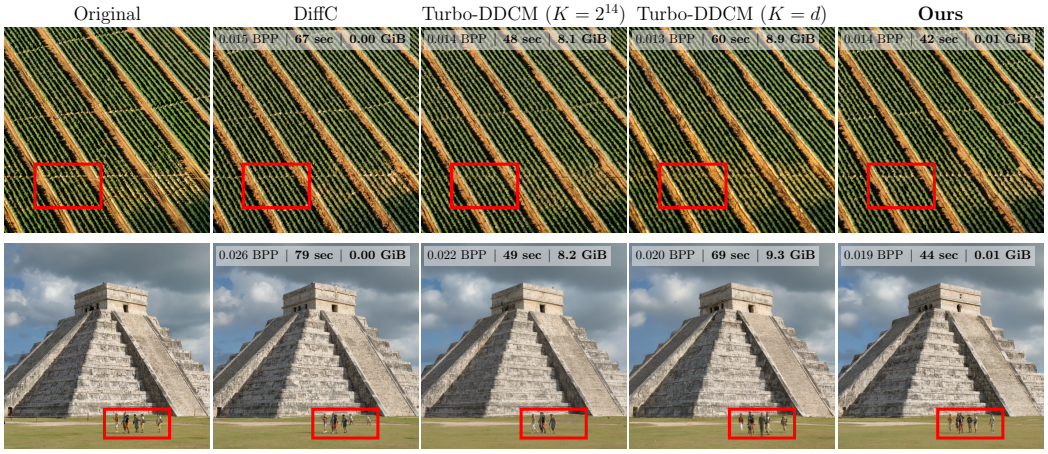

Figure 2: **Qualitative comparison of training-free diffusion-based image compression methods (FLUX on DIV2K).** H-DDCM is the fastest and preserves visual quality and fidelity to the original image, while avoiding the memory overhead of an explicit codebook.

Results. Figure 1 shows that H-DDCM closely matches complete-codebook Turbo-DDCM ($K = d$) and DiffC in terms of PSNR and FID across the evaluated bitrate regime. On FLUX, capping Turbo-DDCM's codebook below the latent dimension d reduces quality, while using the complete codebook substantially increases memory and runtime. At matched bitrate on SD 2.1, H-DDCM

and Turbo-DDCM ($K = d$) both achieve a 1.3 s round trip, compared with 5.5 s for DiffC and 61 s for DDCM. At FLUX scale, H-DDCM reduces the round trip time from 66 s for Turbo-DDCM ($K = d$) to 44 s, while avoiding explicit codebook materialization. These results show that the structured dictionary retains the quality benefit of a complete codebook while scaling more favorably in memory and computation. Figure 2 provides corresponding FLUX reconstructions, including the round trip time and the total required memory for each method (GPU and tables memory), and App. C reports additional datasets and metrics.

6 Application to Streaming Video Compression

Beyond the high-dimensional setting considered for images, causal AR video compression introduces two additional challenges: codec throughput must keep pace with generation, and compression must continue beyond the backbone's finite training context without losing temporal consistency. We adapt our framework to the CF AR video backbone and address the latter challenge with a re-encoded-tail KV-cache refresh, enabling long-form, close to real-time video compression while retaining context across segment boundaries.

Autoregressive video compression. We evaluate our method using CF at 832×480 resolution. We apply the residual-coding procedure to each causal latent chunk, using CF's consistency trajectory in place of the DDPM trajectory used for images. We run $T = 4$ consistency steps and use the $B = 8$ overcomplete (OC) dictionary (Sec. 4.3). Unlike DDPM, in the consistency formulation we choose to use no deterministic tail ($N = 0$), so all T steps perform noise selection and carry bits. We use our efficient lexicographic implementation for all applicable CF-based baselines to isolate codec overhead. Appendix D describes the consistency formulation and initial-step proxy residual.

Re-encoded-tail KV-cache refresh. CF is trained on contexts of 21 latent frames (81 pixel frames), corresponding to 7 chunks of 3 latent frames each. Extending the AR cache beyond this regime leads to severe degradation, while simply resetting the cache at each continuation boundary discards temporal context and introduces discontinuities (see ablation in App. C). We therefore refresh the cache at every segment boundary by re-encoding the last 9 reconstructed pixel frames into a fresh KV-cache prefix. Since these reconstructed frames are available identically to both encoder and decoder, the refreshed prefix requires no additional transmitted bits. The next segment is then compressed conditioned on this prefix, preserving recent temporal context while keeping the cache within the model's training regime. The first newly coded chunk after a refresh is particularly difficult because it has the least generated context available. We therefore use a larger selection budget $M_{\text{start}} \geq M$ for this chunk, after which compression returns to the budget of M indices per step. Appendix B provides additional details of the re-encoded-tail mechanism, M_{start} values, and the remaining video hyperparameters.

Evaluation. We evaluate on UVG (Mercat et al., 2020) and MCL-JCV (Wang et al., 2016), resized and center-cropped to 832×480, and in App. C we further evaluate HEVC Class B/C/D (Bossen, 2010). We measure distortion by PSNR, and perceptual quality with DOVER (Wu et al., 2023) and FVMD (Liu et al., 2024); additional metrics are reported in App. C. Encoding and decoding throughput are measured on a single NVIDIA H200 GPU and compared directly among methods leveraging the same CF backbone.

Competing methods. We compare against the classical codecs HEVC (Sullivan et al., 2012) and VVC (Bross et al., 2021), the recent neural codec DCVC-UF (Li et al., 2026), and training-free generative compression methods using the same CF backbone: ZeroGVC (Gao et al., 2026), and Turbo-DDCM with both a capped codebook ($K = 2^{14}$) and a complete one ($K = d$). As no official CF implementation is available for ZeroGVC or Turbo-DDCM, we implement these baselines following their respective formulations.

Results. Figure 3 presents the quantitative results. HEVC, VVC, and DCVC-UF achieve higher PSNR but substantially weaker perceptual quality as measured by DOVER and FVMD. Among the training-free methods sharing the CF backbone, ZeroGVC achieves comparable PSNR but weaker temporal and motion consistency as captured by FVMD, together with lower throughput. Both Turbo-DDCM variants achieve lower PSNR and perceptual quality, while the complete-codebook variant additionally incurs substantially lower throughput. In contrast, H-DDCM provides competitive rate-distortion-perception performance while achieving the highest measured encoding and decoding throughput among the evaluated CF-based methods. Figure 4 illustrates the corresponding

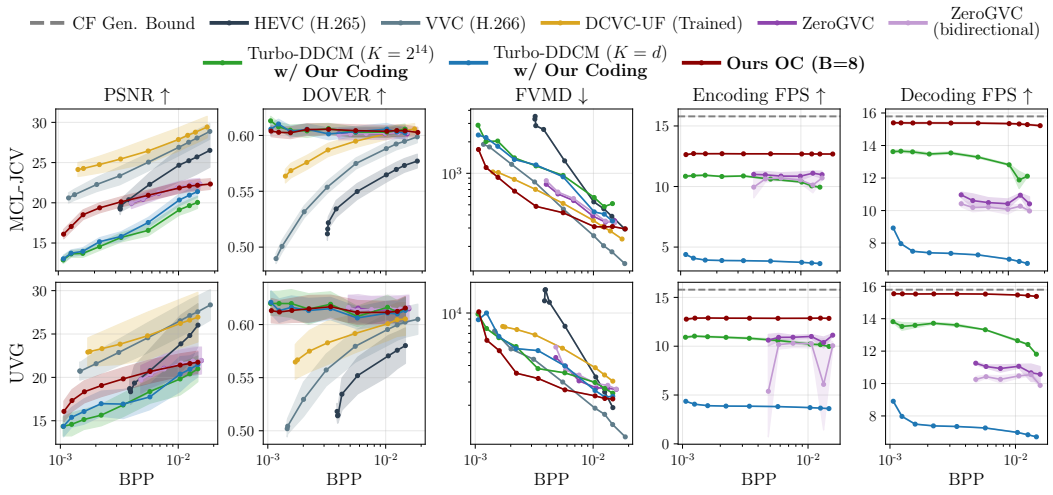

Figure 3: **Quantitative comparison of video compression methods.** We compare H-DDCM OC ($B = 8$) with training-free methods using the same CF backbone and other compression baselines in terms of rate, distortion and perception. Distortion is measured by PSNR, and perceptual quality is measured by DOVER and FVMD (presented in log-scale). Encoding and decoding FPS report compression and decompression throughput, and are compared among methods using the CF backbone. H-DDCM achieves competitive rate-distortion-perception performance among the CF-based training-free methods while providing the highest measured encoding and decoding throughput, without codebooks materialization. Error bars represent the standard error of the mean (SEM).

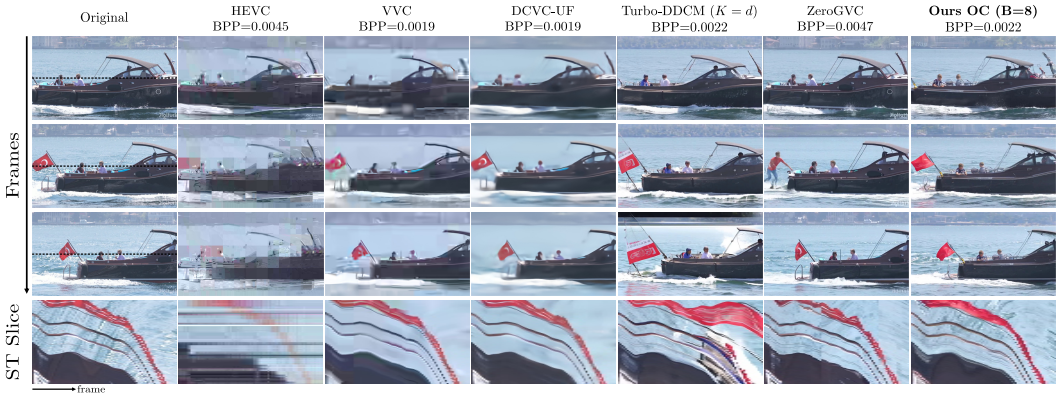

Figure 4: **Qualitative comparison of video-compression methods.** The example is taken from the UVG dataset at 832×480 resolution. Our method preserves source content and temporal consistency while providing the highest measured throughput among the evaluated CF-based methods. The ST slice highlights temporal discontinuities at ZeroGVC's I-frame boundaries.

visual and temporal behavior, including discontinuities at segment boundaries in ZeroGVC. Additional quantitative and qualitative results are provided in App. C.

7 Conclusion

We introduced H-DDCM, a training-free generative codec that replaces Turbo-DDCM's dense Gaussian codebooks with implicit randomized Hadamard dictionaries and its precomputed lexicographic table with scalable table-free, group-factorized coding. This enables efficient compression at large latent dimensions: on FLUX, H-DDCM matches the fidelity of a complete Turbo-DDCM codebook without materializing it, while retaining the runtime efficiency of a capped codebook. For video, we combine an overcomplete Hadamard dictionary with a re-encoded-tail KV-cache refresh to adapt the framework to the CF AR backbone, enabling long-form, close to real-time streaming compression. H-DDCM achieves competitive rate-distortion-perception performance while providing the highest measured throughput among the evaluated methods sharing the CF backbone.

AI USE STATEMENT

Generative AI tools were used in this paper for minor text polishing to improve readability, and for editing software code. We have not used AI to: generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods from scratch, assist with translation, clean and reformat dataset, support qualitative and thematic data analysis, and interpret results. Additionally, we used generative AI tools to create and modify figures, to improve the style. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work builds upon pre-trained generative models, and thus inherits the broader ethical considerations associated with their use. Such models may reflect or amplify societal biases present in the training data, and their outputs could be misinterpreted or misused in sensitive applications. In addition, our approach involves large-scale diffusion and consistency models, which carry the potential risk of being repurposed for harmful or malicious purposes. We emphasize that our contributions are intended solely for advancing research in generative modeling.

REPRODUCIBILITY STATEMENT

Our results can be fully reproduced by implementing the method as described in Sec. 4, and summarized as pseudo-code in App. H, with the hyperparameters as provided in Apps. A and B, and the bitstream implementation details as provided in App. E. Code will be released upon acceptance.

