ELASTIC-QVG: EFFICIENT LONG VIDEO GENERATION WITH ELASTIC KV-CACHE QUANTIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Auto-regressive video generation uses a growing KV-cache to retain long-range context. Existing KV-cache quantization reduces the memory cost per token but still allows total memory usage to grow with context length, whereas eviction bounds memory by discarding cached context entirely. To bridge this gap, we present Elastic-QVG, a training-free framework that progressively lowers cache precision without re-quantizing quantized caches. A bit-level sliding window extends conventional token-level windowing to individual bit-planes, keeping recent chunks at higher precision while truncating older chunks to fit a fixed memory budget. Each chunk is quantized once, and its precision can subsequently decrease as new context arrives or the available memory budget shrinks. A truncation-exact quantizer enables hardware-friendly truncation directly on the quantized KV-cache, with no additional error relative to direct lower-bit quantization. Its nested quantization grids make dropping bit-planes exact, while separately allocated bit-plane storage and fused dequantization turn this property into memory savings without repacking the cache. Across LongCat-Video, HY-WorldPlay, and LingBot-World, Elastic-QVG improves fidelity over static QVG and retains VBench scores close to BF16 at 13–15$\times$ compression. It reduces KV memory by up to $12.7\times$ with less than 5% latency overhead on H100, and avoids KV offloading on an RTX 5090 to achieve a $4.3\times$ speedup over offloaded BF16-cache inference.

1 INTRODUCTION

Auto-regressive video generation supports interactive world simulation by generating new frames while retaining earlier frames in a KV-cache (Yin et al., 2025; Huang et al., 2025). Actions or camera motion can drive the rollout online (He et al., 2025; Tencent HunyuanWorld Team, 2025; Sun et al., 2025), while historical context helps preserve identity, layout, and motion (Hong et al., 2025; Robbyant Team et al., 2026). Retaining this history requires a cache whose memory grows linearly with context (Team et al., 2025). The resulting bottleneck couples the system and algorithm because device capacity limits both deployment feasibility and how much history the model can use.

Existing methods reduce KV-cache memory in two rigid ways. *Quantization* stores every cached token at a fixed low bit-width (Liu et al., 2024; Hooper et al., 2024; Xi et al., 2026). This compresses all cached tokens to reduce memory, but the footprint still grows linearly with context, so even an aggressive INT2 cache eventually exceeds the device budget. *Sliding-window eviction* keeps recent tokens and abruptly removes older ones (Yin et al., 2025; Yang et al., 2025). This caps memory by completely discarding tokens outside the window, making their information unavailable to later frames. A growing context instead calls for gradually reducing the precision of older history, rather than keeping its precision fixed forever or discarding it at once.

This rigidity creates two practical gaps. First, fixed write-time precision cannot use spare memory early in generation or accommodate longer context once the budget is reached. Second, a statically quantized cache targets one operating point, whereas Cloud GPUs and Edge GPUs have different memory capacities. Adapting to a different budget should not require re-quantization from a full-precision source.

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.

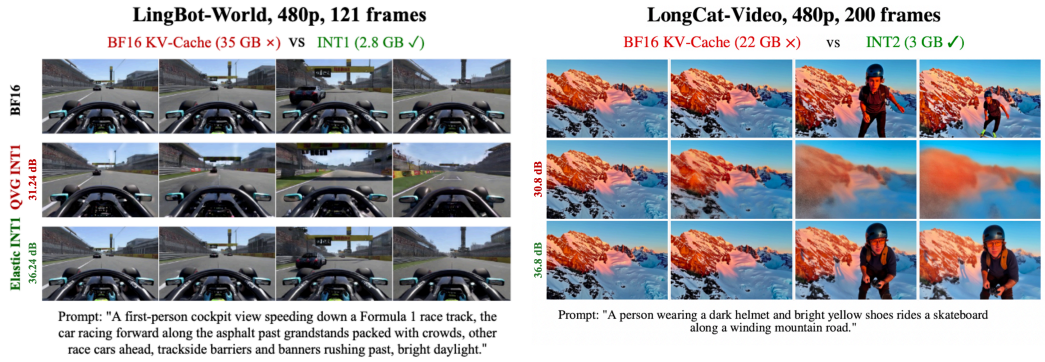

Figure 1: **Video quality with low-bit KV caches.** Representative frames compare BF16, static QVG, and Elastic-QVG on LingBot-World at INT1 (left) and LongCat-Video at INT2 (right). PSNR values and KV-cache footprints are shown in each panel.

Adapting to these budgets requires more than choosing a different bit-width before generation. Precision must remain adjustable after a chunk has been written, even when its original full-precision KV values are no longer available. How the KV-cache is quantized therefore determines whether this adjustment is possible. If truncating bits exactly reproduces direct lower-width quantization, the cache can be resized without its source values or additional conversion error. Precision then becomes a *runtime memory resource* that decreases as cached chunks age.

Elastic-QVG realizes this idea with a **bit-level sliding window** that truncates older cached chunks along a discrete precision ladder under a fixed budget. Where a conventional sliding window removes older tokens entirely, the bit-level sliding window truncates the least significant bits (LSBs) of older chunks while retaining recent chunks at higher precision. Zero bits recovers eviction, making quantization and eviction endpoints of the same precision axis.

This window requires exact, inexpensive truncation, which standard round-to-nearest quantization does not generally support. Re-quantizing reconstructed values adds computation and memory traffic without recovering the original source. Our truncation-exact quantizer instead uses nested integer grids so that truncation operates directly on the quantized KV-cache, while companding improves near-zero resolution without breaking this property (Xi et al., 2025). A lower bit-width saves resident memory only if the discarded bits can actually be released. We therefore store bit-planes separately and implement a fused GPU kernel for unpacking and dequantization, allowing the bit-level window to reduce the allocated cache without repacking its values.

We summarize our contributions as follows.

- **A bit-level sliding window for a progressively truncatable KV-cache.** We recast KV-cache compression as runtime precision allocation under a fixed memory budget. Cached chunks are written once, then lose least-significant bits as they age. Recent context retains higher precision, while older history remains available at fewer bits, with eviction as the zero-bit endpoint.
- **Exact precision reduction without the source values.** We construct a prefix-nested integer representation whose truncation exactly matches direct lower-width quantization using the same block scale. Truncation therefore introduces no additional conversion error and needs neither the original full-precision values nor per-value arithmetic. A companding warp improves near-zero resolution while preserving exactness.
- **Physical memory reclamation through algorithm–system co-design.** We couple truncation-exact quantization with separately allocated bit-planes and fused GPU kernels for packing and dequantization. Truncation frees memory without rewriting retained values, keeping total overhead below 5% on H100 while enabling a $4.3\times$ speedup on RTX 5090 by avoiding KV offloading.

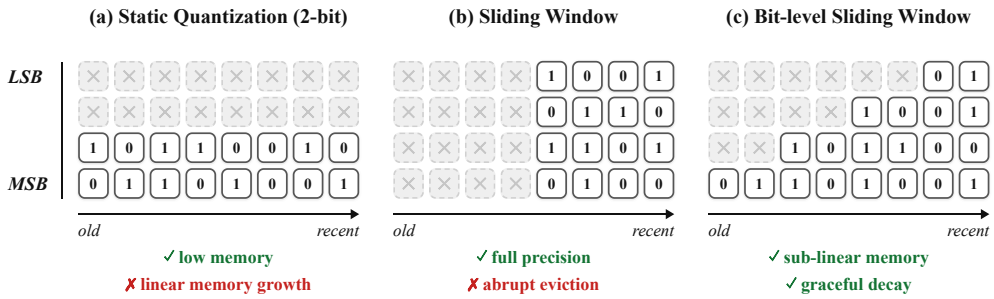

Figure 2: **KV-cache allocation at the bit level.** Columns represent tokens from old to recent; rows represent bits, with the MSB at the bottom. **(a)** Fixed precision. **(b)** Sliding-window eviction. **(c)** Progressive precision reduction with a bit-level sliding window.

