TRANSFORMER-FSQ: DISCRETE LATENTS FOR EXTREME KV-CACHE COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Private, personalised language models must run on local consumer devices, where frontier models cannot fit and capability depends on what the model retains in context. However, context memory—the key-value (KV) cache—grows linearly with retained cache entries, quickly dwarfing the model itself: for Qwen3-1.7B-Base, 200,000 retained entries require 21 GiB, 6.6 times more than the model weights. While high token capacity is urgently needed on edge hardware, the KV-cache compression methods we measured fail beyond modest compression ratios. We introduce *Transformer-FSQ*, addressing extreme KV-cache compression by replacing standard key and value projections with a finite scalar quantisation bottleneck. Rather than storing large vectors, the cache retains a packed vector of discrete integer coordinates per retained token per layer, requiring no per-token quantization metadata. Converting a pretrained transformer requires training these projections on 0.25 tokens per original parameter. We evaluate Transformer-FSQ against five baseline families: token eviction, scalar and vector quantisation, low-rank projection, and latent attention on models from 0.6B to 8B parameters. On long-context retrieval (RULER), the methods fall into two regimes. Up to about 5×, vector quantisation and low-rank projection stay close to the uncompressed model; from 12× to 97.5×, Transformer-FSQ is the most accurate method among our primary evaluated implementations, and leads at extreme compression across scales. Extreme KV-cache compression is therefore a practical way to extend how much context a model on a fixed device can hold.

1 INTRODUCTION

A language model's key-value (KV) cache grows with every token it processes, and on local hardware, the cache competes directly with model weights for a fixed memory budget. Unlike static $O(1)$ model weights, the KV cache grows dynamically with retained cache entries, rapidly outgrowing the model itself. Local deployments matter: prompts stay on device within user control (Nissenbaum, 2010; Stallman, 2015; Zittrain, 2008), and processing spreads beyond centralised intermediaries (Benkler, 2006; Seger et al., 2023); releasing weights alone does not achieve this (Liesenfeld & Dingemanse, 2024; Widder et al., 2024). For Qwen3-8B-Base, caching 144 KiB per token in half precision (36 layers, 8 key-value heads of width 128) (Yang et al., 2025) alongside 15.3 GiB of weights on a 16 GiB GPU leaves room for only 5 thousand uncompressed tokens before runtime activations, reaching 1.52M retained entries only at 256× compression (Figure 1 highlights the difference in magnitudes). Large capacity is essential: in-context learning, multi-step reasoning, and concurrent rollouts all depend on holding many tokens in memory simultaneously (Li et al., 2025b).

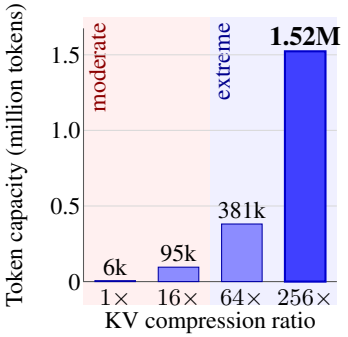

Figure 1: **KV-cache capacity on a 16 GiB GPU.** With only ≈0.7 GiB left for cache after Qwen3-8B model weights, extreme compression is vital in enabling high volumes of tokens.

KV-cache compression trades accuracy for context. Token eviction keeps fewer tokens, scalar and vector quantisation store fewer bits per coordinate, and low-rank projection and latent attention store fewer coordinates (§2). Four of the five published methods we compare report accuracy at compres-

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.