REFERENCES

Eirikur Agustsson, Michael Tschannen, Fabian Mentzer, Radu Timofte, and Luc Van Gool. Generative adversarial networks for extreme learned image compression. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 221–231, 2019. 1, 2

Michael Albergo, Nicholas M Boffi, and Eric Vanden-Eijnden. Stochastic interpolants: A unifying framework for flows and diffusions. *Journal of Machine Learning Research*, 26(209):1–80, 2025. 1

Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. End-to-end optimized image compression. In *International Conference on Learning Representations*, 2017. 2

Fabrice Bellard. Bpg image format, 2018. URL https://bellard.org/bpg/. 6

Black Forest Labs. Flux. https://github.com/black-forest-labs/flux, 2024. 2

Yochai Blau and Tomer Michaeli. Rethinking lossy compression: The rate-distortion-perception tradeoff. In *International Conference on Machine Learning*, pp. 675–685. PMLR, 2019. 6

Frank Bossen. Common test conditions and software reference configurations. In *3rd. JCT-VC Meeting, Guangzhou, CN, October 2010*, 2010. 8

Benjamin Bross, Ye-Kui Wang, Yan Ye, Shan Liu, Jianle Chen, Gary J Sullivan, and Jens-Rainer Ohm. Overview of the versatile video coding (vvc) standard and its applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10):3736–3764, 2021. 8

Marlene Careil, Matthew J Muckley, Jakob Verbeek, and Stéphane Lathuilière. Towards image compression with perfect realism at ultra-low bitrates. In *International conference on learning representations*, volume 2024, pp. 15544–15564, 2024. 1, 2

Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2024. 3

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, 2020. 21

David L Donoho. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006. 2

Michael Elad. *Sparse and redundant representations: from theory to applications in signal and image processing*, volume 2. springer New York, 2010. 4

Noam Elata, Tomer Michaeli, and Michael Elad. PSC: Posterior sampling-based compression. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. 2, 6

William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. John Wiley & Sons, 1950. 5

Fino and Algazi. Unified matrix treatment of the fast walsh-hadamard transform. *IEEE Transactions on Computers*, 100(11):1142–1146, 1976. 4

Yixin Gao, Xiaohan Pan, Lin Liu, Xin Li, Zhibo Chen, and Qi Tian. Zerogvc: Zero-shot generative video compression with autoregressive diffusion priors. *arXiv preprint arXiv:2606.22371*, 2026. 3, 8

Noor Fathima Ghouse, Jens Petersen, Auke Wiggers, Tianlin Xu, and Guillaume Sautiere. A residual diffusion model for high perceptual quality codec augmentation. *arXiv preprint arXiv:2301.05489*, 2023. 2

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020. 1

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. *Advances in neural information processing systems*, 30, 2017. 6, 21

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 1, 3, 29

Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. *Advances in Neural Information Processing Systems*, 38:167283–167308, 2026. 3, 29

Shoma Iwai, Tomo Miyazaki, and Shinichiro Omachi. Controlling rate, distortion, and realism: Towards a single comprehensive neural image compression model. In *2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 2888–2897. IEEE, 2024. 6

Charles H Jones. Generalized hockey stick identities and n-dimensional blockwalking. *The Fibonacci Quarterly*, 34(3):280–288, 1996. 5

Nikolai Körber. Perco (SD): Open perceptual compression. In *Workshop on Machine Learning and Compression, NeurIPS 2024*, 2024. 1, 2, 6

Jiahao Li, Bin Li, and Yan Lu. Neural video compression with diverse contexts. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22616–22626. IEEE, 2023. 2

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, June 2026. 8

Zhiyuan Li, Yanhui Zhou, Hao Wei, Chenyang Ge, and Jingwen Jiang. Toward extreme image compression with latent feature guidance and diffusion prior. *IEEE Transactions on Circuits and Systems for Video Technology*, 35(1):888–899, 2024. 6

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t. 1

Jiahe Liu, Youran Qu, Qi Yan, Xiaohui Zeng, Lele Wang, and Renjie Liao. Fréchet video motion distance: A metric for evaluating motion consistency in videos. In *First Workshop on Controllable Video Generation @ICML24*, 2024. 8

Xingchao Liu, Chengyue Gong, and qiang liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=XVjTT1nw5z. 1

Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. Dvc: An end-to-end deep video compression framework. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10998–11007. IEEE, 2019. 2

Qi Mao, Hao Cheng, Tinghan Yang, Libiao Jin, and Siwei Ma. Generative neural video compression via video diffusion prior. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 43239–43248, 2026. 1, 2

Fabian Mentzer, George D Toderici, Michael Tschannen, and Eirikur Agustsson. High-fidelity generative image compression. *Advances in neural information processing systems*, 33:11913–11924, 2020. 1, 2, 6

Alexandre Mercat, Marko Viitanen, and Jarno Vanne. Uvg dataset: 50/120fps 4k sequences for video codec analysis and development. In *Proceedings of the 11th ACM multimedia systems conference*, pp. 297–302, 2020. 8

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 *International Conference on Machine Learning*, pp. 25426–25443. PMLR, 2023. 6

Guy Ohayon, Hila Manor, Tomer Michaeli, and Michael Elad. Compressed image generation with denoising diffusion codebook models. In *Forty-second International Conference on Machine Learning*, 2025. 1, 2, 3, 29

Chanung Park, Joo Chan Lee, and Jong Hwan Ko. Single-step diffusion for image compression at ultra-low bitrates. In *2026 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 6391–6400. IEEE, 2026. 2

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pp. 2256–2265. pmlr, 2015. 1, 3

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021a. 4, 29

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021b. 1, 3

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 32211–32252. PMLR, 2023. 29

Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012. 8

Lucas Theis, Tim Salimans, Matthew D Hoffman, and Fabian Mentzer. Lossy compression with gaussian diffusion. *arXiv preprint arXiv:2206.08889*, 2022. 2, 6

George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full resolution image compression with recurrent neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 5306–5314, 2017. 2

Joel A Tropp. Improved analysis of the subsampled randomized hadamard transform. *Advances in Adaptive Data Analysis*, 3(01n02):115–126, 2011. 4

Thomas Unterthiner, Sjoerd Van Steenkiste, Karol Kurach, Raphael Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric & challenges. *arXiv preprint arXiv:1812.01717*, 2018. 21

Amit Vaisman, Guy Ohayon, Hila Manor, Miki Elad, and Tomer Michaeli. Turbo-ddcm: Fast and flexible zero-shot diffusion-based image compression. In *International Conference on Learning Representations*, volume 2026, pp. 151475–151520, 2026. 1, 2, 3, 5, 26, 29, 31, 32

Jeremy Vonderfecht and Feng Liu. Lossy compression with pretrained diffusion models. In *The Thirteenth International Conference on Learning Representations*, 2025. 2, 6, 15

Haiqiang Wang, Weihao Gan, Sudeng Hu, Joe Yuchieh Lin, Lina Jin, Longguang Song, Ping Wang, Ioannis Katsavounidis, Anne Aaron, and C-C Jay Kuo. Mcl-jcv: a jnd-based h. 264/avc video quality assessment dataset. In *2016 IEEE international conference on image processing (ICIP)*, pp. 1509–1513. IEEE, 2016. 8

Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The thrity-seventh asilomar conference on signals, systems & computers, 2003*, volume 2, pp. 1398–1402. Ieee, 2003. 21

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. 6

Haoning Wu, Erli Zhang, Liang Liao, Chaofeng Chen, Jingwen Hou, Annan Wang, Wenxiu Sun, Qiong Yan, and Weisi Lin. Exploring video quality assessment on user generated contents from aesthetic and technical perspectives. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 20087–20097. IEEE, 2023. 8

Tongda Xu, Ziran Zhu, Dailan He, Yanghao Li, Lina Guo, Yuanyuan Wang, Zhe Wang, Hongwei Qin, Yan Wang, Jingjing Liu, et al. Idempotence and perceptual image compression. In *International Conference on Learning Representations*, volume 2024, pp. 45826–45853, 2024. 2

Naifu Xue, Zhaoyang Jia, Jiahao Li, Bin Li, Yuan Zhang, and Yan Lu. One-step diffusion-based image compression with semantic distillation. *Advances in neural information processing systems*, 38:37108–37144, 2026. 2, 16

Ruihan Yang and Stephan Mandt. Lossy image compression with conditional diffusion models. *Advances in Neural Information Processing Systems*, 36:64971–64995, 2023. 1, 2

Tianwei Yin, Qiang Zhang, Richard Zhang, William T Freeman, Fredo Durand, Eli Shechtman, and Xun Huang. From slow bidirectional to fast autoregressive video diffusion models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22963–22974. IEEE, 2025. 3

Ziyue Zeng, Xun Su, Haoyuan Liu, Bingyu Lu, Yui Tatsumi, and Hiroshi Watanabe. Gvcc: Zero-shot video compression via codebook-driven stochastic rectified flow. *arXiv preprint arXiv:2603.26571*, 2026. 3

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF conference on computer vision and pattern recognition*, pp. 586–595. IEEE, 2018. 6, 21

Tianyu Zhang, Xin Luo, Li Li, and Dong Liu. Stablecodec: Taming one-step diffusion for extreme image compression. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17379–17389. IEEE, 2025. 6

Zheyuan Zhang and Johnson Wu. Regenvc: End-to-end real-time generative video coding at ultra-low bitrate. *arXiv preprint arXiv:2607.28144*, 2026. 1

Hongzhou Zhu, Min Zhao, Guande He, Hang Su, Chongxuan Li, and Jun Zhu. Causal forcing: Autoregressive diffusion distillation done right for high-quality real-time interactive video generation. In *Forty-third International Conference on Machine Learning*, 2026. 2, 3, 29

SUPPLEMENTARY MATERIAL

A EXPERIMENT IMAGE COMPRESSION CONFIGURATIONS

We report here the exact configurations used for the image experiments in Figs. 1, S1, and S2. Training-free methods are evaluated with the same pretrained backbones reported in the main paper, while trained baselines use their publicly released checkpoints without additional training or fine-tuning.

A.1 SD 2.1 (KODAK24, DIV2K, CLIC2020)

All three datasets use the same method set and the same per-method hyperparameters.

Training-free methods.

- **H-DDCM**: $T = 30$ (latent dimension $d = 2^{14}$), M ranging from 22 to 300 (22 points); the DDIM tail length $N + 1$ follows Turbo-DDCM's own heuristic for setting it automatically per M.
- **Turbo-DDCM** ($K = d = 2^{14}$): $T = 30$, using the same 22-point M-grid and per-M deterministic-tail schedule as H-DDCM. We compute lexicographic rank and unrank operations using our table-free recurrence (App. E) rather than Turbo-DDCM's original precomputed table, so runtime comparisons isolate residual-selection/codebook cost rather than avoidable table-construction overhead.
- **DDCM**: hyperparameters from the original paper – $T = 1000$ for $K \in \{256, 1024, 8192\}$ with $M = 1$, and $T = 1000$ for $K = 2048$ with $M \in \{2, 3, 6\}$ and $C = 3$.
- **DiffC**: the schedulers, D_{KL} schedule, and reconstruction timesteps suggested by the original authors, unchanged from their released configuration.
- **PSC-D / PSC-P**: rank 12 and $2^i, i \in \{4, 5, 6, 7, 8\}$ iterations; PSC-D averages 64 posterior samples per reconstruction, and PSC-P uses the authors' own nested-posterior-sampling estimator.

Trained baselines.

- **DiffEIC**: the released `1_2_X` checkpoints for `X` $\in \{1, 2, 4, 8, 16\}$.
- **PerCo (SD)**: three publicly available checkpoints, with the default hyperparameters.
- **StableCodec**: the released checkpoints `stablecodec_ftX` for `X` $\in \{2, 4, 8, 12, 24, 32\}$.
- **HiFiC**: the official checkpoint for the low-quality regime, run from a PyTorch reimplementation rather than the official TensorFlow release.
- **ILLM**: the released `msillm_quality_X` for `X` $\in \{2, 3\}$ and `msillm_quality_vloY` for `Y` $\in \{1, 2\}$ checkpoints.
- **CRDR-D / CRDR-R**: QF $qf \in \{0, 1, 2, 3, 4\}$, with $\beta \in \{0, 3.84\}$ for CRDR-D and CRDR-R.

Classical codec.

- **BPG**: QP values $q \in \{51, 48, 46, 42, 40\}$.

A.2 FLUX (DIV2K)

Four methods are evaluated at FLUX scale ($d = 2^{18}$): H-DDCM, Turbo-DDCM at $K = 2^{14}$ and at $K = d$, and DiffC.

- **H-DDCM**: $T = 30$, M ranging from 43 to 4641 (14 rate points). We use the same step scheduler as SD 2.1, following Vonderfecht & Liu (2025), but subsampled non-uniformly across the sweep: at low bitrate the 30 steps crowd toward the trajectory's higher-noise timesteps, moving to an evenly-spaced subsampling only once the bitrate is high enough. The DDIM tail length N follows the same principle, shrinking from a larger value at low bitrate toward zero as M grows.

- **Turbo-DDCM** ($K = 2^{14}$): the dense-codebook baseline capped well below d, with N matched per rate point to hit the same 14 bpp targets as H-DDCM.
- **Turbo-DDCM** ($K = d = 2^{18}$): we use the same rate targets as the capped-codebook variant. Materializing the full $d \times K$ Gaussian codebook would require approximately 128 GiB in FP16 (App. F), so we generate and process it in blocks of 2^{14} columns.
- **DiffC**: the schedulers, D_{KL} schedule, and reconstruction timesteps suggested by the original authors, unchanged from their released FLUX configuration.

B EXPERIMENT VIDEO COMPRESSION CONFIGURATIONS

We report the configurations used for all video experiments in Sec. 6, including the re-encoded-tail KV-cache geometry and the rate-matched configurations for each training-free method. The same configurations are used across all evaluated video datasets.