2 Related Work

Auto-regressive video generation and memory. For auto-regressive video generation, the on-device memory that holds the KV-cache directly bounds how much history a model can attend to (Hong et al., 2025; Team et al., 2025), making cache size a first-order constraint on capability rather than merely throughput. The bound has grown sharper as generation shifts from bidirectional, clip-level denoising (Wan et al., 2025; Wu et al., 2025) toward chunk-level auto-regressive generation (Yin et al., 2025; Huang et al., 2025) driven by long-horizon synthesis (Bruce et al., 2024; Tencent HunyuanWorld Team, 2025) and real-time interaction (Feng et al., 2025), where frames are committed and streamed as soon as they are produced (Tencent HunyuanWorld Team, 2025; Sun et al., 2025; He et al., 2025; Shin et al., 2025) and the cache itself becomes durable world memory (Robbyant Team et al., 2026). In contrast to horizon-extending methods that act on the generation process—noise rescheduling and frequency extrapolation (Qiu et al., 2023; Lu et al., 2024; Zhao et al., 2025), diffusion–causal hybrids (Chen et al., 2024; Song et al., 2025), and explicit memory or chunked rollouts (Henschel et al., 2025; Xiao et al., 2025; Zhang et al., 2025b)—we study the orthogonal cache-compression axis that governs how much of that history a device can afford to keep.

Compressing the KV-cache by quantization and eviction. Existing methods reduce cache memory either by compressing retained tokens or by removing them. *Quantization* follows the first approach. KIVI (Liu et al., 2024) and KVQuant (Hooper et al., 2024) exploit per-channel and per-token statistics with outlier handling, rotation-based methods (Ashkboos et al., 2024; Su et al., 2025) smooth distributions before rounding, and vector-quantization methods (Zhang et al., 2025a; Li et al., 2025) represent tokens with learned dictionaries. QVG (Xi et al., 2026) applies low-bit quantization to video generation using semantic-aware smoothing as preprocessing. These methods reduce storage per token, whereas *eviction* and *windowing* limit how many tokens remain in the cache. Examples include StreamingLLM (Xiao et al., 2024), H2O (Zhang et al., 2023), and Scissorhands (Liu et al., 2023) for LLMs, and rolling or sliding windows with sink tokens for auto-regressive video (Yin et al., 2025; Yang et al., 2025; Liu et al., 2025; Shin et al., 2025). Elastic-QVG connects the two approaches by lowering precision as cached chunks move behind the recent window, with eviction as the 0-bit endpoint. It is agnostic to the pre-quantization transform and uses QVG's smoothing as one instantiation.

Elastic precision via a truncation-ready quantizer. Changing cache precision after storage is related to scalable and progressive media compression (Schwarz et al., 2007), where one bitstream serves multiple qualities, and to any-precision network quantization (Yu et al., 2021), where one stored tensor supports several bit-widths. TruncQuant (Kim et al., 2025) further trains weights for truncation-ready quantization. These ideas motivate a representation whose precision can decrease without repeated quantization, but a streaming KV-cache differs from fixed model weights because new values arrive continuously during generation. Elastic-QVG addresses this setting with a training-free, truncation-exact quantizer and a bit-level sliding window, allowing the cache footprint to adapt as the context grows.

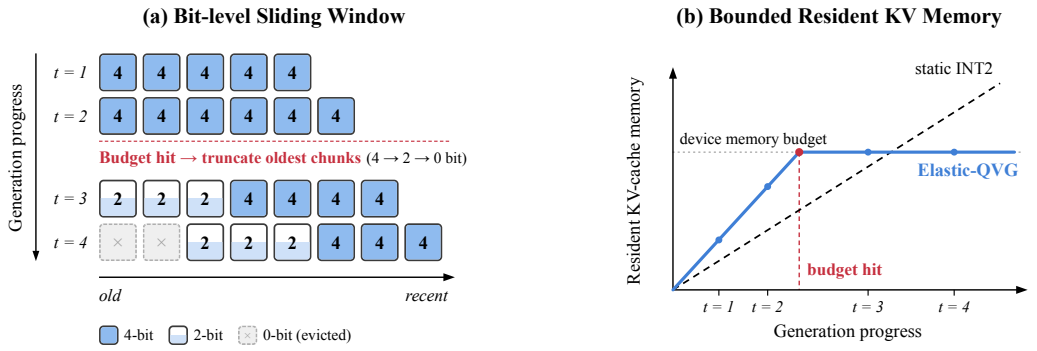

Figure 3: **Bit-level sliding window. (a)** Older chunks are truncated as new chunks arrive; the initial window contains five chunks. **(b)** Truncation keeps KV-cache memory within a fixed budget, while static quantization grows with context.

3 METHOD

Elastic-QVG quantizes each KV-cache chunk once, then reduces its precision as needed during generation. We use QVG's Semantic-Aware Smoothing as preprocessing with its settings unchanged; the design below concerns cache quantization, precision allocation, and storage.

After introducing the quantization notation (§3.1), we describe three components. The bit-level sliding window selects chunks for precision reduction under a memory budget (§3.2); the truncation-exact quantizer makes bit truncation match direct lower-bit quantization (§3.3); and bit-major storage with fused read kernels turns these reductions into memory savings without per-value conversion during truncation (§3.4).

3.1 QUANTIZATION PRELIMINARIES

Let $\mathbf{X}$ be a nonzero block of KV values, $\boldsymbol{Q}_{\mathbf{X}}$ its integer representation, and $Q(\mathbf{X})$ its reconstruction. For symmetric b-bit round-to-nearest (RTN) quantization,

$$\boldsymbol{Q}_{\mathbf{X}} = \text{round}\left(\frac{\mathbf{X}}{s_{\mathbf{X}}}\right), \qquad s_{\mathbf{X}} = \frac{\max|\mathbf{X}|}{q_{\max}}, \qquad Q(\mathbf{X}) = s_{\mathbf{X}}\boldsymbol{Q}_{\mathbf{X}}. \tag{1}$$

Here $q_{\max} = 2^{b-1} - 1$ for $b \geq 2$, and $s_{\mathbf{X}}$ is the quantization step size. The cache stores $\boldsymbol{Q}_{\mathbf{X}}$ and its scale rather than the full-precision values. We use lowercase $q_b(x)$ for a scalar integer index and $Q_b(x)$ for its reconstructed value, with the bit-width made explicit.

Reconstruction conventions. A zero-centered uniform *mid-tread* quantizer has reconstruction levels at integer multiples of its step size, including zero, as in Eq. (1). A uniform *mid-rise* quantizer places zero at a bin boundary and reconstructs at half-integer multiples, with no zero level. These terms describe reconstruction conventions, not whether stored indices support exact bit truncation.

3.2 THE BIT-LEVEL SLIDING WINDOW

The bit-level sliding window extends a sliding window from tokens to individual bit-planes of cached chunks. Stored bit-widths live on a fixed ladder $\mathcal{L} = \{b_1 > b_2 > \cdots > b_J\}$ with $J = |\mathcal{L}|$ tiers, where $b_J = 0$ denotes eviction. Each newly written chunk enters at the top width b_1. Once the resident cache reaches the memory budget, the window frees space by truncating older chunks one tier down the ladder, dropping their lowest bit-planes (Figure 3).

The window is monotone and recency-preserving. A chunk's width never rises after it is written, and when the budget is exceeded the window truncates the oldest chunk among those still at the highest current width. Recent chunks retain higher precision; zero bits denotes eviction. Algorithm 1 gives the full sweep. To implement intermediate precision reductions without re-quantization, we require the quantized KV-cache to support exact truncation.

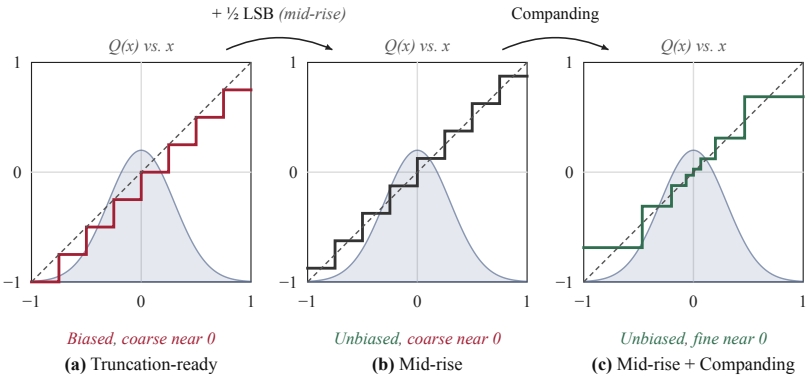

Figure 4: **Constructing the truncation-exact quantizer.** **(a)** Prefix-nested floor indices support truncation; lower-edge reconstruction has a one-sided offset. **(b)** A half-step shift gives mid-rise reconstruction at bin centers without changing the indices. **(c)** Companding applies this grid in a warped domain, concentrating levels near 0 in the original domain while preserving the prefix relation.