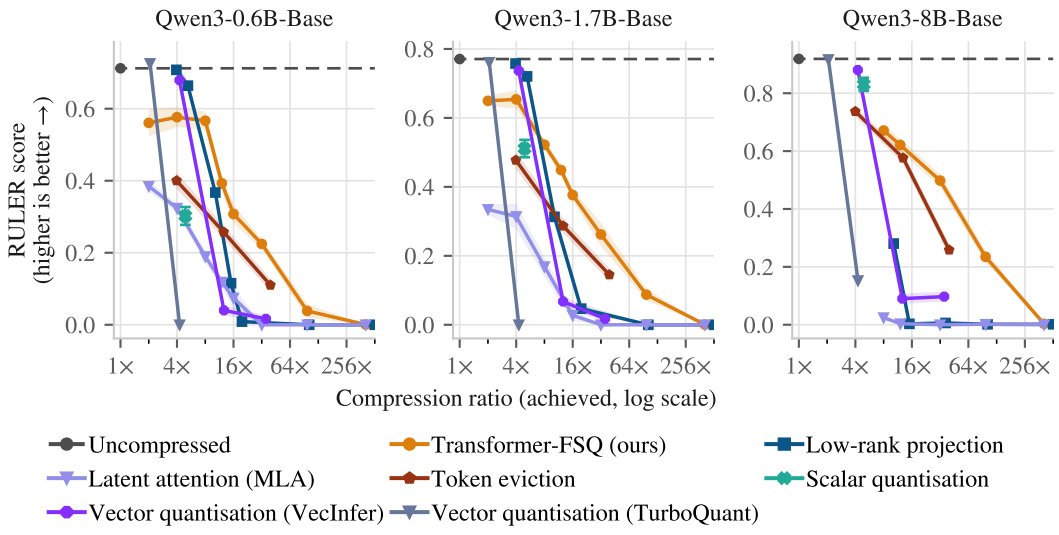

Figure 2: **Retrieval accuracy across compression regimes:** RULER score at 8,192 tokens against achieved KV-cache compression ratio across Qwen3-0.6B, 1.7B, and 8B; dashed horizontal line: uncompressed baseline (1.0×). Points average up to three seeds (one on 8B) with 95% bootstrap intervals over items stratified by task. Beyond about 12×, Transformer-FSQ maintains the highest retrieval accuracy across all three models, while low-rank, vector quantisation, and eviction baselines experience sharp degradation.

sion ratios of at most about 14× (Table 1), whereas holding several million tokens beside a model's weights on a 16 GiB GPU needs ratios of tens (Appendix D.3), the extreme regime this paper targets.

To reach extreme compression we use finite scalar quantisation (FSQ) (Mentzer et al., 2024). FSQ projects a continuous representation into a low-dimensional space and rounds each coordinate to a fixed number of levels. Unlike learned-codebook vector quantisation (van den Oord et al., 2017; Esser et al., 2021), FSQ requires no stored codebook: the discrete grid is defined implicitly by the bounding levels, eliminating codebook search, learned codebook collapse, and gradient noise from categorical relaxations. Transformer-FSQ replaces the key and value projections of attention with an FSQ encoder and a linear decoder, storing bit-packed integer codes in the KV cache rather than floating-point vectors. Conversion inserts these projections into a pretrained transformer in closed form, and finetuning trains only the new parameters on 0.25 tokens per original parameter, leaving the backbone model frozen. The inserted projections are lightweight at extreme compression: at 97.5× on Qwen3-1.7B-Base they add under 10M parameters in total, though at 2× on Qwen3-0.6B-Base they add 352M parameters (≈ 60% of the model) as the code approaches the residual dimension. This inflation is not inherently a part of the method (see Appendix E.4).

Figure 2 evaluates Transformer-FSQ against five baselines from five families, plus TurboQuant on RULER (Hsieh et al., 2024) (13 official tasks, 50 items each) at 8,192 tokens across three Qwen3 models. Below 8×, training-free vector quantisation and low-rank projection preserve accuracy with less overhead, whereas Transformer-FSQ's uniform 16-level allocation is tuned for high compression rather than mild ratios. From 12× to 97.5×, Transformer-FSQ retains substantially higher retrieval accuracy than the primary evaluated baseline configurations, outscoring token eviction by 0.12 and 0.15 at 12× on Qwen3-0.6B and 1.7B, and retaining the highest share of uncompressed accuracy through 97.5×. Furthermore, because Transformer-FSQ compresses channel width while token eviction reduces sequence length, the two compose: on a 16 GiB GPU, Qwen3-0.6B-Base with both yields an aggregate projected capacity of 3.8M tokens across concurrent sequences at RULER $\geq$ 0.30, 2.7× the best baseline.

Our contributions are: (1) Transformer-FSQ, a discrete latent compression method that stores packed integer codes in the KV cache (§4.1); (2) a closed-form conversion and finetuning operator requiring only 0.25 tokens per original parameter on ClimbMix with the backbone frozen (§4.2); and (3) an empirical evaluation showing that beyond 12×, Transformer-FSQ dominates the Pareto frontier among evaluated primary baseline implementations on RULER at 8,192 tokens, stacking with eviction to expand aggregate projected capacity by 2.7× on a 16 GiB GPU (§5).