Re-encoded-tail geometry. Each CF chunk contains 3 latent frames with flattened dimension $d = 299{,}520$. For the Hadamard transform we zero-pad each chunk to $D = 2^{19}$ and use $T = 4$ bit-carrying consistency steps. The first video segment generates 21 latent frames (81 pixel frames). Each continuation generates 18 new latent frames, preceded by a 3 latent-frame prefix obtained by re-encoding the last 9 reconstructed pixel frames from the preceding segment. Because these frames are already available at both encoder and decoder, the refreshed prefix requires no transmitted bits.

At the first newly coded chunk after each refresh, we use a larger budget $M_{\mathrm{start}} \geq M$ to compensate for the reduced generated context. We choose M_{start} per method such that this chunk carries at least 20,000 bits across its four selection steps: $M_{\mathrm{start}} = 390$ for H-DDCM, 308 for H-DDCM OC, 724 for Turbo-DDCM ($K = 2^{14}$), and 420 for Turbo-DDCM ($K = d$). Once the steady-state M reaches the corresponding floor, we set $M_{\mathrm{start}} = M$.

Training-free methods. All training-free generative methods are evaluated using the same CF backbone at its native 832×480 resolution.

- **H-DDCM**: 10 reference bitrates with M ranging from 16 to 2048 over the padded support $D = 2^{19}$. We use hockey-stick ranking and binary-search unranking (App. E).
- **Overcomplete dictionary H-DDCM** ($B = 8$): 10 configurations rate-matched to the corresponding single-basis H-DDCM points. The OC selection budgets are smaller than the reference M values because each selected atom is drawn from the larger BD-atom dictionary; the exact matched pairs are listed in App. G.
- **Turbo-DDCM** ($K = 2^{14}$): configurations are rate-matched to the same target bit budgets using a deterministic Gaussian codebook; search and reconstruction use FP16.
- **Turbo-DDCM** ($K = d = 299{,}520$): configurations are rate-matched in the same way using the complete Gaussian codebook. Because the complete codebook cannot be stored alongside the backbone, it is processed in blocks of 2^{14} columns.
- **ZeroGVC**: we use independent 32-pixel-frame GOPs. Each GOP begins with a OneDC (Xue et al., 2026) ($\lambda = 1$) I-frame that resets the autoregressive cache; subsequent latents use frame-wise CF with a deterministic $K = 2^{14}$ Turbo-DDCM codebook and 4 denoising steps. We sweep M from 64 to 1024 over 6 configurations and evaluate both bidirectional and non-bidirectional streaming variants. GOPs are repeat-padded to a valid $1 + 4q$ length and cropped back after decoding.

Trained baselines.

- **DCVC-UF**: Microsoft's official implementation and checkpoints, with QP-I $=$ QP-P $\in \{0, \ldots, 63\}$ found via a two-pass search targeting each rate point.

Classical codecs.

- **HEVC**: FFmpeg's `libx265` implementation, with two-pass average-bitrate encoding targeting each rate point.
- **VVC**: Fraunhofer's VVenC/VVdeC implementation, with the same two-pass average-bitrate targeting.

C ADDITIONAL RESULTS

C.1 IMAGE COMPRESSION QUANTITATIVE COMPARISON

Figure S1 extends the SD 2.1 comparison of Fig. 1 to DIV2K and CLIC2020. Across both datasets, H-DDCM follows the same trend observed on Kodak24: it achieves competitive rate-distortion-perception performance while avoiding explicit Gaussian-codebook storage and retaining low round-trip time among the evaluated training-free methods.

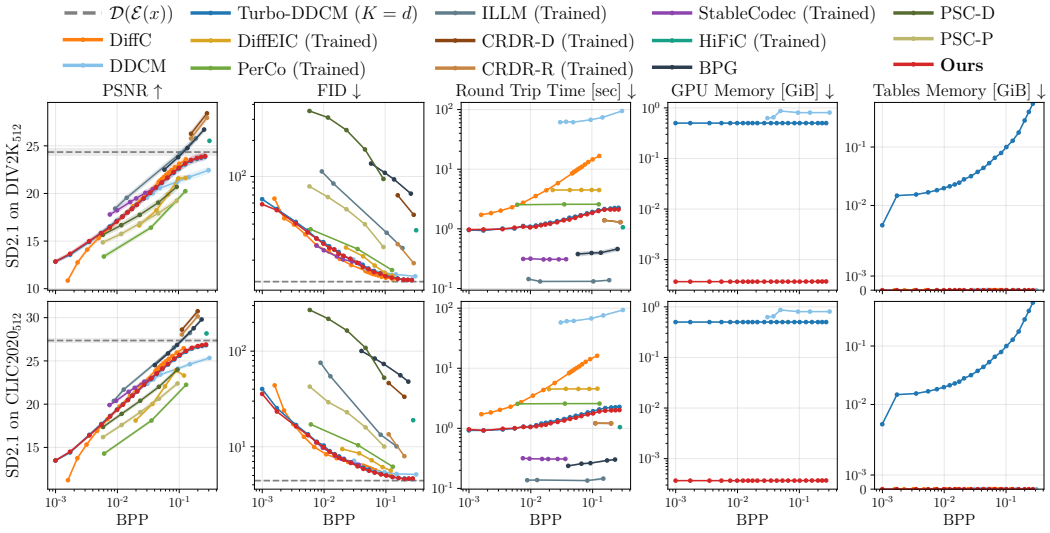

Figure S1: **Additional quantitative comparison of image compression methods (SD2.1).** We compare H-DDCM with training-free diffusion-based and other compression methods in terms of rate, distortion and perception. Distortion is measured by PSNR, perceptual quality is measured by FID (presented in log-scale), and runtime is measured by round-trip encoding and decoding time. GPU memory denotes incremental memory used by training-free methods for storage and computations, while tables memory denotes disk storage required by Turbo-DDCM's precomputed combinatorial tables. The first row corresponds to DIV2K dataset, and the second row corresponds to CLIC2020 dataset. Both datasets are resized and center-cropped to resolution of 512×512, corresponding to $d = 2^{14}$. The dashed gray vertical line corresponds to the encoder-decoder bound. H-DDCM achieves competitive results on the rate-distortion-perception tradeoff, all the while being the fastest training-free method, avoiding explicit codebook materialization. Error bars represent the standard error of the mean (SEM).

Figure S2 reports complementary metrics for the image experiments in Figs. 1 and S1. We include SSIM and LPIPS together with separate encoding, decoding, and round-trip times and the memory requirements of the training-free methods. PSC is omitted from the timings and memory consumption due to its slow runtime.

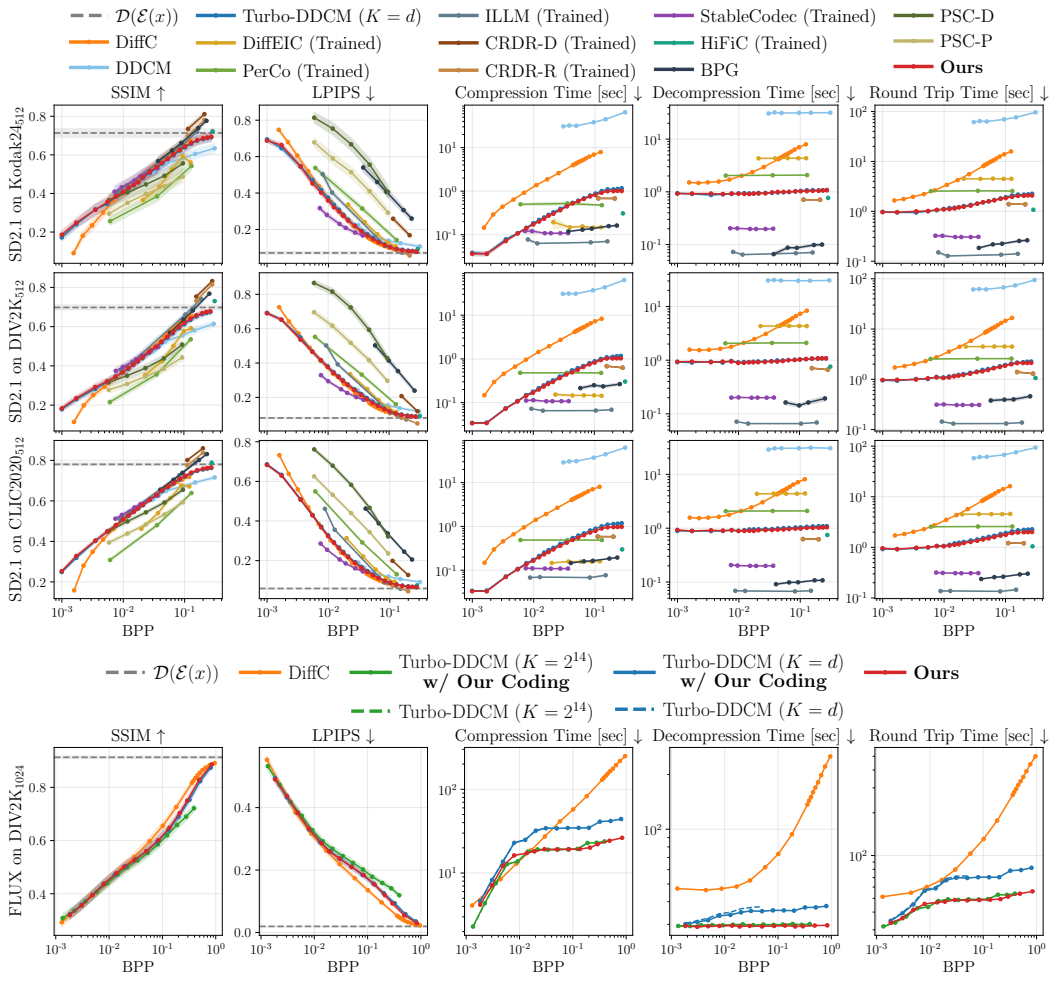

Figure S2: **Complementary quantitative comparison of Figs. 1 and S1 of image compression methods.** We compare H-DDCM with training-free diffusion-based and other compression methods in terms of rate, distortion and perception. Distortion is measured by SSIM and LPIPS, runtime is measured by encoding (compression) time, decoding (decompression) time, and round-trip time (compression and decompression time). PSC is omitted due to its slow runtime. The first, second and third rows correspond to Kodak24, DIV2K and CLIC2020 datasets, respectively, evaluated on SD2.1 (512×512, corresponding to $d = 2^{14}$). The fourth row corresponds to DIV2K dataset, evaluated on FLUX (1024×1024, corresponding to $d = 2^{18}$). The dashed gray vertical line corresponds to the encoder-decoder bound. H-DDCM achieves competitive quality, at the same time of being the fastest training-free method, without materializing the dictionary. Error bars represent the standard error of the mean (SEM).

C.2 IMAGE COMPRESSION QUALITATIVE COMPARISON

Figures S3, S4 and S5 present additional SD 2.1 qualitative comparisons between diffusion-based training-free methods on Kodak24, DIV2K, and CLIC2020 datasets (at 512×512), respectively. Figure S6 presents the same, but for FLUX on DIV2K dataset (at 1024×1024).



Figure S3: **Qualitative comparison of training-free diffusion-based image compression methods (SD2.1).** The images are taken from the Kodak24 dataset (resized and center-cropped to resolution of 512×512).

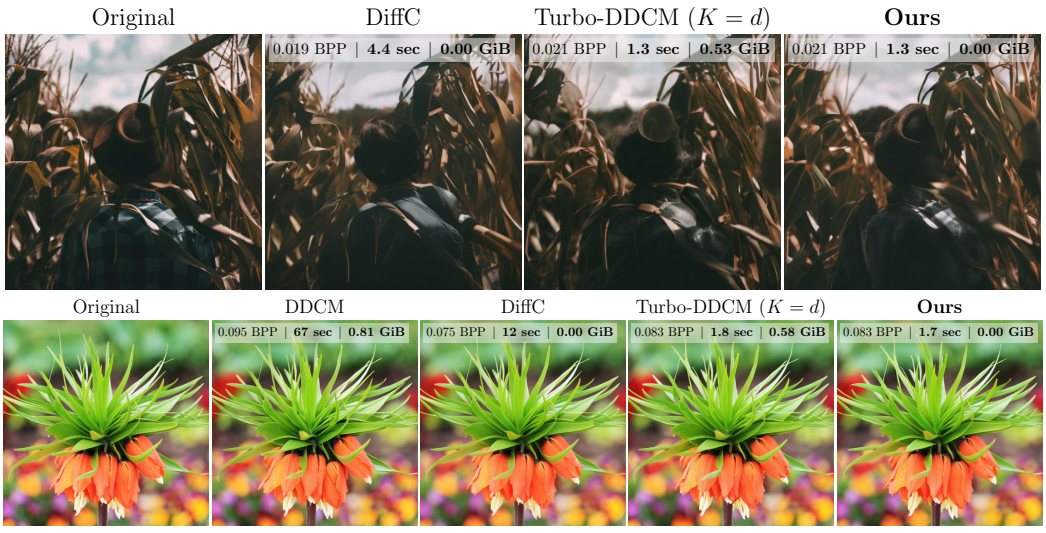

Figure S4: **Qualitative comparison of training-free diffusion-based image compression methods (SD2.1).** The images are taken from the DIV2K dataset (resized and center-cropped to resolution of 512×512).