3.3 Truncation-Exact Quantizer

Why direct bit truncation fails. The window requires precision reduction after a chunk has been stored. For the RTN quantizer in §3.1, however, changing the bit-width changes both the grid spacing and rounding boundaries. For a block with $\max|\mathbf{X}| = 1$, $x = 0.2$ gives $q_4(x) = 1 = (0001)_2$. Dropping one bit leaves 0, whereas direct 3-bit quantization gives $q_3(x) = 1$. Thus, truncation need not reproduce direct lower-bit quantization; TruncQuant (Kim et al., 2025) also studies this mismatch for weights.

Unpacking, dequantizing, and re-quantizing the cache instead requires arithmetic and memory reads and writes for every affected value, plus temporary storage if reconstructions are materialized. Re-quantizing these reconstructions also need not match quantizing the original values, which are no longer retained. Keeping a full-precision copy would undermine the memory savings. We therefore seek indices that can be truncated directly while reproducing direct quantization at the target bit-width.

Exactness under truncation. Let $u_n(x)$ denote the stored n-bit unsigned integer index of a scalar cache value x. Exact truncation requires the lower-bit index to be a prefix of the higher-width one:

$$u_{n-k}(x) \;=\; \Big\lfloor u_n(x)/2^{\,k} \Big\rfloor \;=\; u_n(x) \gg k, \qquad 0 \le k < n. \tag{2}$$

Following TruncQuant (Kim et al., 2025), we call this property *truncation-ready*. Our cache satisfies it by construction for every stored value, allowing truncation without accessing x or re-quantizing.

Prefix-nested indices with mid-rise reconstruction. We separate index construction from reconstruction (Figure 4). Floor-based indices on a fixed normalized interval have the required prefix structure. Reconstructing at the lower bin edges, however, gives a one-sided offset. We instead use mid-rise reconstruction at bin centers; this changes the reconstructed values without changing the stored indices. For each nonzero 64-element block, we store its maximum magnitude $a_{\mathbf{X}} = \max|\mathbf{X}|$ and use the step size $s_{\mathbf{X},n} = a_{\mathbf{X}}/2^{n-1}$. The signed integer index and reconstructed value are

$$q_n(x) \;=\; \mathrm{clip}\Big(\big\lfloor x/s_{\mathbf{X},n} \big\rfloor,\; -2^{\,n-1},\; 2^{\,n-1}-1\Big), \qquad Q_n(x) \;=\; s_{\mathbf{X},n}\big(q_n(x) + \tfrac{1}{2}\big). \tag{3}$$

The $+\frac{1}{2}$-bin shift centers reconstruction within each bin; it does not itself guarantee exact truncation. To establish the prefix relation, we keep $a_{\mathbf{X}}$ fixed across bit-widths and store the unsigned index $u_n = q_n(x) + 2^{n-1}$. For $v = \frac{1}{2}(x/a_{\mathbf{X}} + 1) \in [0, 1)$, $u_n = \lfloor 2^n v \rfloor$ holds the first n fractional bits of v; at $v = 1$, clipping gives $u_n = 2^n - 1$. In both cases, dropping bits gives

$$u_n \gg k \;=\; u_{n-k}, \qquad 0 \le k < n, \tag{4}$$

with the boundary case derived in Appendix C. Since the packer stores exactly u_n, Eq. (4) is precisely the truncation-ready requirement of Eq. (2), letting Algorithm 1 truncate the quantized KV-cache directly.

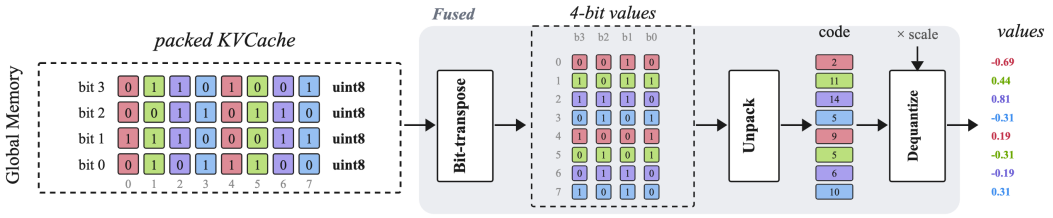

Figure 5: **Fused read kernel.** Bit-plane unpacking and dequantization reconstruct KV values in one launch, without an intermediate index tensor.

Companding. The uniform mid-rise grid is truncation-ready with these indices, but it under-resolves the near-zero region where the smoothed values concentrate. We propose a companded quantizer that improves near-zero resolution without breaking the exact truncation relation. It applies a fixed, strictly monotone amplitude warp before quantization and inverts it after reconstruction:

$$g(x) = \operatorname{sign}(x)\,|x|^{p}, \quad 0 < p \le 1, \qquad q_n^{\mathrm{c}}(x) = q_n\big(g(x)\big), \qquad Q_n^{\mathrm{c}}(x) = g^{-1}\big(Q_n(g(x))\big), \tag{5}$$

where $g^{-1}(w) = \operatorname{sign}(w)\,|w|^{1/p}$, and q_n and Q_n use the warped block's maximum $a_{g(\mathbf{X})}$. The grid is uniform and mid-rise in the warped domain; its levels are nonuniform in the original domain for $p < 1$. With $p < 1$ the warp expands small amplitudes before quantization, moving quantization levels toward the origin; $p{=}1$ recovers the unwarped grid. Using the same warp and block maximum at every bit-width preserves the prefix relation. In unsigned form, $u_n^{\mathrm{c}} = q_n^{\mathrm{c}}(x) + 2^{n-1}$ therefore satisfies

$$u_{n-k}^{\mathrm{c}} \;=\; u_n^{\mathrm{c}} \gg k, \qquad 0 \le k < n. \tag{6}$$

3.4 Efficient Algorithm–System Co-design

Algorithm–system co-design is essential for a truncation-friendly bit-level sliding window with no per-value conversion during truncation. Separate bit-plane allocations avoid re-quantization and repacking; a fused read kernel reconstructs the retained KV values (Figure 5).

Bit-major packing. The write kernel quantizes each block and transposes its unsigned indices into byte-packed bit-planes (Appendix D.1). Each byte holds one bit from eight consecutive values. Register-level shifts and masks perform the transpose within the quantization kernel, avoiding an extra pass over the chunk. Each plane owns a separate allocation; $a_{\mathbf{X}}$ is stored alongside it.

Bit-plane truncation. To reduce a chunk from n to $e = n - k$ bits, the truncation sweep releases the k lowest planes (Figure 6). The remaining bytes and block maximum do not change. By Eq. (4), these planes already encode $u_e = u_n \gg k$. Separate allocations allow the dropped planes to be reclaimed; a slice of a single packed tensor would still hold its original storage. The sweep maintains a running footprint and batches plane reclamation; its cost is linear in the number of chunks for the fixed precision ladder.

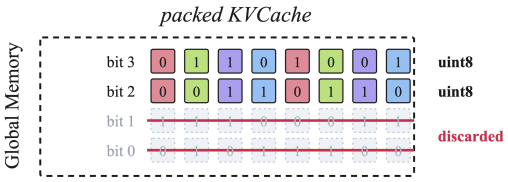

Figure 6: **Bit-plane truncation.** Reducing 4 bits to 2 releases the two lowest planes without rewriting the others.

Fused dequantization. The read kernel unpacks retained planes and reconstructs KV values in one launch (Figure 5), without an intermediate index tensor. It reads compressed indices and block maxima through an unrolled loop gated by the effective bit-width, so the same kernel handles chunks before and after truncation. With companding, the inverse warp is fused into the output store. The truncation sweep adds approximately 0.5% over static QVG, and total latency overhead stays below 5% relative to BF16 (§4.4). Packing equations and numerical examples are given in Appendix D.

Table 1: **Quality at comparable KV-cache footprints (480p).** Fidelity is measured against BF16; group-header rows give BF16 VBench scores. Bit labels denote target budgets for Elastic-QVG and fixed widths for baselines.