2 Related Work

Attention remains the dominant sequence mixer in large language models, despite alternatives (sparse, linear, recurrent, state-space) that alter sequence processing to reduce long-context costs. We study a complementary problem: *retaining Transformer attention while reducing the state stored during inference*. In autoregressive decoding, persistent key–value caches grow linearly with context length; FlashAttention (Dao et al., 2022) optimizes working memory during execution, but leaves this persistent cache uncompressed.

KV-cache compression mitigates this bottleneck while preserving existing model weights and architectures. Prior methods compress along four axes: token count, coordinate precision, coordinate dimension, or discrete codebooks. We review these below (baseline details: Appendices C and E).

Token eviction: fewer stored tokens. Token eviction discards less important key–value pairs along the sequence dimension, keeping full precision for surviving tokens. StreamingLLM (Xiao et al., 2024) retains attention sinks and a local sliding window; PyramidKV (Cai et al., 2024), SnapKV (Li et al., 2024), and H_2O (Zhang et al., 2023) allocate cache budgets by attention patterns. While eviction lowers both memory and attention arithmetic, discarded token context is permanently lost.

Scalar quantisation: fewer bits per coordinate. Scalar quantisation preserves key–value dimensionality while reducing numerical precision. KIVI (Liu et al., 2024) quantises keys per channel and values per token to 2 bits alongside a full-precision window; KVQuant (Hooper et al., 2024) and GEAR (Kang et al., 2024) employ non-uniform grids or error-correcting residuals. TurboQuant (Zandieh et al., 2026) rotates activations before scalar quantisation; it appears in Figure 2 evaluated training-free at mild compression, without recovery finetuning or extreme-budget adaptation (Appendix E).

Latent projection: fewer coordinates per token. Latent projection reduces key–value dimensionality into narrower continuous vectors. Multi-Head Latent Attention (MLA) (DeepSeek-AI, 2024) replaces per-head storage with a shared continuous latent and decoupled rotary key; TransMLA (Meng et al., 2025) adapts models to MLA via finetuning, while Palu (Chang et al., 2025) factorises key–value projections via SVD. These latents remain floating-point vectors unless subsequently quantised.

Vector and discrete quantisation: discrete representations per token. Vector quantisation replaces continuous representations with discrete codes. Classical VQ maps inputs to nearest codebook entries (van den Oord et al., 2017); product quantisation codes subspaces independently (Jégou et al., 2011), and residual quantisation codes successive error stages (Lee et al., 2022; Zeghidour et al., 2022). For KV caches, coupled quantisation (Zhang et al., 2024) and CommVQ (Li et al., 2025c) learn codebooks; VecInfer (Yao et al., 2026) suppresses outliers before VQ; and PQCache (Zhang et al., 2025), Kumar (2024), VQKV (Wang et al., 2026), and GSRQ (Kim et al., 2026) use product or residual variants.

Transformer-FSQ bridges dimensionality reduction and discrete representation: it maps the residual stream to a compact latent, rounding coordinates onto a finite grid. Unlike conventional VQ, this Cartesian grid is implicit and unlearned, eliminating stored codebooks and nearest-neighbour search.

3 Discrete Latents for KV-Cache Compression

Our goal is to push KV-cache compression into the extreme regime, where reducing only precision or representation width is insufficient. We seek a representation reducing both coordinates and bits per coordinate without auxiliary tables: a wide, low-bit latent narrower than the full KV cache but wider than an equal-byte floating-point vector. We examine VQ and FSQ as two routes to this discrete bottleneck.

Vector quantisation. Discrete representation learning maps a continuous latent vector to a finite set of states. In vector quantisation (VQ), states are stored explicitly as codebook $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$, with $\mathbf{c}_k \in \mathbb{R}^d$, mapping input $\mathbf{z} \in \mathbb{R}^d$ to its nearest codeword:

$$Q_{\mathrm{VQ}}(\mathbf{z}) = \mathbf{c}_{k^\star}, \quad k^\star = \arg\min_k \|\mathbf{z} - \mathbf{c}_k\|_2^2. \tag{1}$$

Nearest-neighbour assignment partitions latent space into Voronoi cells (Gray & Neuhoff, 1998). However, learned codebooks require commitment losses, updates, and nearest-neighbour search (shown on Figure 3a), risking centroid collapse where entries become inactive (van den Oord et al., 2017; Mentzer et al., 2024).