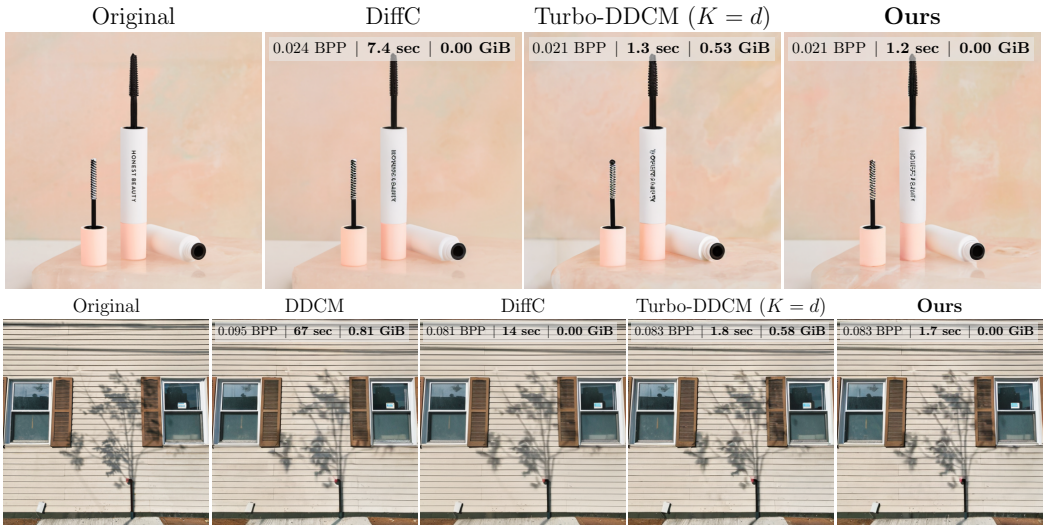

Figure S5: **Qualitative comparison of training-free diffusion-based image compression methods (SD2.1).** The images are taken from the CLIC2020 dataset (resized and center-cropped to resolution of 512×512).

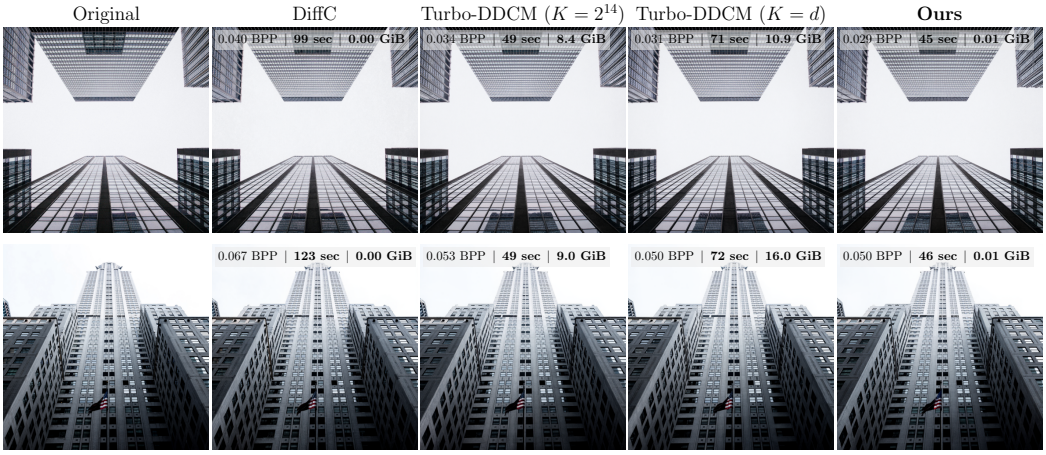

Figure S6: **Qualitative comparison of training-free diffusion-based image compression methods (FLUX).** The images are taken from the DIV2K dataset (resized and center-cropped to resolution of 1024×1024).

C.3 VIDEO COMPRESSION QUANTITATIVE COMPARISON

Figure S7 extends the video comparison of Fig. 3, to HEVC Classes B, C and D datasets, resized and center-cropped to resolution of 832×480 resolution. Because HEVC-D is natively 416×240 resolution, we upsample it to 832×480 to match the backbone's native operating resolution, but evaluate all quality metrics at the original 416×240 resolution rather than the upsampled size. The results follow the main paper trend: H-DDCM OC achieves a competitive rate-distortion-perception performance to that of competing methods, while providing the highest measured throughput among the evaluated methods using the CF backbone, without materializing the dictionary.

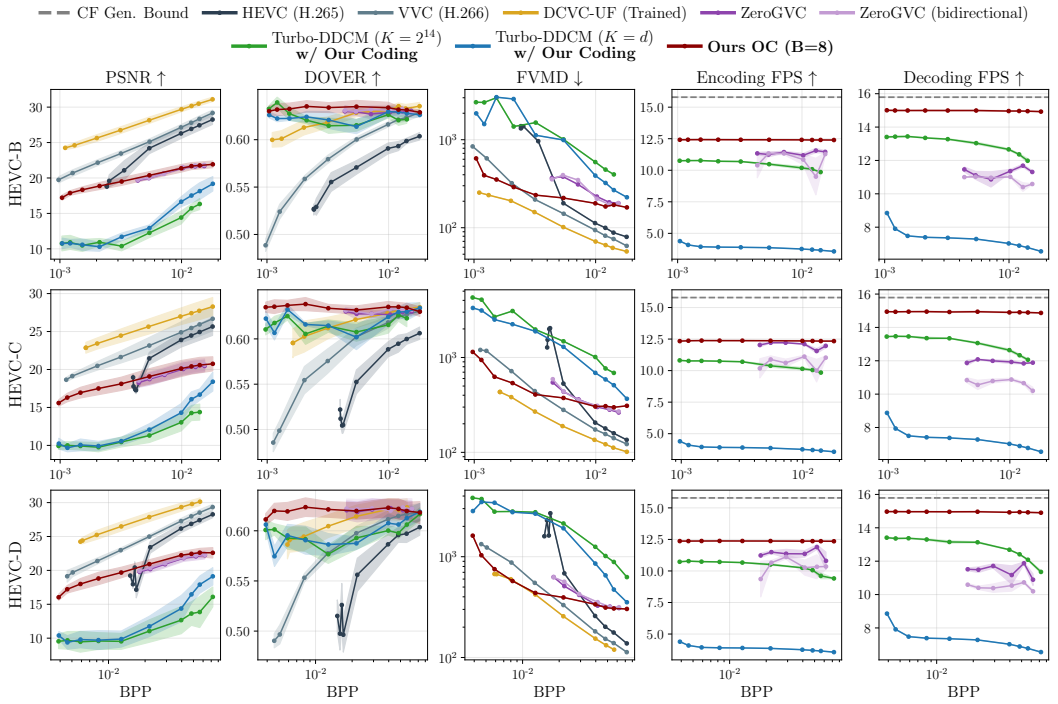

Figure S7: **Quantitative comparison of video compression methods.** We compare H-DDCM OC ($B = 8$) with training-free methods using the same CF backbone, and other compression baselines in terms of rate, distortion and perception. Distortion is measured by PSNR, and perceptual quality is measured by DOVER and FVMD (presented in log-scale). Encoding and decoding FPS report compression and decompression throughput, respectively, and are compared among methods using the CF backbone for fair comparison. The first row corresponds to HEVC-B dataset, the second row corresponds to HEVC-C dataset, and the last row corresponds to HEVC-D dataset. H-DDCM achieves competitive rate-distortion-perception performance among the CF-based training-free methods while providing the highest measured encoding and decoding throughput, without codebooks materialization. Error bars represent the standard error of the mean (SEM).

Figure S8 presents complementary metrics of Figs. 3 and S7. Specifically, it presents the MS-SSIM (Wang et al., 2003), LPIPS (Zhang et al., 2018) and DISTS (Ding et al., 2020) distortion metrics, and perceptual quality metrics of FID (Heusel et al., 2017) and FVD (Unterthiner et al., 2018).

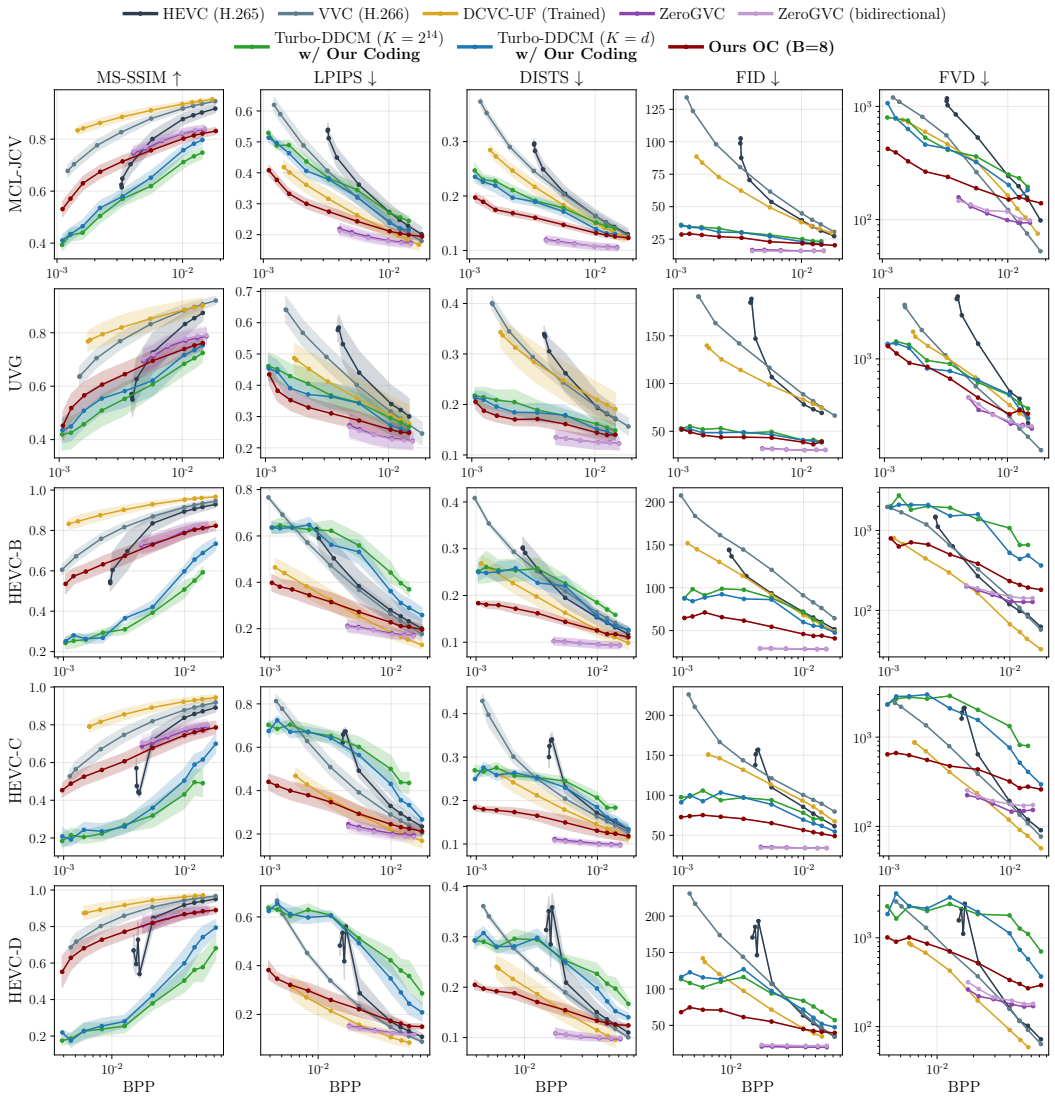

Figure S8: **Complementary quantitative comparison of Figs. 3 and S7 of video compression methods.** We compare H-DDCM OC ($B = 8$) with training-free methods using the same CF backbone and other compression baselines in terms of rate, distortion and perception. Distortion is measured by MS-SSIM, LPIPS and DISTS, perceptual quality is measured by FID and FVD (presented in log-scale). The first, second and third rows correspond to HEVC-B, HEVC-C and HEVC-D datasets, respectively. H-DDCM achieves competitive rate-distortion-perception performance among the CF-based training-free methods. Error bars represent the standard error of the mean (SEM).

C.4 VIDEO COMPRESSION QUALITATIVE COMPARISON

Figures S9, S10, S11, S12 and S13 present additional qualitative comparisons using the same setting and baselines as presented in Sec. 6. We include spatial frames together with spatiotemporal slices to show both reconstruction fidelity and temporal continuity.



Figure S9: **Qualitative comparison of video-compression methods.** The presented video is taken from MCL-JCV (832×480) dataset. Our method produces comparable results on the rate-distortion-perception tradeoff, while being the fastest method utilizing the CF backbone. Note the temporal discontinuities at I-frame boundaries in the ST slice of ZeroGVC.