Method	Compression Ratio (BF16)	PSNR↑	SSIM↑	LPIPS↓	Background Consist.	Image Quality	Subject Consist.	Aesthetic Quality
LongCat-Video-13B		*2-bit KV Cache, 480p*			96.22	72.72	95.51	64.83
RTN	6.40×	20.87	0.719	0.203	84.84	59.60	70.63	43.38
KIVI	6.40×	20.31	0.719	0.208	84.84	38.10	75.25	41.58
QuaRot	6.40×	21.57	0.759	0.171	86.12	50.70	80.61	49.49
QVG	6.94×	28.71	0.909	0.065	95.06	71.47	94.11	62.22
Elastic-QVG	6.94×	**30.83**	**0.934**	**0.049**	95.91	72.10	94.88	63.71
LongCat-Video-13B		*1-bit KV Cache, 480p*			96.22	72.72	95.51	64.83
RTN	12.8×	14.83	0.518	0.401	73.20	37.45	57.80	31.60
QVG	13.5×	24.46	0.781	0.159	92.7	68.4	90.9	58.2
Elastic-QVG	13.5×	**29.40**	**0.890**	**0.082**	95.0	71.3	93.8	61.8
HY-WorldPlay-8B		*2-bit KV Cache, 480p*			97.92	74.33	97.90	69.85
RTN	6.40×	24.19	0.696	0.229	96.16	71.86	96.08	69.15
KIVI	6.40×	24.27	0.701	0.230	96.95	71.40	95.89	68.19
QuaRot	6.40×	25.20	0.738	0.205	97.34	72.26	96.64	69.38
QVG	7.05×	29.17	0.882	0.094	97.98	73.87	97.90	69.80
Elastic-QVG	7.05×	**31.42**	**0.921**	**0.070**	98.01	74.12	97.93	69.71
HY-WorldPlay-8B		*1-bit KV Cache, 480p*			97.92	74.33	97.90	69.85
RTN	12.8×	17.62	0.548	0.352	89.10	59.85	88.40	56.70
QVG	13.5×	24.13	0.769	0.166	95.4	70.9	95.7	67.1
Elastic-QVG	13.5×	**29.10**	**0.876**	**0.098**	97.9	73.6	97.8	69.5
LingBot-World		*2-bit KV Cache, 480p*			92.70	66.62	88.10	62.30
RTN	6.40×	27.10	0.498	0.425	87.50	61.40	79.80	56.90
QVG	7.28×	33.75	0.636	0.316	92.30	66.12	87.20	62.10
Elastic-QVG	7.28×	**36.84**	**0.726**	**0.230**	92.60	66.53	87.90	62.30
LingBot-World		*1-bit KV Cache, 480p*			92.70	66.62	88.10	62.30
RTN	12.8×	20.35	0.402	0.548	80.20	54.30	68.10	49.80
QVG	14.6×	31.24	0.537	0.436	90.80	65.51	84.60	62.30
Elastic-QVG	14.6×	**36.24**	**0.706**	**0.253**	92.30	66.33	87.40	62.30

4 EXPERIMENTS

We evaluate quality at comparable cache footprints, the precision–quality trade-off, and memory savings relative to execution overhead.

4.1 SETUP

Models and tasks. We evaluate three open auto-regressive video generators at 480p. LongCat-Video-13B (Team et al., 2025) continues video from a fixed 73-frame context and emits the next 20 frames. HY-WorldPlay-8B (Tencent HunyuanWorld Team, 2025; Sun et al., 2025) uses its full generated history in 12-frame chunks. LingBot-World (Robbyant Team et al., 2026) uses full-history attention for interactive generation. We also evaluate LingBot-World-v2 at 720p (Table 2); its additional 480p results are in Appendix F.

Metrics and baselines. PSNR, SSIM, and LPIPS measure agreement with BF16-generated videos, while VBench (Huang et al., 2024) reports Background Consistency, Image Quality, Subject Consistency, and Aesthetic Quality following QVG (Xi et al., 2026). We measure peak resident KV-cache compression and end-to-end latency relative to BF16. Baselines are RTN, KIVI (Liu et al., 2024), QuaRot (Ashkboos et al., 2024), and static QVG, which shares our smoothing preprocessing and provides the closest fixed-precision comparison.

Precision budgets. Chunks are written at 4 bits and truncated online through $\mathcal{L} = \{4, 3, 2, 1, 0\}$, using block size 64 and companding exponent $p = 0.6$. The 1- and 2-bit settings in Table 1 denote target storage budgets rather than a uniform bit-width, since the active window keeps recent chunks at higher precision while progressively truncating older ones. For planned rollouts, b_{target} and the

Table 2: **LingBot-World-v2 at 720p.** Evaluation on 100 MovieGen prompts (285 frames each), using the metrics of Table 1. Compression is the codec ratio over the quantized span; group-header rows report BF16 VBench scores.

Method	Compression Ratio (BF16)	PSNR↑	SSIM↑	LPIPS↓	Background Consist.	Image Quality	Subject Consist.	Aesthetic Quality
LingBot-World-v2		*2-bit KV Cache, 720p*			89.25	66.93	81.56	54.49
QVG	6.22×	24.64	0.674	0.271	89.19	66.45	80.94	53.60
Elastic-QVG	6.22×	**26.40**	**0.710**	**0.228**	89.27	66.84	81.56	54.52
LingBot-World-v2		*1-bit KV Cache, 720p*			89.25	66.93	81.56	54.49
QVG	10.40×	20.17	0.571	0.449	86.73	57.52	73.98	44.69
Elastic-QVG	10.40×	**23.34**	**0.643**	**0.314**	89.13	66.27	80.87	54.12

horizon determine a fixed cache ceiling; for open-ended LingBot-World, the ceiling is set by device capacity. Compression ratios include the measured resident cache footprint.

4.2 Quality at Comparable Storage

Table 1 compares fidelity and video quality at comparable cache footprints. At the 2-bit budget, Elastic-QVG improves PSNR over QVG by 2.12 dB on LongCat-Video, 2.25 dB on HY-WorldPlay, and 3.09 dB on LingBot-World. SSIM also increases and LPIPS decreases on all three models.

At 1 bit, Elastic-QVG exceeds QVG by approximately 5 dB on each backbone at 13.5×, 13.5×, and 14.6× compression, respectively. VBench remains close to BF16 across all four dimensions despite lower reconstruction fidelity than at 2 bits, indicating that perceptual quality is better preserved than the exact BF16 trajectory.

On LingBot-World-v2 at 720p (Table 2), Elastic-QVG improves PSNR over QVG by 1.76 dB at 2 bits and 3.17 dB at 1 bit, with higher SSIM and lower LPIPS at both budgets. VBench scores remain close to BF16, including at 10.40× codec compression.

Figure 1 provides visual comparisons; the appendix includes additional examples, quantizer behavior (Appendix B), and 480p LingBot-World-v2 results (Appendix F).

4.3 One Cache, Many Budgets

Figure 7 examines the quality trade-off across average precision budgets from 2 to 0.5 bits. We vary b_{target} using the same truncatable KV-cache; lower-bit caches are obtained by dropping bit-planes rather than re-quantizing from a full-precision cache.

VBench stays close to BF16 at approximately 1 bit across all three models, with clearer quality loss below this point. Static QVG is shown at its reported operating point, not as an exhaustive codec sweep. Unlike a fixed-precision cache, Elastic-QVG supports these budget reductions on the already quantized KV-cache during generation.

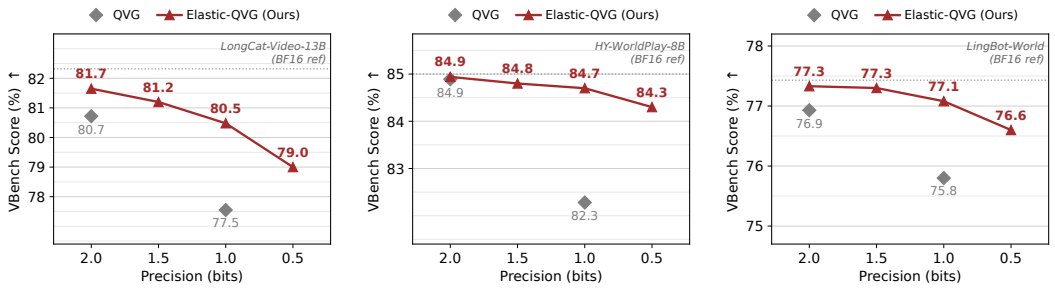

Figure 7: **Quality across precision budgets.** Mean of the four reported VBench dimensions versus average cache precision. Elastic-QVG remains close to BF16 at approximately 1 bit; gray diamonds show the reported static QVG baseline.