Finite scalar quantisation. FSQ replaces the learned vector codebook with fixed scalar levels in each latent coordinate (Mentzer et al., 2024). Let $\mathcal{L}_i$ denote the finite set of allowed scalar values for coordinate i, and let $L_i = |\mathcal{L}_i|$ be the number of levels in that coordinate; for instance, in Figure 3b every coordinate may take one of the $L_i = 3$ values $\mathcal{L}_i = \{-1, 0, 1\}$. FSQ first bounds each encoder output and quantises coordinates independently (see Eq. 8, Appendix F.1),

$$\hat{z}_i = Q_i(z_i) \in \mathcal{L}_i. \tag{2}$$

The complete set of representable vectors is therefore the Cartesian product

$$\mathcal{C}_{\text{FSQ}} = \mathcal{L}_1 \times \cdots \times \mathcal{L}_d, \quad |\mathcal{C}_{\text{FSQ}}| = \prod_{i=1}^{d} L_i. \tag{3}$$

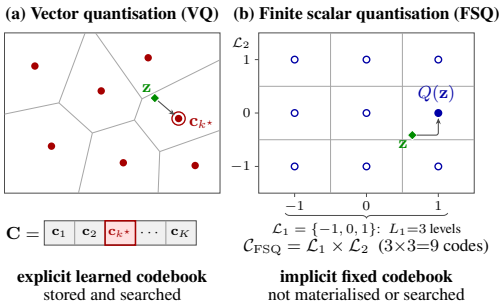

Figure 3: **Discrete latent geometry: (a)** VQ searches an explicit codebook, inducing irregular Voronoi cells (Gray & Neuhoff, 1998). **(b)** FSQ quantises coordinates independently onto a fixed Cartesian grid $\mathcal{L}_1 \times \mathcal{L}_2$, an implicit codebook that is never materialised or searched.

When all coordinates use $L_i = L$ levels, the implicit code space contains L^d vectors ($3^2 = 9$ grid points in Figure 3b); in practice the L_i may differ across coordinates to meet a target code-space size (Mentzer et al., 2024). This space is combinatorially large but is never materialised: encoding requires only coordinate-wise rounding, with no centroid table or nearest-neighbour search. Gradients pass through rounding via a straight-through estimator. Because codewords are fixed to a regular Cartesian grid rather than learned vectors, FSQ avoids commitment losses, codebook updates, and learned codebook collapse (Mentzer et al., 2024), while code utilisation across individual coordinates is empirically evaluated in Appendix E.7.

FSQ latents for KV-cache compression. These properties make FSQ an ideal bottleneck for transformer KV-cache compression: (i) *Simplicity*: stateless coordinate-wise rounding requires no learned codebook or commitment objectives (van den Oord et al., 2017); (ii) *Live capacity*: an implicit code space (L^d) removes learned-centroid collapse failure modes; (iii) *Zero metadata*: fixed integer coordinates cost exactly $\lceil \log_2 L_i \rceil$ bits with no per-token or per-channel scale overhead; and (iv) *Decode footprint*: lookup-free elementwise dequantization directly reduces stored cache bytes during autoregressive decoding. Section 4 integrates this bottleneck into the attention pathway.

4 TRANSFORMER-FSQ

Now we retrofit a pretrained transformer with an extremely compact per-token state that can replace the conventional KV cache, without pretraining the model from scratch. This setting differs from both conventional discrete tokenization and existing KV-cache quantization. FSQ is typically used to map a continuous signal to a small number of quantized coordinates whose Cartesian product acts as a discrete token, while KV-cache quantization usually operates after keys and values have already been formed. Transformer-FSQ instead makes two design choices tailored to the internal state of a pretrained language model.

Wide product codes. A cached language-model state must preserve enough information to support many future, query-dependent KV readouts. We therefore use FSQ in a different regime from conventional image and audio tokenizers Li et al. (2026): not as a narrow categorical token, but as a *wide product representation* consisting of many low-precision coordinates. Under a fixed byte budget, lowering the precision of each coordinate buys proportionally more latent dimensions, while FSQ can scale to this width without materializing or searching the exponentially large Cartesian code space. Thus L^G should be understood as the number of representable product codes, not as a categorical vocabulary.