Figure S10: **Qualitative comparison of video-compression methods.** The presented video is taken from HEVC-C (832×480) dataset. Our method produces comparable results on the rate-distortion-perception tradeoff, while being the fastest method utilizing the CF backbone. Note the temporal discontinuities at I-frame boundaries in the ST slice of ZeroGVC.

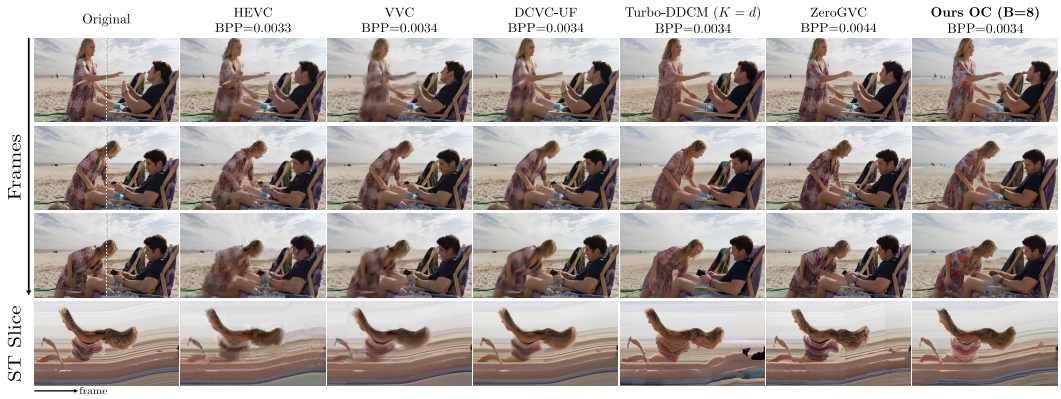

Figure S11: **Qualitative comparison of video-compression methods.** The presented video is taken from MCL-JCV (832×480) dataset. Our method produces comparable results on the rate-distortion-perception tradeoff, while being the fastest method utilizing the CF backbone. Note the temporal discontinuities at I-frame boundaries in the ST slice of ZeroGVC.

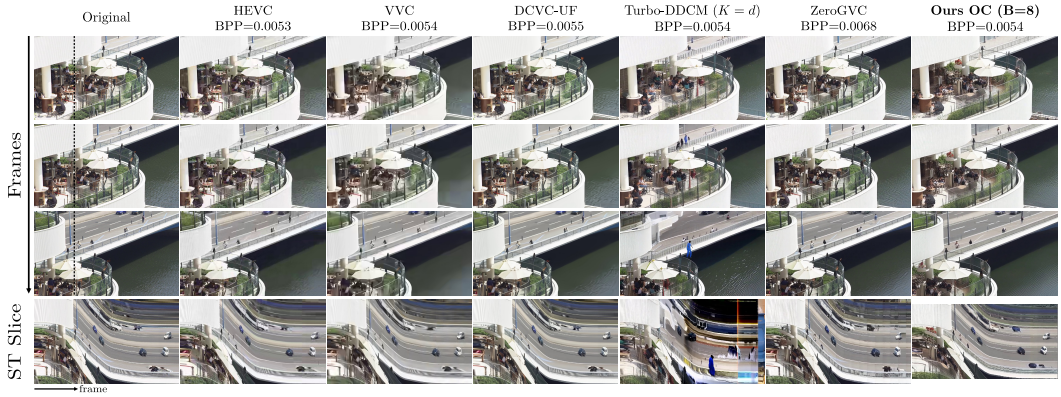

Figure S12: **Qualitative comparison of video-compression methods.** The presented video is taken from HEVC-B (832×480) dataset. Our method produces comparable results on the rate-distortion-perception tradeoff, while being the fastest method utilizing the CF backbone. Note the temporal discontinuities at I-frame boundaries in the ST slice of ZeroGVC.

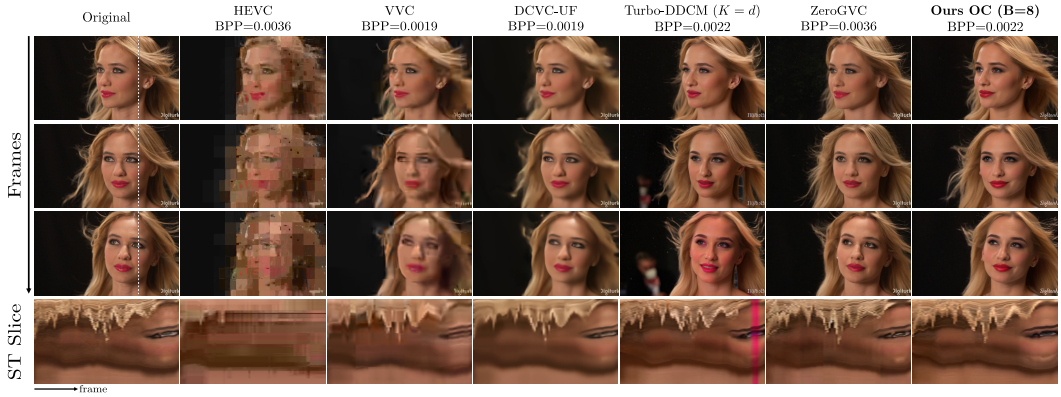

Figure S13: **Qualitative comparison of video-compression methods.** The presented video is taken from UVG (832×480) dataset. Our method produces comparable results on the rate-distortion-perception tradeoff, while being the fastest method utilizing the CF backbone. Note the temporal discontinuities at I-frame boundaries in the ST slice of ZeroGVC.

C.5 Ablation of the overcomplete Hadamard dictionary

We compare the single-basis H-DDCM dictionary with the overcomplete (OC) variants using $B \in \{2, 4, 8\}$ randomized bases on the MCL-JCV dataset, following the same evaluation described in Sec. 6. Figure S14 presents the quantitative comparison, and shows that the OC variants improve both distortion and perceptual metrics across all evaluated bitrates, with $B = 8$ providing the strongest results among the tested values. Figure S15 presents the qualitative comparison between the single-basis variant and $B = 8$ variant at BPP $= 0.01$. Both methods achieve compelling results for this low BPP value, however, the overcomplete dictionary variant is more consistent with the original video and exhibits higher visual realism

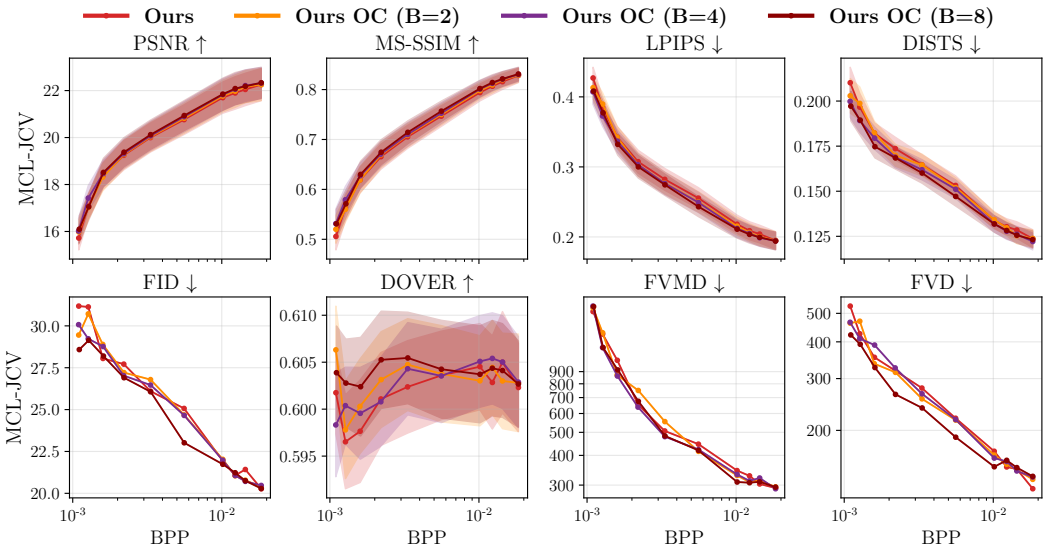

Figure S14: **Quantitative comparison ablation between H-DDCM and H-DDCM OC ($B \in \{2, 4, 8\}$).** The overcomplete dictionary variant provides better performance across all distortion and perceptual quality metrics. Error bars represent standard error of the mean (SEM).

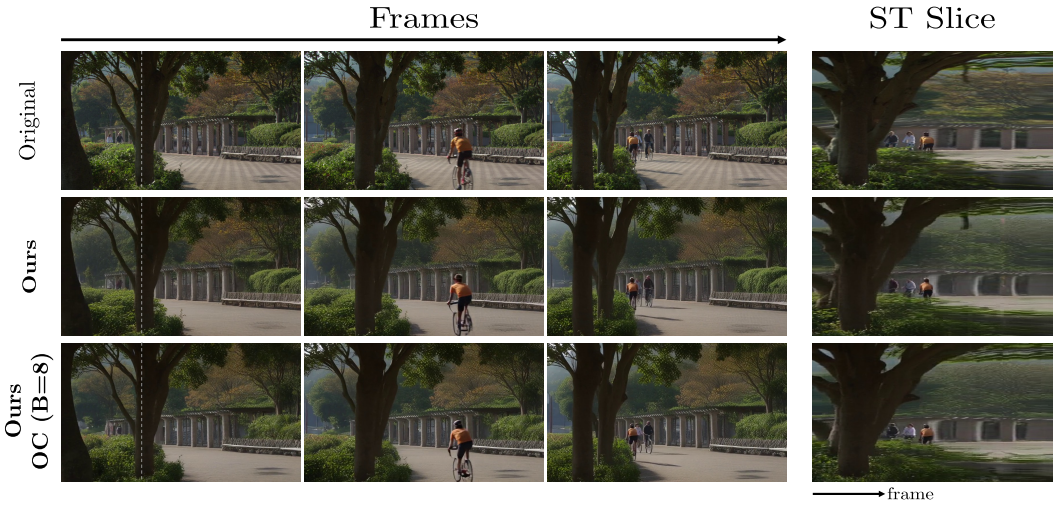

Figure S15: **Qualitative comparison ablation between H-DDCM and H-DDCM OC ($B = 8$).** The presented video is taken from HEVC-B (832×480) dataset, for BPP $= 0.01$. Both proposed methods achieve compelling results, however the perceptual quality and video consistency of the overcomplete dictionary is better.

C.6 Ablation of transpose and pseudo-inverse synthesis

When the dictionary is orthonormal, the columns of the selected sub-dictionary $\mathbf{C}_S \in \mathbb{R}^{d\times M}$ are mutually orthogonal ($\mathbf{C}_S^\top \mathbf{C}_S = \mathbf{I}_M$), so synthesizing via the pseudo-inverse $(\mathbf{C}_S^\dagger)^\top \boldsymbol{\alpha}_S = \mathbf{C}_S(\mathbf{C}_S^\top \mathbf{C}_S)^{-1}\boldsymbol{\alpha}_S$ collapses identically to the direct combination $\mathbf{C}_S\boldsymbol{\alpha}_S$ in (4). For the overcomplete dictionary, where selected atoms may originate from different bases are thus no longer mutually orthogonal. Therefore, we compare between synthesizing the noise direction via the pseudo-inverse $(\mathbf{C}_S^\dagger)^\top \boldsymbol{\alpha}_S$ and retaining the direct summation heuristic $\mathbf{C}_S\boldsymbol{\alpha}_S$ of Vaisman et al. (2026), on the UVG dataset. As shown in Fig. S16, both synthesis variants achieve nearly identical rate-distortion-perception performance across all evaluated BPP values, while the cosine similarity between their selected noise vectors remains highly aligned. We therefore use direct synthesis for H-DDCM OC variant to avoid the additional solve.