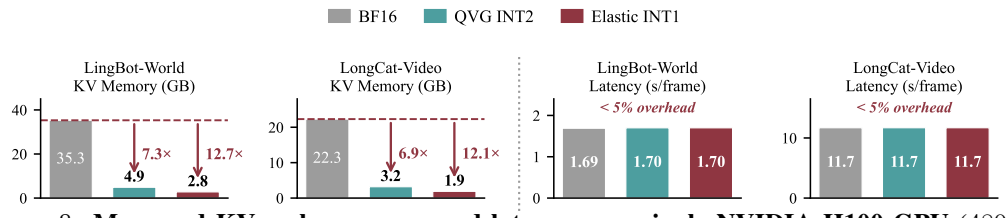

Figure 8: **Measured KV-cache memory and latency on a single NVIDIA H100 GPU** (480p). **(a–b)** Elastic INT1 cuts resident KV-cache memory by up to 12.7×, below QVG INT2. **(c–d)** Per-frame latency remains within 5% of the baselines.

4.4 Efficiency

KV-cache quantization introduces additional packing and dequantization operations, and changing precision online could add further overhead. We therefore test whether the memory savings of Elastic-QVG outweigh this cost, both when the BF16 cache fits on the device and when it requires offloading.

Cloud GPU (H100). Elastic INT1 reduces KV memory by 12.7× on LingBot-World and 12.1× on LongCat-Video, while keeping latency overhead below 5% (Figure 8). The bit-major layout releases planes without reconstructing or requantizing values, so the truncation sweep adds only about 0.5% over static QVG with the same fused read kernel. Most of the remaining overhead comes from accessing the quantized cache, rather than changing its precision.

Edge GPU (RTX 5090, 32 GB). FP8 W+A with a BF16 KV-cache still OOMs (Figure 9). Elastic INT1 fits in 26GB, avoids offloading, and cuts latency from 2.35s/step to 0.55s/step (4.3×). Here, eliminating transfers outweighs the added quantization operations.

Figure 10 shows that BF16 and static QVG INT2 grow with context length, whereas Elastic-QVG remains near 31GB by truncating older entries, enabling long-context generation within a fixed Edge GPU budget. This sweep measures memory as context grows; Figure 7 separately reports the quality trade-off across precision budgets. B300 latency measurements and the comparison with static QVG are provided in Appendix E.

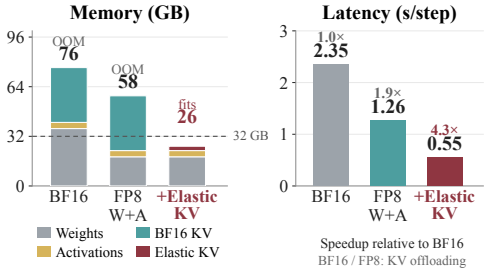

Figure 9: **LingBot-World on a single RTX 5090 GPU.** Elastic INT1 fits in 26 GB, avoids KV offloading, and delivers a 4.3× speedup.

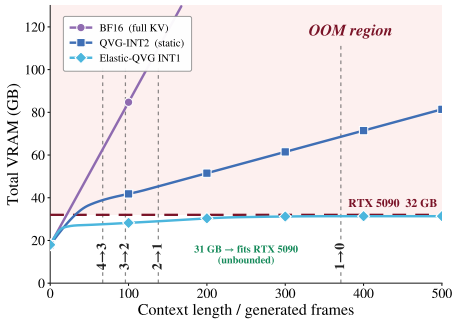

Figure 10: **Context-length scalability.** Elastic-QVG stays within 31 GB by progressively truncating older KV entries, while BF16 and static QVG grow with context length.

5 Conclusion

We presented Elastic-QVG, a training-free framework that treats KV-cache precision as a runtime memory resource rather than a fixed write-time choice. A bit-level sliding window reduces the precision of older chunks under a fixed budget, with eviction as its zero-bit endpoint. Prefix-nested quantization and bit-major storage make these updates exact without re-quantizing the cached values. Across three video generators, Elastic-QVG improves fidelity over static QVG at comparable footprints and retains VBench scores close to BF16 at the reported 1-bit budgets. It reduces KV memory by up to 12.7× on an H100 GPU with less than 5% latency overhead, and enables a 4.3× speedup over offloaded BF16-cache inference on an RTX 5090 GPU.

AI USE STATEMENT

We used generative AI tools to assist with grammar, language editing, and sentence-level phrasing. We did not use generative AI tools for drafting or finalizing the manuscript. All research ideas and experimental designs were developed without the use of generative AI. Generative AI tools were also used to assist with verifying mathematical derivations related to the experiments. All AI-assisted content was reviewed and verified by the authors, who take full responsibility for the final content of this work.

REFERENCES

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240, 2024.

Jake Bruce, Michael Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Bechtle, Feryal Behbahani, Stephanie Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando de Freitas, Satinder Singh, and Tim Rocktäschel. Genie: Generative interactive environments, 2024. URL https://arxiv.org/abs/2402.15391.

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

Tianrui Feng, Zhi Li, Shuo Yang, Haocheng Xi, Muyang Li, Xiuyu Li, Lvmin Zhang, Keting Yang, Kelly Peng, Song Han, Maneesh Agrawala, Kurt Keutzer, Akio Kodaira, and Chenfeng Xu. Streamdiffusionv2: A streaming system for dynamic and interactive video generation, 2025. URL https://arxiv.org/abs/2511.07399.

Xianglong He, Chunli Peng, Zexiang Liu, Boyang Wang, Yifan Zhang, Qi Cui, Fei Kang, Biao Jiang, Mengyin An, Yangyang Ren, Baixin Xu, Hao-Xiang Guo, Kaixiong Gong, Size Wu, Wei Li, Xuchen Song, Yang Liu, Yangguang Li, and Yahui Zhou. Matrix-game 2.0: An open-source real-time and streaming interactive world model, 2025. URL https://arxiv.org/abs/2508.13009.

Roberto Henschel, Levon Khachatryan, Hayk Poghosyan, Daniil Hayrapetyan, Vahram Tadevosyan, Zhangyang Wang, Shant Navasardyan, and Humphrey Shi. Streamingt2v: Consistent, dynamic, and extendable long video generation from text, 2025. URL https://arxiv.org/abs/2403.14773.

Yicong Hong, Yiqun Mei, Chongjian Ge, Yiran Xu, Yang Zhou, Sai Bi, Yannick Hold-Geoffroy, Mike Roberts, Matthew Fisher, Eli Shechtman, Kalyan Sunkavalli, Feng Liu, Zhengqi Li, and Hao Tan. Relic: Interactive video world model with long-horizon memory, 2025. URL https://arxiv.org/abs/2512.04040.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, Yaohui Wang, Xinyuan Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. VBench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21807–21818, 2024.

Jinhee Kim, Seoyeon Yoon, Taeho Lee, Joo Chan Lee, Kang Eun Jeon, and Jong Hwan Ko. TruncQuant: Truncation-ready quantization for DNNs with flexible weight bit precision. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2025. arXiv:2506.11431.

Junyan Li, Yang Zhang, Muhammad Yusuf Hassan, Talha Chafekar, Tianle Cai, Zhile Ren, Pengsheng Guo, Foroozan Karimzadeh, Colorado Reed, Chong Wang, and Chuang Gan. Commvq: Commutative vector quantization for kv cache compression, 2025. URL https://arxiv.org/abs/2506.18879.

Kunhao Liu, Wenbo Hu, Jiale Xu, Ying Shan, and Shijian Lu. Rolling forcing: Autoregressive long video diffusion in real time, 2025. URL https://arxiv.org/abs/2509.25161.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time, 2023. URL https://arxiv.org/abs/2305.17118.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*, 2024.

Yu Lu, Yuanzhi Liang, Linchao Zhu, and Yi Yang. Freelong: Training-free long video generation with spectralblend temporal attention. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Haonan Qiu, Menghan Xia, Yong Zhang, Yingqing He, Xintao Wang, Ying Shan, and Ziwei Liu. Freenoise: Tuning-free longer video diffusion via noise rescheduling, 2023.

Robbyant Team, Zelin Gao, Qiuyu Wang, Yanhong Zeng, Jiapeng Zhu, Ka Leong Cheng, Yixuan Li, Hanlin Wang, Yinghao Xu, Shuailei Ma, et al. Advancing open-source world models. *arXiv preprint arXiv:2601.20540*, 2026.

Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. Overview of the scalable video coding extension of the H.264/AVC standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1103–1120, 2007. doi: 10.1109/TCSVT.2007.905532.

Joonghyuk Shin, Zhengqi Li, Richard Zhang, Jun-Yan Zhu, Jaesik Park, Eli Shechtman, and Xun Huang. Motionstream: Real-time video generation with interactive motion controls, 2025. URL https://arxiv.org/abs/2511.01266.