Compress once, before K/V expansion. Standard attention first maps the residual state x_t into separate keys and values, possibly across several KV heads, and caches these derived representations. Post-hoc KV quantization therefore compresses multiple projections of the same underlying state. Transformer-FSQ instead places the bottleneck on the shared residual representation,

$$x_t \longrightarrow z_t \longrightarrow \{\hat{k}_t^{(h)}, \hat{v}_t^{(h)}\}_{h=1}^{n_{kv}},$$

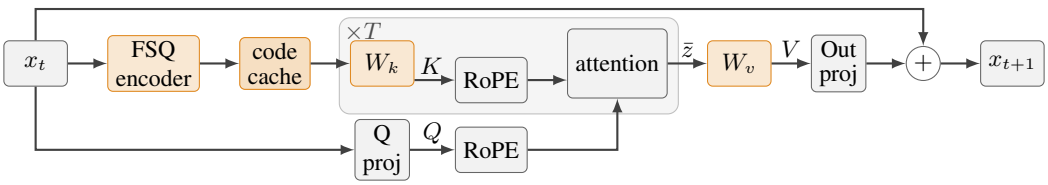

Figure 4: **Transformer-FSQ decoding.** Top: standard attention caches separately projected keys and values, whereas Transformer-FSQ caches their shared discrete source. Bottom: one decoding step. A token is encoded once into a bit-packed code z_t, and the same code serves all KV heads. In the RoPE-based Qwen models evaluated here, keys are reconstructed and rotated at each cached position ($\times T$); values exploit linearity and are decoded only after attention pools the codes, $\bar{z} = \sum_t a_t z_t$. Orange denotes components introduced by Transformer-FSQ; grey denotes frozen backbone operations. The bottleneck itself is independent of the positional encoding. Appendix F contrasts this with standard transformer decoding.

so each token is compressed once and the same cached code serves all keys, values, and KV heads. This is the discrete analogue of caching a shared continuous latent as in MLA. Figure 4 illustrates the resulting architecture.

4.1 The FSQ bottleneck

Let $x_t \in \mathbb{R}^C$ denote the residual stream at token t. A learned linear encoder $E \in \mathbb{R}^{G\times C}$ maps it to a G-dimensional latent, which is bounded and rounded coordinate-wise to L fixed levels:

$$z_t = Q_{\mathrm{FSQ}}(Ex_t) \in \mathcal{L}_1 \times \cdots \times \mathcal{L}_G. \tag{4}$$

Gradients pass through the rounding operation with a straight-through estimator (Bengio et al., 2013). We use the standard FSQ bounding parameterization of Mentzer et al. (2024); its exact form and integer packing convention are given in Appendix F.1.

With $b = \lceil \log_2 L \rceil$ bits per coordinate, the cache occupies $\lceil \frac{Gb}{8} \rceil$ bytes per token per layer and stores nothing beside the packed code. All main-text experiments use $L = 16$, i.e. four bits per coordinate. At equal storage, this permits four times as many coordinates as an FP16 latent. Because the codebook is implicit, increasing G introduces no centroid table, nearest-neighbour search, commitment loss, codebook update, or dead-code replacement.

A single decoder $W_{kv} \in \mathbb{R}^{2n_{kv}d_h \times G}$, split into W_k and W_v, reconstructs all KV heads:

$$\hat{k}_t = \mathrm{RoPE}_t(N_k(W_k z_t)), \qquad \hat{v}_t = W_v z_t, \tag{5}$$

where N_k denotes the model's native key normalization when present. The value projection need not be applied separately to every cached token. For query head h with attention weights $a_t^{(h)}$,

$$\sum_t a_t^{(h)} W_v z_t = W_v \left(\sum_t a_t^{(h)} z_t \right). \tag{6}$$

Values are therefore decoded only after attention has pooled the cached codes. Keys, in contrast, must be reconstructed before their dot products with the query because positional transformations act on each cached position separately. Transformer-FSQ thus trades cache traffic for additional key-side arithmetic; Section 5.2 measures this runtime cost.