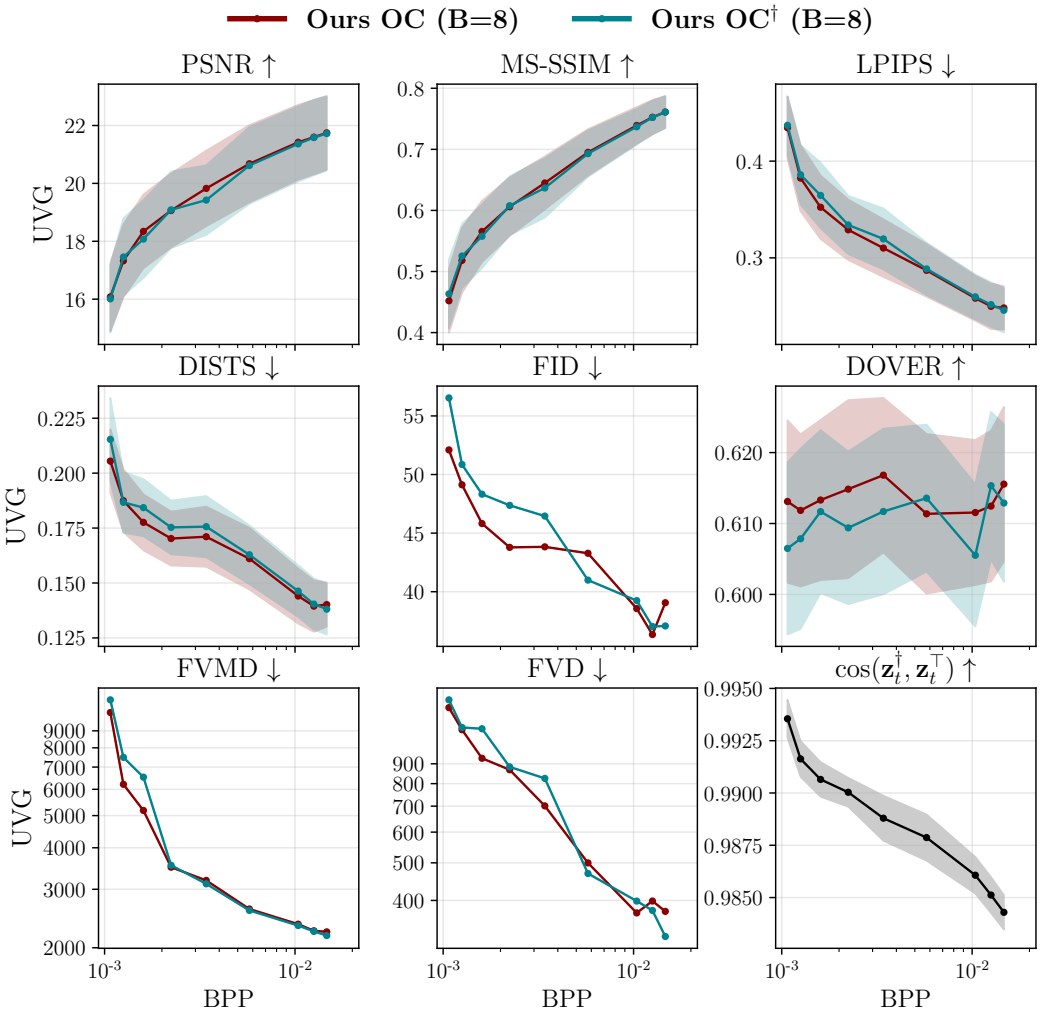

Figure S16: **Quantitative comparison ablation between H-DDCM OC utilizing the transpose variant, and H-DDCM OC utilizing the pseudo-inverse, both with $B = 8$.** Both variants achieve similar performance, and the cosine similarity between their selected noise vectors remains close to 1, *i.e.*, the selected $\mathbf{z}_t^\star$ is almost aligned. Pseudo-inverse is denoted by $(\cdot)^\dagger$. Error bars represent standard error of the mean (SEM).

C.7 Ablation of cache refresh and first-step initialization

We ablate two video-specific components of Sec. 6, using the OC dictionary with $B = 8$: the re-encoded-tail KV-cache refresh and the first-step initialization described in App. D. For cache handling, we compare our full method with (i) a hard reset at each continuation boundary ("w/o re-encoded-tail"), and (ii) a continuously growing cache without refresh ("w/ continuous cache").

Figure S17 presents the quantitative comparison on the MCL-JCV dataset, following the same evaluation described in Sec. 6. The full refresh mechanism improves over both cache alternatives across the evaluated rates. The qualitative comparison in Fig. S18 shows the corresponding failure modes, for BPP $= 0.01$: hard resets introduce discontinuities at segment boundaries, while the continuously growing cache produces severe artifacts once the context extends beyond the regime seen during training.

Additionally, we remove the first-step initialization mechanism ("w/o first step init."). Although this variant can improve some aggregate quantitative metrics as seen in Fig. S17, it struggles to consistently temporally align with the input video, as seen in Fig. S18.

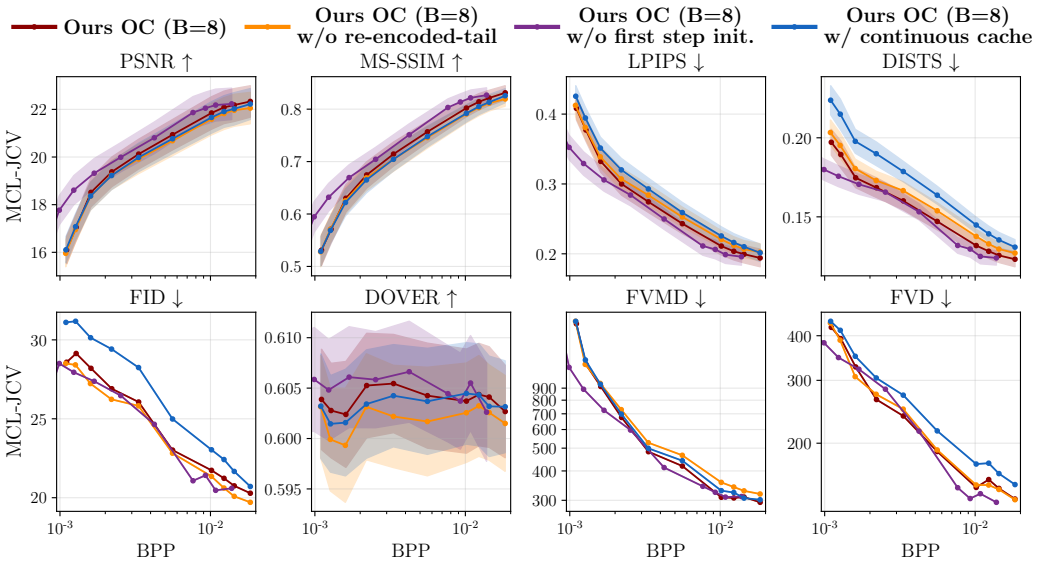

Figure S17: **Quantitative comparison ablation between H-DDCM OC ($B = 8$), H-DDCM OC ($B = 8$) without the re-encoded-tail mechanism for both hard cache reset – "w/o re-encoded-tail", and without cache reset – "w/ continuous cache", and H-DDCM OC ($B = 8$) without the first timestep initialization mechanism.** H-DDCM OC ($B = 8$) achieves better with the re-encoded tail mechanism. Without the first timestep initialization mechanism provides better metrics, however as can be seen in Fig. S18, it struggles to consistently align with the input video. Error bars represent standard error of the mean (SEM).

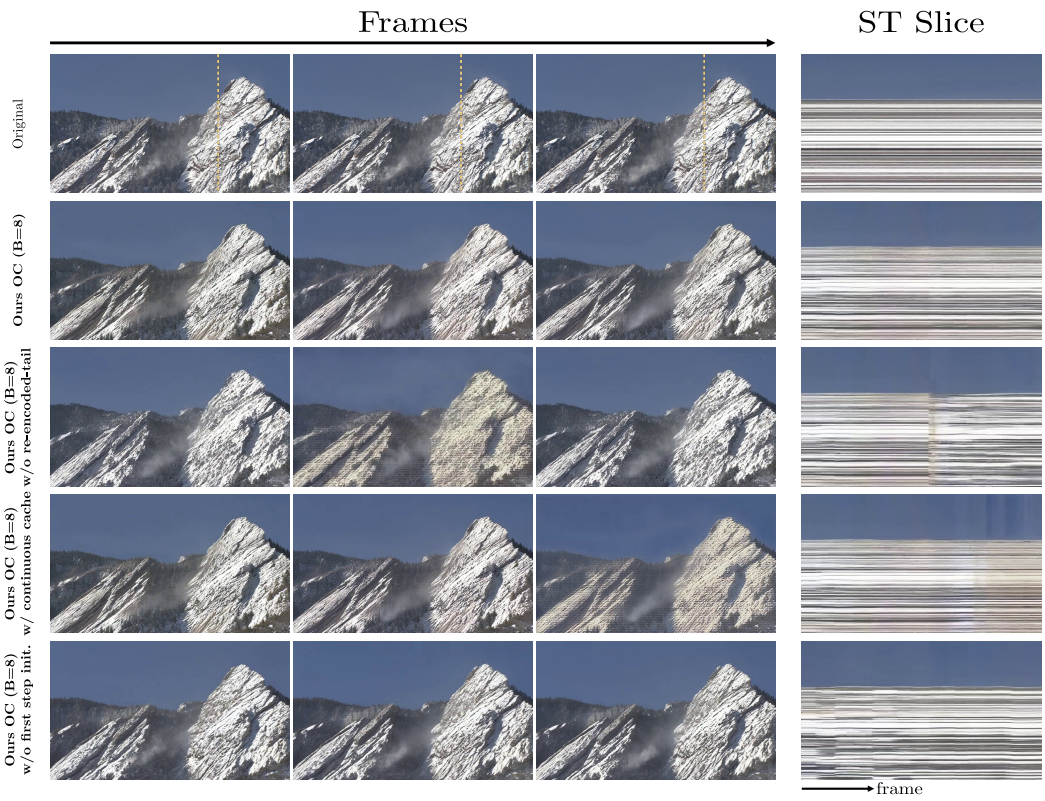

Figure S18: **Qualitative comparison ablation between H-DDCM OC ($B = 8$), H-DDCM OC ($B = 8$) without the re-encoded-tail mechanism for both hard cache reset – "w/o re-encoded-tail", and without cache reset – "w/ continuous cache", and H-DDCM OC ($B = 8$) without the first timestep initialization mechanism.** The presented video is taken from MCL-JCV (832×480) dataset, for BPP $= 0.01$. While H-DDCM OC ($B = 8$) is aligned with the input video, without the re-encoded-tail mechanism when a new segment begins and a KV cache refresh occurs – there is a temporal discontinuity which can be observed in the spatiotemporal slice, and artifacts which can be observed in the second presented frame. For the continuous cache variant, *i.e.*, without cache reset, there are severe artifacts caused as this size of a cache/context is out-of-distribution for the model, which could be observed in the third presented frame. Additionally, without the first timestep initialization the method struggles to consistently align with the input video.

D SAMPLING FORMULATIONS

Section 3.1 writes the stochastic reverse transition in a sampler-agnostic form. Here we give the exact image and video updates used in our experiments.

Image formulation: DDPM ancestral sampling. Following Vaisman et al. (2026), our image backbones (SD 2.1, FLUX) share the standard DDPM posterior mean (Ho et al., 2020). Writing $\gamma_t \in (0, 1)$ for the per-step signal-retention coefficient and $\bar{\alpha}_t = \prod_{i \le t} \gamma_i$ for its cumulative product; we use γ_t rather than the more common α_t, to distinguish the schedule from the sparse coefficient vector $\boldsymbol{\alpha}_t$. The mean and variance of the DDPM schedule are

$$\boldsymbol{\mu}_t(\mathbf{x}_t) = \frac{\sqrt{\bar{\alpha}_{t-1}}\,(1-\gamma_t)}{1-\bar{\alpha}_t}\,\hat{\mathbf{x}}_{0|t} + \frac{\sqrt{\gamma_t}\,(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\,\mathbf{x}_t, \qquad \sigma_t = \sqrt{1-\gamma_t}. \tag{S1}$$

This is exactly the affine map of $\mathbf{x}_t$ referenced in Sec. 3.1: linear in $\mathbf{x}_t$ once the network's estimate $\hat{\mathbf{x}}_{0|t}$ is fixed by the forward pass at step t.

The last $N + 1$ steps of the trajectory (Sec. 4.2) are instead run deterministically, since they carry no bits and the decoder reproduces them unassisted. This replaces (S1) with the $\sigma_t = 0$ DDIM update (Song et al., 2021a),

$$\boldsymbol{\mu}_t^{\mathrm{DDIM}}(\mathbf{x}_t) = \sqrt{\bar{\alpha}_{t-1}}\,\hat{\mathbf{x}}_{0|t} + \sqrt{1-\bar{\alpha}_{t-1}} \cdot \frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\,\hat{\mathbf{x}}_{0|t}}{\sqrt{1-\bar{\alpha}_t}}, \tag{S2}$$

which still reads off the same network estimate $\hat{\mathbf{x}}_{0|t}$, only recombined with $\mathbf{x}_t$ along the noiseless direction consistent with it.