Kiwhan Song, Boyuan Chen, Max Simchowitz, Yilun Du, Russ Tedrake, and Vincent Sitzmann. History-guided video diffusion, 2025. URL https://arxiv.org/abs/2502.06764.

Zunhai Su, Zhe Chen, Wang Shen, Hanyu Wei, Linge Li, Huangqi Yu, and Kehong Yuan. Rotatekv: Accurate and robust 2-bit kv cache quantization for llms via outlier-aware adaptive rotations. *arXiv preprint arXiv:2501.16383*, 2025.

Wenqiang Sun, Haiyu Zhang, Haoyuan Wang, Junta Wu, Zehan Wang, Zhenwei Wang, Yunhong Wang, Jun Zhang, Tengfei Wang, and Chunchao Guo. WorldPlay: Towards long-term geometric consistency for real-time interactive world modeling. *arXiv preprint arXiv:2512.14614*, 2025.

Meituan LongCat Team, Xunliang Cai, Qilong Huang, Zhuoliang Kang, Hongyu Li, Shijun Liang, Liya Ma, Siyu Ren, Xiaoming Wei, Rixu Xie, et al. Longcat-video technical report. *arXiv preprint arXiv:2510.22200*, 2025.

Tencent HunyuanWorld Team. HY-World 1.5: A systematic framework for interactive world modeling with real-time latency and geometric consistency. Technical report, Tencent, 2025. https://3d-models.hunyuan.tencent.com/world/world1_5/HYWorld_1.5_Tech_Report.pdf.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Bing Wu, Chang Zou, Changlin Li, Duojun Huang, Fang Yang, Hao Tan, Jack Peng, Jianbing Wu, Jiangfeng Xiong, Jie Jiang, et al. Hunyuanvideo 1.5 technical report. *arXiv preprint arXiv:2511.18870*, 2025.

Haocheng Xi, Han Cai, Ligeng Zhu, Yao Lu, Kurt Keutzer, Jianfei Chen, and Song Han. Coat: Compressing optimizer states and activations for memory-efficient fp8 training. In *International Conference on Learning Representations*, volume 2025, pp. 42989–43011, 2025.

Haocheng Xi, Shuo Yang, Yilong Zhao, Muyang Li, Han Cai, Xingyang Li, Yujun Lin, Zhuoyang Zhang, Jintao Zhang, Xiuyu Li, Zhiying Xu, Jun Wu, Chenfeng Xu, Ion Stoica, Song Han, and Kurt Keutzer. Quant VideoGen: Auto-regressive long video generation via 2-bit KV-cache quantization. In *International Conference on Machine Learning (ICML)*, 2026.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL https://arxiv.org/abs/2309.17453.

Zeqi Xiao, Yushi Lan, Yifan Zhou, Wenqi Ouyang, Shuai Yang, Yanhong Zeng, and Xingang Pan. Worldmem: Long-term consistent world simulation with memory, 2025. URL https://arxiv.org/abs/2504.12369.

Shuai Yang, Wei Huang, Ruihang Chu, Yicheng Xiao, Yuyang Zhao, Xianbang Wang, Muyang Li, Enze Xie, Yingcong Chen, Yao Lu, Song Han, and Yukang Chen. Longlive: Real-time interactive long video generation, 2025. URL https://arxiv.org/abs/2509.22622.

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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22963–22974, 2025.

Haichao Yu, Haoxiang Li, Humphrey Shi, Thomas S. Huang, and Gang Hua. Any-precision deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 10763–10771, 2021.

Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. Pqcache: Product quantization-based kvcache for long context llm inference, 2025a. URL https://arxiv.org/abs/2407.12820.

Lvmin Zhang, Shengqu Cai, Muyang Li, Gordon Wetzstein, and Maneesh Agrawala. Frame context packing and drift prevention in next-frame-prediction video diffusion models, 2025b. URL https://arxiv.org/abs/2504.12626.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023. URL https://arxiv.org/abs/2306.14048.

Min Zhao, Guande He, Yixiao Chen, Hongzhou Zhu, Chongxuan Li, and Jun Zhu. Riflex: A free lunch for length extrapolation in video diffusion transformers. *arXiv preprint arXiv:2502.15894*, 2025.

Algorithm 1 Bit-Level Sliding Window: budgeted truncation sweep

Require: chunks $c_1, \ldots, c_M$ ordered oldest→newest; token counts n_i; usable bit-widths $\mathcal{L} = \{b_1 > b_2 > \cdots > b_J\}$, $J = |\mathcal{L}|$ (e.g. $\{4, 3, 2, 1, 0\}$); target width b_{target}; planned tokens N_{planned}

$\mathcal{B} \leftarrow b_{\text{target}} \cdot N_{\text{planned}}$ ▷ fixed quantized-cache budget
$F \leftarrow \sum_{i=1}^{M} \text{bits}(c_i) \cdot n_i$ ▷ compute current footprint once

for $j = 1$ **to** $J - 1$ **do** ▷ truncate tier $b_j \to b_{j+1}$, highest pair first
  if $F \leq \mathcal{B}$ **then break**
  for $i = 1$ **to** M **do** ▷ oldest (farthest-behind) chunk first
    if $F \leq \mathcal{B}$ **then break**
    if $\text{bits}(c_i) = b_j$ **then**
      if $b_{j+1} = 0$ **then**
        DROP(c_i) ▷ zero-bit endpoint
      else
        TRUNCATE(c_i, b_{j+1}) ▷ drop low planes; zero-copy (Eq. (4))
      end if
      $F \leftarrow F - (b_j - b_{j+1})\, n_i$ ▷ constant-time footprint update
    end if
  end for
end for
reclaim freed planes in one batched pass ▷ monotone: precision never rises

A Bit-Level Sliding Window: Full Sweep

Algorithm 1 scans precision tiers from highest to lowest and, within each tier, chunks from oldest to newest until the footprint meets $\mathcal{B}$. It computes $F = \sum_{i=1}^{M} \text{bits}(c_i) n_i$ once, then subtracts $(b_j - b_{j+1}) n_i$ whenever chunk i drops from b_j to b_{j+1}. Each budget check and update therefore takes $O(1)$ time. Initial accounting costs $O(M)$ and the sweep visits at most $(J-1)M$ chunk–tier pairs, where $J = |\mathcal{L}|$ is fixed to five for the ladder $\{4, 3, 2, 1, 0\}$. DROP and TRUNCATE update the chunk's bit-width and queue its released planes; chunk indices remain fixed until the final batched reclamation, which runs even when the budget is met early. This requires only scalar accounting, not reconstruction or re-quantization of cached KV values.

B Quantizer Comparison

Figure 11 compares QVG with the proposed truncation-exact quantizer under the same storage budget. The zero-centered mid-tread reconstruction shown for QVG includes an exact zero level. Our construction instead uses mid-rise reconstruction, with companding placing more levels near zero. Neither convention alone guarantees lower reconstruction error or exact truncation. In our quantizer, exactness follows from prefix-nested indices with the same normalization and warp at every bit-width (§3.3); dropping low-order bit-planes then reproduces direct lower-bit quantization. The quality comparison is reported in Table 1.

C Truncation-Exactness Lemma

Lemma. For $v \in [0, 1]$ and $0 \leq k < n$, $\min(\lfloor 2^n v \rfloor, 2^n - 1) \gg k = \min(\lfloor 2^{n-k} v \rfloor, 2^{n-k} - 1)$; substituting $v = \frac{1}{2}(x/a_{\mathbf{X}} + 1)$ proves Eq. (4), including the clipped endpoint.

D Kernel and System Implementation

The truncation-exact quantizer uses bit-major packing on the write path (Appendix D.1), fused unpacking and dequantization on the read path (Appendix D.2), and plane dropping under the bit-level sliding window (Appendix D.3, Appendix D.4). Throughout, $n \in \{1, 2, 3, 4\}$ is the stored width, $e \leq n$ the *effective* width of a chunk after truncating $k = n - e$ bits, and D (a multiple of 8) the

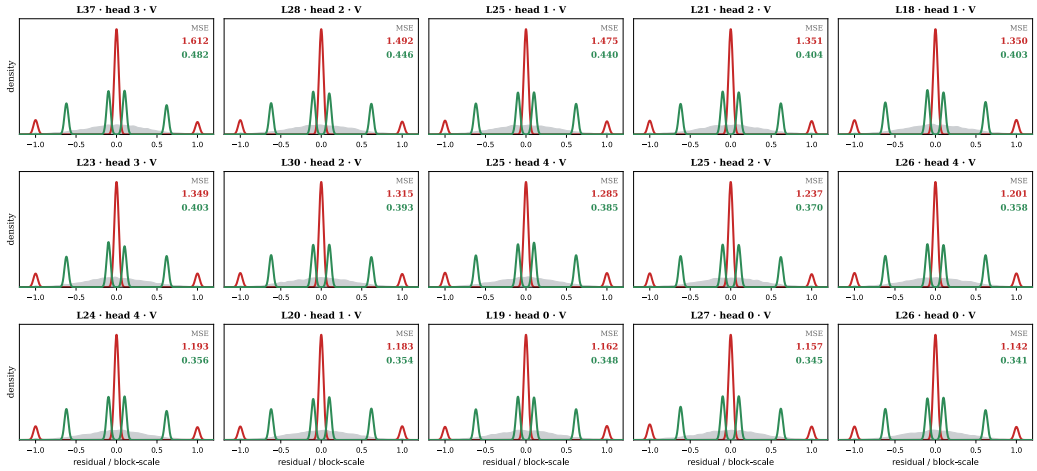

Figure 11: Quantizer comparison under 2-bit quantization

feature width. The examples below follow one 8-element group through the full round trip, using $n{=}4$ and a truncation to $e{=}2$ as the running example.

D.1 Bit-Major Packing (Write Path)

Figure 12 shows the write path. A newly generated chunk is quantized once, per 64-element block along the channel axis, and the signed index is recentered to unsigned, $u_n = q_n(x) + 2^{n-1}$. At this point the indices are still *value-major*, with the n bits $b_{n-1} \dots b_0$ of one element stored together, which is the layout that makes a later precision change expensive, since lowering the width would have to touch and rewrite every element. The `Bit-transpose` stage rotates this tile so that the stored object is one byte-packed plane per bit position. Grouping the feature axis by eight, plane b holds

$$P_b[\dots, g] \;=\; \sum_{i=0}^{7} \big((u_n)_{8g+i} \gg b \,\&\, 1\big) \;\ll\; (7-i), \tag{7}$$

so element i of the group sits in bit $7{-}i$ of the byte, and the n planes of the group are the n `uint8` rows written to global memory as the *packed KV-cache* on the right of Figure 12. The transpose is a register-level shift-and-mask over 32-bit words rather than a per-element scatter, so packing costs a handful of bitwise operations per word and runs fused with the quantizer, adding no extra pass over the chunk.

Two properties of this layout matter for truncation. First, the planes are a *list*, not a stacked tensor. Each plane owns its storage, so releasing the k lowest planes is a pointer drop that returns memory immediately, whereas a slice of a stacked tensor stays a view of the original allocation and would free nothing without a copy. Second, because the planes store exactly u_n, the top $n{-}k$ planes hold $u_n \gg k \;=\; u_{\,n-k}$ by Eq. (4)—the $(n{-}k)$-bit index, obtained without converting or consulting the original value.

D.2 Fused Unpacking and Dequantization (Read Path)

Figure 5 shows the read path at full width. Everything inside the shaded *Fused* region runs in a single kernel launch that loads the kept planes, applies the inverse bit-transpose to recover the per-element bits $b_{n-1} \dots b_0$, unpacks them into the integer index u_e, and dequantizes with the mid-rise rule of Eq. (3),

$$q_e \;=\; u_e - 2^{\,e-1}, \qquad Q_e(x) \;=\; s_{\mathbf{X},e}\big(q_e + \tfrac{1}{2}\big) \;=\; (2q_e + 1)\, 2^{-e}\, a_{\mathbf{X}}, \tag{8}$$

where $s_{\mathbf{X},e} = a_{\mathbf{X}}/2^{e-1}$. In the example, the first element's four planes give $u_4 = 2$, hence $q_4 = -6$ and $Q_4(x) = -11/16 \approx -0.69$ at $a_{\mathbf{X}}{=}1$; the remaining rows follow the same rule.

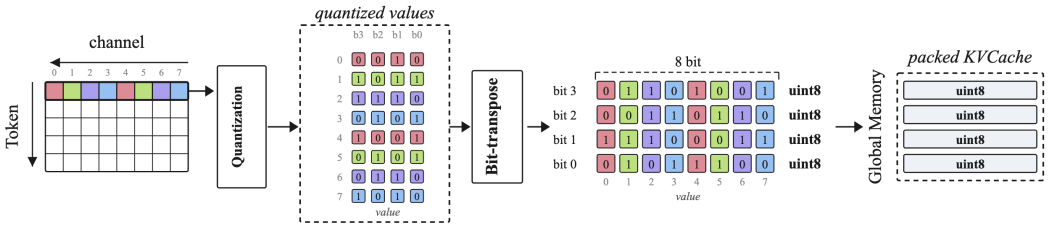

Figure 12: **Write path.** Quantized KV values are bit-transposed into separately allocated byte-packed planes. Releasing the lowest planes later reduces precision without re-quantization.

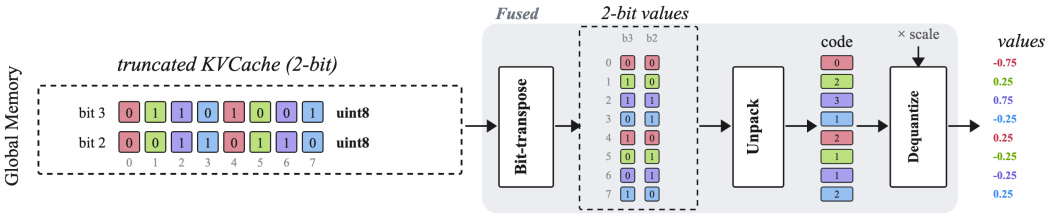

Figure 13: **Reading a truncated chunk.** The same fused kernel reconstructs values from two retained planes, exactly matching direct 2-bit quantization (Eq. (4)).

The plane loop is unrolled at compile time over all n positions and each iteration is gated by the predicate $b \geq k$, so one compiled kernel serves any effective width $e \in \{1, 2, 3, 4\}$, including a batch in which different chunks carry different widths—no recompilation and no divergence between chunks. The kernel reads only the still-compressed planes and block maxima, and writes only the requested cache values. No decoded index tensor is ever materialized, so peak memory during a read is the compressed footprint plus the output tile. Accumulation and the reciprocal are in `fp32`. Widths outside $\{1, 2, 3, 4\}$ fall back to a reference unpack, and when companding is enabled the inverse warp g^{-1} of Eq. (5) is fused into the store. The zero-bit endpoint is handled by the sliding window rather than by this positive-width dequantization rule.

D.3 Truncation as Plane Dropping

Figure 6 shows what a truncation from n=4 to e=2 does to the stored cache. The two lowest planes, `bit 1` and `bit 0`, are released, and the surviving planes `bit 3` and `bit 2` are left byte-for-byte unchanged. No value is re-quantized, no full-precision source is consulted, and no data is moved—the truncation is a metadata update plus a deallocation.

Figure 13 then reads the truncated chunk with the *same* kernel as Appendix D.2, now with k=2 dropped planes instead of none. This is where truncation-exactness becomes visible end to end. The first element was stored as $u_4 = 2 = (0010)_2$ and decoded to -0.69; after truncation its index is $u_4 \gg 2 = 0 = (00)_2$, which decodes to $(2(0 - 2) + 1)/4 = -0.75$—exactly the value that quantizing the original scalar at 2 bits directly would have produced. The second element likewise moves from $u_4 = 11 = (1011)_2$ and 0.44 to $u_2 = 2 = (10)_2$ and 0.25, and so on down the group. The truncated cache is therefore not an approximation of a 2-bit cache; by Eq. (4) it *is* the 2-bit cache, which is what allows Algorithm 1 to lower precision at any point during generation directly from the quantized KV-cache.