Video formulation: consistency sampling. The causal video backbone distills a bidirectional teacher into a few-step AR student (Zhu et al., 2026; Huang et al., 2026) parameterized as a consistency model (Song et al., 2023). Specifically, a single forward evaluation $f_\theta(\mathbf{x}_t, t)$ maps a noisy chunk at any noise level t directly to the origin of its probability flow trajectory, satisfying $f_\theta(\mathbf{x}_t, t) \approx f_\theta(\mathbf{x}_{t'}, t') \approx \hat{\mathbf{x}}_0$ for all $t \neq t'$, subject to the boundary condition $f_\theta(\mathbf{x}_\epsilon, \epsilon) = \mathbf{x}_\epsilon$ at the terminal noise scale ϵ. In practice, the distilled student yields an empirical approximation of this mapping, which we denote as $\hat{\mathbf{x}}_{0|t} := f_\theta(\mathbf{x}_t, t)$ to maintain consistent notation. Unlike the DDPM transition in (S1), which interpolates between the current state $\mathbf{x}_t$ and the estimate $\hat{\mathbf{x}}_{0|t}$, consistency sampling constructs the transition drift toward the next discretized step τ_{n+1} by directly rescaling the predicted clean chunk:

$$\boldsymbol{\mu}_{\tau_n}(\mathbf{x}_{\tau_n}) := (1-\tau_{n+1})\, f_\theta(\mathbf{x}_{\tau_n}, \tau_n) = (1-\tau_{n+1})\, \hat{\mathbf{x}}_{0|\tau_n}. \tag{S3}$$

The residual noise vector $\mathbf{z}^\star_{\tau_n}$ is selected against $\hat{\mathbf{x}}_{0|\tau_n}$ and injected at the target schedule level τ_{n+1} scaled by $\sigma_{\tau_n} := \tau_{n+1}$:

$$\mathbf{x}_{\tau_{n+1}} = \boldsymbol{\mu}_{\tau_n}(\mathbf{x}_{\tau_n}) + \sigma_{\tau_n}\, \mathbf{z}^\star_{\tau_n}. \tag{S4}$$

Initializing the first step. Both formulations select noise against the residual $\mathbf{r}_t := \mathbf{x}_{\mathrm{src}} - \hat{\mathbf{x}}_{0|t}$ (Sec. 3.2), which needs $\hat{\mathbf{x}}_{0|t}$ from the network. Our video backbone's consistency schedule is just $n \in \{1, \ldots, 4\}$ steps end to end, and $\hat{\mathbf{x}}_{0|\tau_n}$ in (S4) is only defined once a previous step has run; at $n = 0$, *i.e.*, before the model evaluation, there is no previous step, so no network prediction exists yet to form $\mathbf{r}_t$ against. Ohayon et al. (2025) and Vaisman et al. (2026) suggest using a shared seed to generate random Gaussian noise for the initial noise $\mathbf{x}_T$. We instead use an approximation of the posterior mean: the source chunk's own spatial mean, $\boldsymbol{\mu}_{\mathrm{src}} := \mathrm{mean}(\mathbf{x}_{\mathrm{src}})\mathbf{1}$, giving a proxy residual of $\mathbf{x}_{\mathrm{src}} - \boldsymbol{\mu}_{\mathrm{src}}$ for that one step only; every subsequent step reverts to the network's own $\hat{\mathbf{x}}_{0|t}$. We ablate this initialization in App. C.

Bit-carrying steps. Section 4.2 writes the number of bit-carrying steps as $S = T - N - 1$ for images and $S = T$ for video (Eq. (10)). The $N + 1$ accounts for the tail described above: the image formulation spends its last $N + 1$ steps on the deterministic, bit-free $\boldsymbol{\mu}_t^{\mathrm{DDIM}}$ update (S2) rather than the bit-carrying $\boldsymbol{\mu}_t$ of (S1) – including the final transition $t = 1 \to 0$, where $\bar{\alpha}_0 = 1$ collapses (S2) to the network's bare estimate $\hat{\mathbf{x}}_{0|1}$, carrying no bits regardless of N. The video formulation has no deterministic tail at all: with only four consistency steps to begin with, every one of them – including the first, bootstrapped from $\boldsymbol{\mu}_{\mathrm{src}}$ above – selects and transmits noise.

E BITSTREAM PROTOCOL

At each bit-carrying step, the encoder transmits an unordered support of M selected atoms together with their quantized coefficients. Listing the indices independently would require $M\lceil \log_2 K \rceil$ bits and redundantly encode all $M!$ permutations of the same support. We instead encode the sorted support by its rank among the $\binom{K}{M}$ possible M-subsets, requiring $\lceil \log_2 \binom{K}{M} \rceil$ support bits plus MC bits for the corresponding quantized coefficients in $\mathcal{V}$.

Group-factorized encoding. Rather than rank the full support of size K in one piece, we partition it into G equal groups of $K_G := K/G$ elements each and factorize the bitstream by group: writing m_g for the number of atoms selected from group g, so $\sum_{g=1}^{G} m_g = M$, we first encode the count vector $(m_1, \ldots, m_G)$ itself, then rank each group's own local support of size K_G separately with the machinery below.

A stars-and-bars map turns the count vector into $G-1$ strictly increasing separators,

$$s_g = \sum_{q=1}^{g} m_q + (g-1), \qquad g = 1, \ldots, G-1, \tag{S5}$$

which is exactly a $(G-1)$-subset of an $(M+G-1)$-element universe and is ranked by the same lexicographic construction as below, at a cost of

$$L_{\text{counts}} = \left\lceil \log_2 \binom{M+G-1}{G-1} \right\rceil \tag{S6}$$

bits. Every nonempty group then contributes its own rank of an m_g-subset of $\{0, \ldots, K_G - 1\}$, plus $m_g C$ coefficient bits, for a total slot length of

$$L_{\text{grouped}} = L_{\text{counts}} + \sum_{g=1}^{G} \left(\left\lceil \log_2 \binom{K_G}{m_g} \right\rceil + m_g C \right); \tag{S7}$$

empty groups contribute neither term. Decoding reverses the order: unrank the separators to recover every m_g, which fixes the exact length of every local payload that follows, then unrank each nonempty group's own support and coefficients and re-form the flat index $\ell = gK_G + j$. No group identity is transmitted on its own – it is implicit in which local rank a given index came from.

Setting $G = 1$ recovers a single flat rank over all K coordinates ($K_G = K$, $L_{\text{counts}} = 0$). The overcomplete dictionary of Sec. 4.3 sits at the other end, $G = B$, with each of the B randomized bases forming one group of $K_G = D$ atoms (total support $K = BD$); a flat index over the full BD-atom union would push the arithmetic below to a combinadic space B times larger than any other method here ever ranks (App. G gives the geometry in full).

Lexicographical combination encoding and decoding. Let $c = [c_1, \ldots, c_M]$, $c_1 < \cdots < c_M$, be the sorted combination of selected indices drawn from $\{0, \ldots, K-1\}$. Its lexicographic rank is

$$\text{rank}(c) \;=\; \sum_{i=1}^{M} \sum_{x=c_{i-1}+1}^{c_i - 1} \binom{K-x-1}{M-i}, \qquad c_0 := -1, \tag{S8}$$

i.e., for every index x skipped before reaching c_i, we add the number of combinations that would have been reachable had x been chosen instead. This gives a bijection between combinations and integers in $[0, \binom{K}{M})$, so $\text{rank}(c)$ alone determines c uniquely.

Decoding inverts this by the same accounting in reverse: starting from $x = 0$, we repeatedly test whether the next binomial term still fits inside the remaining rank,

$$\text{while } \binom{K-x-1}{M-i} \le \text{rank}: \quad \text{rank} \leftarrow \text{rank} - \binom{K-x-1}{M-i}, \;\; x \leftarrow x+1, \tag{S9}$$

and take the first x for which the term exceeds the remaining rank as c_i, before moving on to c_{i+1}. This recovers every element of c in increasing order and is the exact inverse of (S8).

Implementation details. A direct evaluation of (S8) and (S9) requires up to K binomial-coefficient evaluations per encoded/decoded combination. A natural way to avoid recomputing each one from scratch is to precompute and cache the full table of values $\binom{K-x-1}{M-i}$ for every (x, i) once, ahead of time, and simply look up the relevant entries at runtime (Vaisman et al., 2026). This is effective as long as K and M remain modest, but the table itself has size $O(K \cdot M)$: at the scale required for high-resolution latent spaces – *e.g.*, K on the order of 2^{18}, with M occasionally exceeding 10^3 – such a table would need on the order of 10^8 entries, making it impractical to build, store, or keep resident in memory.

We instead avoid materializing any table with an incremental recurrence: consecutive binomial terms in (S8) and (S9), writing n, k for $(K-x-1, M-i)$, are related by two exact, $O(1)$ identities (Pascal's rule, applied in each of its two directions):

$$\binom{n-1}{k-1} = \binom{n}{k} \cdot \frac{k}{n}, \qquad \binom{n-1}{k} = \binom{n}{k} \cdot \frac{n-k}{n}. \tag{S10}$$

Starting from a single initial value, we advance a running binomial coefficient through the same (n, k) lattice that (S8) and (S9) would otherwise re-derive from scratch at every term, using whichever of the two recurrences in (S10) matches the step being taken. This yields exactly the same rank (and its exact inverse) using $O(K)$ total $O(1)$ updates and $O(1)$ auxiliary memory, with no dependence on a precomputed table at all. Because the intermediate binomial coefficients can still grow to several hundred bits at the K, M values we consider, the running value is maintained with an arbitrary-precision integer library rather than fixed-width integers. We verified this procedure produces bit-identical encodings and decodings to the direct formula at every (K, M) pair we tested, while remaining tractable at scales where a precomputed table is not.

A second table-free option, the hockey-stick identity (see below for additional details), rewrites (S8)'s inner sum in closed form: for each selected index c_i, the count skipped since c_{i-1} becomes a single difference of two fresh binomial evaluations,

$$\binom{K - c_{i-1} - 1}{M - i + 1} - \binom{K - c_i}{M - i + 1}, \tag{S11}$$

summed over $i = 1, \ldots, M$ to encode; decoding inverts (S11) by a binary search over c_i at each of the M steps. This costs $O(M)$ fresh binomial evaluations to encode and $O(M \log K)$ to decode, against the recurrence's flat $O(K)$ – and since evaluating $\binom{n}{k}$ from scratch is not itself $O(1)$ in k, which mechanism wins depends on how large M is next to K. Both run on the CPU in parallel with the diffusion model's own GPU forward pass, so whichever one is cheap for the M regime it runs at stays negligible next to the model's own step cost. We use both in this paper: for images, whose M sweeps well into the regime where hockey-stick is no longer competitive, we use the recurrence for both ranking and unranking. For video, H-DDCM and both Turbo-DDCM variants alike, both ranking and unranking use the hockey-stick identity throughout at every M we sweep.

For completeness, in the following, we summarize the hockey-stick identity and the derivation of (S11). The hockey-stick identity is defined as

$$\sum_{y=k}^{n} \binom{y}{k} = \binom{n+1}{k+1}. \tag{S12}$$

In (S8) we have $\sum_{x=c_{i-1}+1}^{c_i-1} \binom{K-x-1}{M-i}$, where the top index varies, not the bottom. Writing $k = M - i$ and substituting $y = K - x - 1$ we obtain

$$\sum_{y=K-c_i}^{K-c_{i-1}-2} \binom{y}{k} = \sum_{y=k}^{K-c_{i-1}-2} \binom{y}{k} - \sum_{y=k}^{K-c_i-1} \binom{y}{k} = \binom{K - c_{i-1} - 1}{M - i + 1} - \binom{K - c_i}{M - i + 1},$$

where the last equality obtained by (S12), which also completes the derivation.

Measured comparison (with $G = 1$). At FLUX scale ($M = 219$, one image) the table already costs 61.1 s to build at $K = 2^{14}$, but 1386.8 s ($\approx$ 23 min) at $K = 2^{18} = d$, occupying 0.32 and 8.10 GiB on disk respectively – and, since each entry's bit-length grows with both K and M, the table size grows worse than linearly in M: extrapolating to the larger M used elsewhere in our

K	Mechanism	One-time setup	Compress (s)	Decompress (s)	Total (s)
2^{14}	Table (Vaisman et al., 2026)	61.1 s	19.05	29.75	48.80
2^{14}	Recurrence (ours)	0 s	18.06	27.68	45.73
$2^{18} = d$	Table (Vaisman et al., 2026)	1386.8 s	37.09	38.14	75.23
$2^{18} = d$	Recurrence (ours)	0 s	33.81	34.69	68.50

Table S1: **Turbo-DDCM round-trip bitstream cost at $M = 219$ (FLUX, with $d = 2^{18}$).** The per-call cost is close between the two mechanisms; the one-time table build is what the original protocol pays fresh for every (K, M) it is run at, and which the recurrence never pays.

FLUX grid projects tables of hundreds of gigabytes to several terabytes, one of which would have to be rebuilt from scratch for every distinct (K, M) pair. Our recurrence pays none of this, at a per-call cost that is comparable to or faster than a table lookup once the table already exists (Table S1).

Standalone rank/unrank timings make the crossover between the two table-free mechanisms explicit (Table S2). Hockey-stick evaluation is substantially cheaper when M is small because it touches only the selected indices, whereas the recurrence performs an $O(K)$ sweep. As M grows, repeated arbitrary-precision binomial evaluations make hockey-stick decoding increasingly expensive, while the recurrence grows much more slowly. We therefore use hockey-stick coding for the low-M video regime and the recurrence for the broader image sweeps.

M	Recurrence rank (s)	Recurrence unrank (s)	Hockey-Stick rank (s)	Hockey-Stick unrank (s)	Rank ratio	Unrank ratio
43	0.034	0.049	0.0001	0.0008	0.004×	0.02×
67	0.033	0.050	0.0003	0.0022	0.008×	0.04×
84	0.039	0.055	0.0004	0.0035	0.01×	0.06×
128	0.041	0.060	0.0010	0.0091	0.02×	0.15×
219	0.051	0.071	0.0034	0.0320	0.07×	0.45×
373	0.063	0.083	0.0117	0.1153	0.19×	1.4×
638	0.088	0.107	0.0457	0.4502	0.52×	4.2×
1063	0.120	0.152	0.1709	1.6447	1.4×	10.8×
1559	0.161	0.190	0.4594	4.4245	2.9×	23.3×
2697	0.247	0.285	1.9003	18.291	7.7×	64.1×
4641	0.352	0.399	7.8415	75.405	22.3×	189.2×

Table S2: **Recurrence vs. hockey-stick, at $K = d = 2^{18}$ standalone rank/unrank cost, without model calls.** Ratios are hockey-stick over recurrence.