D.4 Age-Based Eviction

The evaluated policy uses $\mathcal{L} = \{4, 3, 2, 1, 0\}$ in Algorithm 1 under a fixed ceiling $\mathcal{B} = b_{\text{target}} \cdot N_{\text{planned}}$ (the static-QVG footprint of the *whole planned* sequence). Below $\mathcal{B}$ no truncation is needed—early generation stays high-bit. Over it, the bit-level sliding window first truncates the oldest spans from 4 to 3 bits, then, if the cache is still over budget, from 3 to 2 bits, from 2 to 1 bit, and finally from 1 bit to eviction, stopping the sweep as soon as the cache footprint falls back under the ceiling. Every individual step is the plane drop of Figure 6, so each phase is monotone—a span's width never

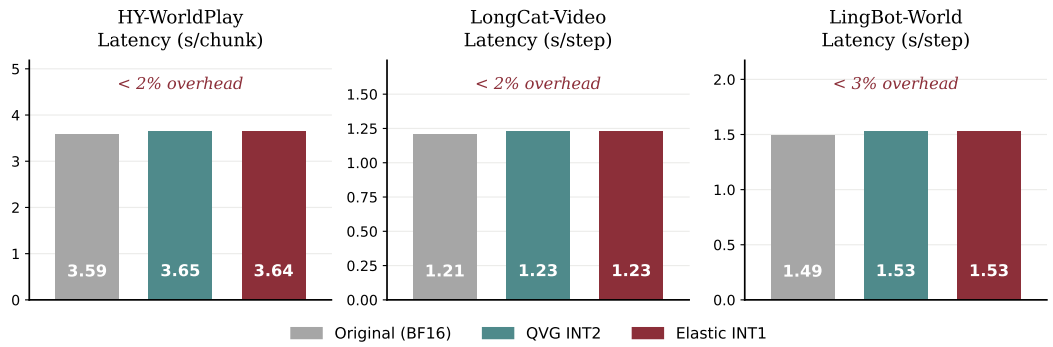

Figure 14: **End-to-end latency on a single NVIDIA B300 GPU** (480p). Across all three backbones, Elastic INT1 stays within 3% of the uncompressed BF16 baseline and matches static QVG INT2, despite using a lower target precision and running the truncation sweep online.

rises—and zero-copy, and the freed planes are reclaimed in one batched pass after the sweep rather than one deallocation per truncation. No sink exemption is applied.

E Latency Profile on B300

Figure 14 reports the result for all three backbones. Latency is quoted per generated chunk for HY-WorldPlay, which emits 12 frames at a time, and per denoising step for LongCat-Video and LingBot-World. Elastic INT1 adds 1.4% (3.59 $\rightarrow$ 3.64 s/chunk) on HY-WorldPlay, 1.7% (1.21 $\rightarrow$ 1.23 s/step) on LongCat-Video, and 2.7% (1.49 $\rightarrow$ 1.53 s/step) on LingBot-World over the uncompressed BF16 baseline. All three measurements remain within the 5% overhead reported on H100 in §4.4.

Two comparisons clarify this overhead. First, Elastic-QVG has nearly the same measured latency as static QVG INT2 on every backbone (3.64 *vs.* 3.65, 1.23 *vs.* 1.23, and 1.53 *vs.* 1.53), even though it additionally runs a truncation sweep at every generation step. As explained in Appendix D, truncation never re-quantizes a chunk, so the sweep contributes only $O(\#\text{chunks})$ metadata updates plus one batched deallocation, and the two methods share the same fused read kernel. Second, the remaining gap to BF16 is attributable to the unpack+dequant work on the read path rather than to elasticity, since it is already present in the static baseline.

F Additional Results on LingBot-World-v2

Table 3: **Additional results on LingBot-World-v2.** Same protocol as Table 1 (480p, 285 frames, 1003 MovieGen prompts). Fidelity is measured against the BF16 KV-cache; the compression ratio is the codec ratio over the quantized span.

Method	Compr. (BF16)	PSNR↑	SSIM↑	LPIPS↓	Bg Consist.	Image Quality	Subj. Consist.	Aesth. Quality
BF16 (reference)	1.00×	–	–	–	89.78	65.03	81.53	55.75
2-bit KV Cache								
QVG	5.12×	26.14	**0.713**	**0.223**	89.81	62.46	81.71	53.96
Elastic-QVG	5.35×	**26.35**	0.695	0.238	89.72	64.83	81.39	55.59
1-bit KV Cache								
QVG	8.17×	21.38	0.609	0.396	87.54	55.33	76.28	47.62
Elastic-QVG	8.17×	**23.64**	**0.638**	**0.308**	89.66	64.10	81.04	55.36

The 720p evaluation is reported separately in Table 2 in the main paper; this appendix provides the additional 480p results on 1003 MovieGen prompts.

G Additional Qualitative Results



Figure 15: **LingBot-World.** Elastic-QVG better preserves BF16 visual quality than QVG under low-bit KV-cache compression.

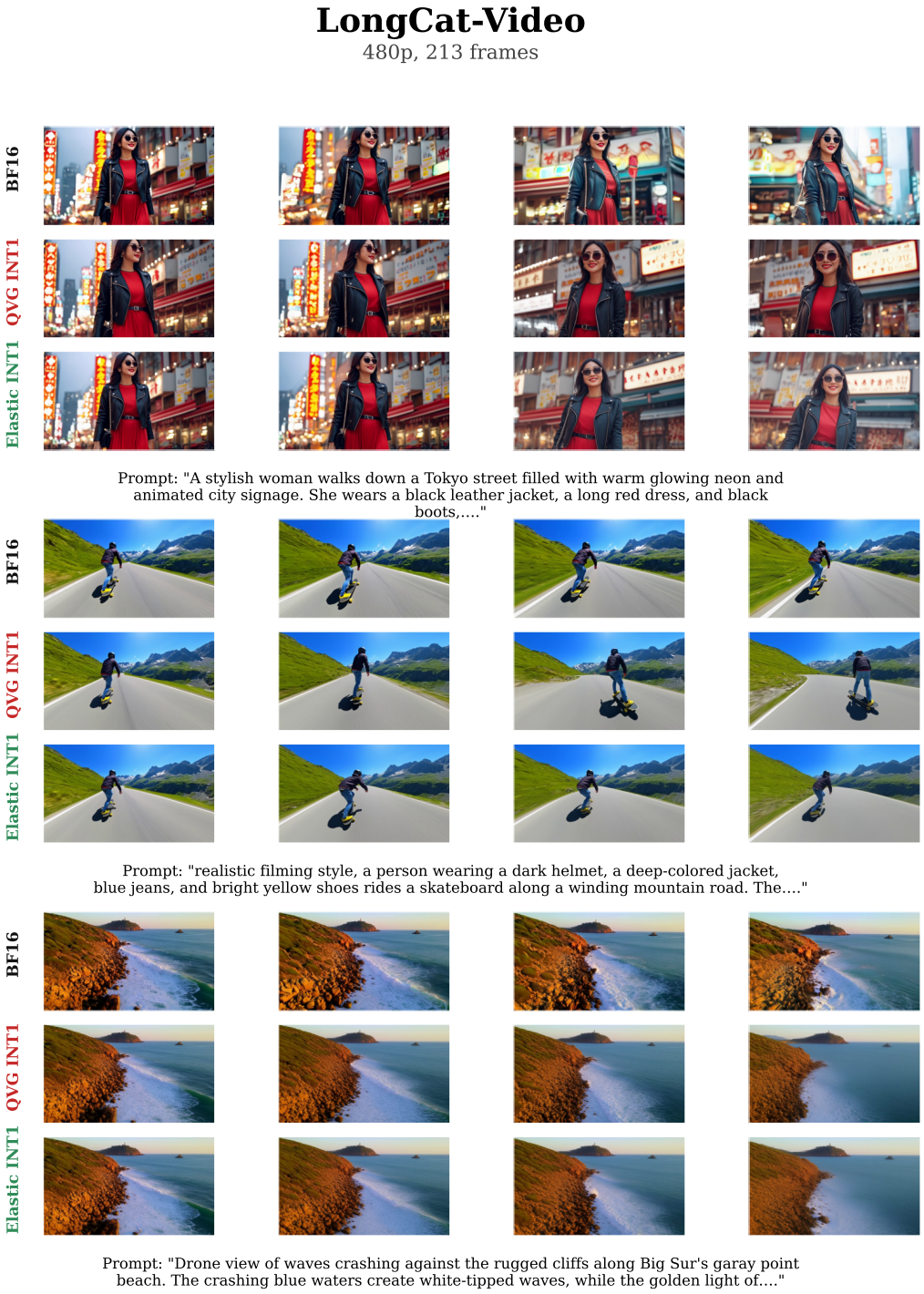

Figure 16: **LongCat-Video.** Elastic-QVG maintains long-context visual consistency with fewer compression artifacts than QVG.



Figure 17: **HY-WorldPlay.** Elastic-QVG produces outputs closer to BF16 while reducing KV-cache precision.