F COST OF AN EXPLICIT CODEBOOK

This appendix quantifies the dense-codebook bottleneck discussed in Sec. 3.2. Computing the correlations in (3) accesses all Kd codebook entries, so at every bit-carrying denoising step $\boldsymbol{C}_t$ must either be stored in memory or regenerated in tiles. Reconstruction of $\mathbf{z}_t^\star$ accesses only the M selected columns, but those columns originate from the same dense codebook.

In `fp16`, forming $\mathbf{u}_t = \boldsymbol{C}_t^\top \mathbf{r}_t$ requires approximately $2Kd$ FLOPs while reading $2Kd$ bytes. The operation is therefore strongly bandwidth-sensitive. More importantly, the codebook itself occupies $2Kd$ bytes: approximately 0.5 GiB for SD 2.1 with $K = d = 2^{14}$, 128 GiB for FLUX with $K = d = 2^{18}$, and 167 GiB for a CF video chunk with $K = d = 299{,}520$. At the larger scales, the complete codebook cannot reside alongside the backbone and must instead be regenerated and processed in blocks, increasing runtime.

Capping K reduces this cost but restricts the span of the available atoms. For $K < d$, components of the residual outside the codebook span cannot be represented, regardless of the number of selected atoms. Consistent with this limitation, our FLUX experiments show lower quality for the capped $K = 2^{14}$ Turbo-DDCM codebook than for the complete $K = d$ codebook at matched bitrate (Sec. 5). Thus, explicit Gaussian codebooks impose a memory-fidelity tradeoff that becomes increasingly restrictive as latent dimension grows.

G OVERCOMPLETE HADAMARD DICTIONARY

This appendix provides the full construction and rate-matching details for the overcomplete extension of Sec. 4.3, evaluated with $B = 8$ bases in the streaming-video experiments of Sec. 6.

Padded and cropped geometry. A CF latent chunk has true flattened dimension $d = 299{,}520$, whereas the FWHT operates at the next supported power-of-two dimension $D = 2^{19}$. Let $\mathbf{P} \in \{0,1\}^{d\times D}$ denote cropping to the true latent coordinates, so $\mathbf{P}^\top$ performs zero-padding and $\mathbf{P}\mathbf{P}^\top = \mathbf{I}_d$. In the padded domain, each $\mathbf{D}_{t,b}\mathbf{H} \in \mathbb{R}^{D\times D}$ is exactly orthogonal. The corresponding effective dictionary after cropping is

$$\mathbf{A}_{t,b} = \mathbf{P}\mathbf{D}_{t,b}\mathbf{H} \in \mathbb{R}^{d\times D}, \qquad \mathbf{u}_{t,b} = \mathbf{A}_{t,b}^\top \mathbf{r}_t = \mathbf{H}\mathbf{D}_{t,b}\mathbf{P}^\top \mathbf{r}_t, \tag{S13}$$

for $b = 1, \dots, B$, with $\mathbf{D}_{t,b} = \mathrm{diag}(\boldsymbol{\varepsilon}_{t,b})$ drawn from the shared seed exactly as in Sec. 4.1. The cropped dictionary has orthonormal rows, $\mathbf{A}_{t,b}\mathbf{A}_{t,b}^\top = \mathbf{I}_d$, but not orthonormal columns when $D > d$. Thus, the exact single-basis thresholding statement in Sec. 4.1 applies to the padded-domain objective; after cropping, the effective dictionary is overcomplete even before multiple randomized bases are concatenated.

Selection and reconstruction. For $b = 1, \dots, B$ let $\boldsymbol{\alpha}_{t,b}^\star \in \{-1, 0, +1\}^D$ collect basis b's coefficients. The encoder keeps the M largest-magnitude entries across the concatenated scores $\{\mathbf{u}_{t,1}, \dots, \mathbf{u}_{t,B}\}$, recording for each the basis it came from:

$$\mathcal{I}_t^{\mathrm{oc}} := \text{Top-M}\left(\{|(\mathbf{u}_{t,b})_j| : b \in [B],\, j \in [D]\}\right), \qquad (\boldsymbol{\alpha}_{t,b}^\star)_j = \begin{cases} \mathrm{sign}\left((\mathbf{u}_{t,b})_j\right), & (b,j) \in \mathcal{I}_t^{\mathrm{oc}}, \\ 0, & \text{otherwise}, \end{cases} \tag{S14}$$

where $[B] := \{1, \dots, B\}$ and $\mathcal{I}_t^{\mathrm{oc}}$ is now a set of (basis, coordinate) pairs, unlike the single-basis support $\mathcal{I}_t$ of Sec. 4.1. The decoder regenerates every $\mathbf{D}_{t,b}$ from the shared seed and reconstructs

$$\mathbf{y}_t = \sum_{b=1}^{B} \mathbf{P}\mathbf{D}_{t,b}\mathbf{H}\boldsymbol{\alpha}_{t,b}^\star, \qquad \mathbf{z}_t^\star = \mathbf{y}_t / \mathrm{std}(\mathbf{y}_t). \tag{S15}$$

Hierarchical top-M. A full sort over all BD scores is unnecessary when only the global top M are required. It is sufficient to retain each basis's local top M scores and then select the global top M from the resulting BM candidates.

Proposition 1. *Let $\mathcal{I}_t^{\mathrm{local}}$ be the union of each basis's own top-M indices, $|\mathcal{I}_t^{\mathrm{local}}| = BM$. The global top-$M$ of $\{|(\mathbf{u}_{t,b})_j|\}_{b,j}$ is exactly the top-M of $\{|(\mathbf{u}_{t,b})_j| : (b,j) \in \mathcal{I}_t^{\mathrm{local}}\}$.*

Proof. No basis contributes more than M entries to a global set of size M, so discarding a basis's entries below its own top-M cannot discard a global top-M member. □

This turns a sort over BD scores into B local top-M searches plus one search over $BM \ll BD$ candidates, and batches reconstruction across the active bases rather than looping over them.

Bitstream and rate matching. The overcomplete dictionary contains BD candidate atoms, so comparing it with the single-basis dictionary at the same M would allocate a larger bitrate budget to the overcomplete variant. We therefore compare the two at matched slot length. Using the grouped protocol of App. E, for the overcomplete variant, each randomized basis forms one group ($G = B$, $K_G = D$). For each single-basis configuration M_r, we choose the overcomplete budget M_{oc} whose grouped slot length is closest to the single-basis cost $\lceil \log_2 \binom{D}{M_r} \rceil + M_r C$. We note that since the realized per-basis counts $(m_1, \ldots, m_B)$ are unknown before the actual support is selected, we estimate the grouped slot length during rate matching using the balanced allocation $m_b \in \{\lfloor M_{\mathrm{oc}}/B \rfloor, \lceil M_{\mathrm{oc}}/B \rceil\}$. Table S3 lists the matched configuration pairs used in Sec. 6.

Reference H-DDCM M_r	Overcomplete M_{oc}	H-DDCM bits/slot	Overcomplete bits/slot
16	13	276	272
32	26	523	523
64	52	984	985
128	103	1,844	1,843
256	204	3,436	3,438
512	402	6,365	6,367
1,024	789	11,710	11,713
1,280	979	14,226	14,225
1,536	1,168	16,668	16,668
2,048	1,541	21,375	21,370

Table S3: **Overcomplete dictionary rate-matched configurations at $B = 8$, $D = 2^{19}$.** The overcomplete dictionary always needs strictly fewer atoms than the reference it matches, since each of its indices costs more to name.

H H-DDCM PSEUDO-CODE

Algorithms S1 and S2 summarize the single-trajectory compression and decompression shared by the image and video formulations, over one target $\mathbf{x}_{\text{src}}$. They omit the outer streaming-video segmentation and KV-cache refresh logic of Sec. 6, which is specified separately in App. B. The operator $\text{opt}(\mathbf{A}, \mathbf{x}, \hat{\mathbf{x}}, M)$ denotes the closed-form top-M threshold and coefficient quantization using the implicit dictionary $\mathbf{A}$ of Sec. 4.1; $\mathbf{A}$ is applied through the FWHT and is never materialized explicitly.

Algorithm S1 H-DDCM – Compression

Input: Target $\mathbf{x}_{\text{src}}$, trajectory length T, tail length N ($N = 0$ for the consistency formulation), atoms per step M, formulation $\in \{\text{DDPM}, \text{consistency}\}$, shared seed s.
Output: Compressed representation $E(\mathbf{x}_{\text{src}})$.

$E \leftarrow \emptyset$ ▷ Initialize the compressed representation
if formulation $=$ DDPM **then**
  $\mathbf{x}_T \leftarrow \text{rand_normal}(s)$ ▷ $\mathbf{x}_T$ generation; seed s shared with decoder
else ▷ Bootstrap selection counts as the first of the T bit-carrying consistency steps
  $\boldsymbol{\alpha} \leftarrow \text{opt}(\mathbf{A}_{T+1}, \mathbf{x}_{\text{src}}, \text{mean}(\mathbf{x}_{\text{src}}), M)$ ▷ Closed-form threshold, Eq. (3); proxy bootstrap against the source mean, App. D
  $\mathbf{x}_T \leftarrow \mathbf{A}_{T+1}\boldsymbol{\alpha} / \text{std}(\mathbf{A}_{T+1}\boldsymbol{\alpha})$ ▷ Eq. (4)
  $E \leftarrow E \,\|\, \text{pack}(\boldsymbol{\alpha})$ ▷ Lexicographic rank & sign bits, App. E; concatenated to the compressed representation
end if
for $t = T, \ldots, N+2$ **do**
  $\hat{\mathbf{x}}_{0|t} \leftarrow D(\mathbf{x}_t, t)$ ▷ Denoiser activation
  $\boldsymbol{\alpha} \leftarrow \text{opt}(\mathbf{A}_t, \mathbf{x}_{\text{src}}, \hat{\mathbf{x}}_{0|t}, M)$ ▷ Closed-form threshold against the residual, Eq. (3)
  $\mathbf{z}^\star \leftarrow \mathbf{A}_t\boldsymbol{\alpha} / \text{std}(\mathbf{A}_t\boldsymbol{\alpha})$ ▷ Rescaled to unit variance, Eq. (4)
  $\mathbf{x}_{t-1} \leftarrow \boldsymbol{\mu}_t(\mathbf{x}_t) + \sigma_t \mathbf{z}^\star$ ▷ Eq. (1)
  $E \leftarrow E \,\|\, \text{pack}(\boldsymbol{\alpha})$ ▷ Lexicographic rank & sign bits, App. E; concatenated to the compressed representation
end for
return $E(\mathbf{x}_{\text{src}})$

Algorithm S2 H-DDCM – Decompression

Input: Compressed representation E, trajectory length T, tail length N, formulation $\in \{\text{DDPM}, \text{consistency}\}$, shared seed s.
Output: Decompressed target $\tilde{\mathbf{x}}_0$.

if formulation $=$ DDPM **then**
  $\tilde{\mathbf{x}}_T \leftarrow \text{rand_normal}(s)$ ▷ identical to the encoder's $\mathbf{x}_T$, same shared seed s
else
  $\boldsymbol{\alpha} \leftarrow \text{unpack}(E)$ ▷ Reads the bootstrap support & signs the encoder packed
  $\tilde{\mathbf{x}}_T \leftarrow \mathbf{A}_{T+1}\boldsymbol{\alpha} / \text{std}(\mathbf{A}_{T+1}\boldsymbol{\alpha})$ ▷ Identical to the encoder
end if
for $t = T, \ldots, N+2$ **do**
  $\hat{\mathbf{x}}_{0|t} \leftarrow D(\tilde{\mathbf{x}}_t, t)$ ▷ Denoiser activation
  $\boldsymbol{\alpha} \leftarrow \text{unpack}(E)$ ▷ Reads this step's support & signs from the bitstream
  $\mathbf{z}^\star \leftarrow \mathbf{A}_t\boldsymbol{\alpha} / \text{std}(\mathbf{A}_t\boldsymbol{\alpha})$ ▷ Rescaled to unit variance, Eq. (4)
  $\tilde{\mathbf{x}}_{t-1} \leftarrow \boldsymbol{\mu}_t(\tilde{\mathbf{x}}_t) + \sigma_t \mathbf{z}^\star$ ▷ Eq. (1)
end for
if formulation $=$ DDPM **then**
  perform $N+1$ deterministic DDIM steps ▷ Eq. (S2)
else
  perform the final deterministic sampling step ▷ Eq. (S3)
end if
return $\tilde{\mathbf{x}}_0$

See Secs. 5 and 6, together with Apps. A and B, for the configurations used in practice.