4.2 Retrofitting a pretrained transformer

Transformer-FSQ is applied as a post-training operator rather than requiring a new model to be pretrained from scratch. For each attention layer, the original write-time key and value projections are replaced by the encoder E and shared decoder W_{kv}. These inserted tensors are initialized in closed form from a small calibration set using the pretrained model's own activations, and only the inserted parameters are subsequently finetuned under the language-model objective while the backbone remains frozen. Conversion alone already produces a functional model, analogous to the post-training conversion procedures of Palu (Chang et al., 2025) and TransMLA (Meng et al., 2025).

The closed-form initialization uses principal directions of the calibration residual states for the encoder and a ridge-regularized least-squares fit for the decoder. The exact construction, including

Table 1: **Compression baselines and 16 GiB GPU capacity limits:** Comparison across representative methods from each compression family. **Deepest**: deepest ratio where original authors report downstream accuracy under our uniform byte pricing (§4.4); for PyramidKV, evaluated on needle retrieval in the original work; for Transformer-FSQ, the deepest ratio evaluated ($410\times$), where RULER retrieval collapses to near zero (0.003) on Qwen3-0.6B-Base. **Max tokens**: maximum retained cache token entries fitting alongside weights in 16 GiB VRAM on Qwen3-0.6B-Base at RULER ≥ 0.30, priced analytically by the byte solver. Eliminating auxiliary metadata enables Transformer-FSQ to hold 2.21M retained tokens standalone at $16\times$ and compose with PyramidKV eviction to reach an aggregate projected capacity of 3.8M tokens across concurrent sequences at $32\times$.

Method	**Family**	**Cache contents per token**	**Trained**	**Deepest**	**Max tokens (16 GiB, ≥ 0.30)**
PyramidKV	token eviction	full-precision KV of kept tokens	no	$174\times$	1.5M
KIVI	scalar quantisation	2-bit KV, full-precision window	no	$5.2\times$	720k
VecInfer	vector quantisation	codebook indices, window, sinks	no	$11.0\times$	756k
Palu	low-rank projection	quantised low-rank SVD latent	yes	$10.2\times$	1.42M
TransMLA	latent attention	latent, fp16 rotary key	yes	$14.2\times$	570k
Transformer-FSQ (ours)	finite scalar quantisation	bit-packed integer code	yes	$410\times$	2.21M

the $G > C$ case and all regularization details, is deferred to Appendix C. The important point is that the bottleneck is initialized to preserve the pretrained computation before any gradient update, making recovery finetuning data-efficient.

4.3 Interaction with positional encoding

The converted model retains its native positional encoding. In Qwen-style RoPE attention (Su et al., 2024), the cached code itself carries no explicit position; position is introduced only after reconstructing each cached key, $\hat{k}_t = \mathrm{RoPE}_t(N_k(W_k z_t))$. Because the position-dependent rotation lies between W_k and the query–key dot product, the key decoder cannot in general be folded into the query. Consequently, keys must be reconstructed at every decoding step, which is the main arithmetic overhead of Transformer-FSQ.

This overhead is specific to the interaction with RoPE rather than to the discrete bottleneck itself. Without position-dependent key rotation, the linear key decoder can be absorbed into the query when key normalisation is absent or handled separately. We evaluate a pre-RoPE variant, FSQ-pre, in Appendices E and F; the exact interaction with query–key normalization is also detailed there.

4.4 Sizing every cache from one byte budget

We compare all methods using the same realized cache-byte budget. Every quantity stored per token is charged, including quantization metadata, full-precision windows, positional state, and auxiliary cached representations. Reported compression ratios therefore divide the uncompressed FP16 KV-cache size,

$$c_{\text{full}} \;=\; 2\, n_{kv}\, d_h \cdot \mathrm{sizeof}(\text{float16}) \qquad \text{bytes per token per layer,} \tag{7}$$

by the actual bytes retained by each configuration; c_{full} is 4096 B for the three Qwen models and 3072 B for the 186M model we pretrain. The complete accounting rules are given in Appendix C.

5 Experiments

5.1 Setup

Models We convert three pretrained models, Qwen3-0.6B-Base, Qwen3-1.7B-Base and Qwen3-8B-Base (Yang et al., 2025) (Appendix C). Against Transformer-FSQ we compare representative methods from five distinct families (Table 1): token eviction (■ PyramidKV (Cai et al., 2024)), scalar quantisation (■ KIVI (Liu et al., 2024)), vector quantisation (■ VecInfer (Yao et al., 2026)),

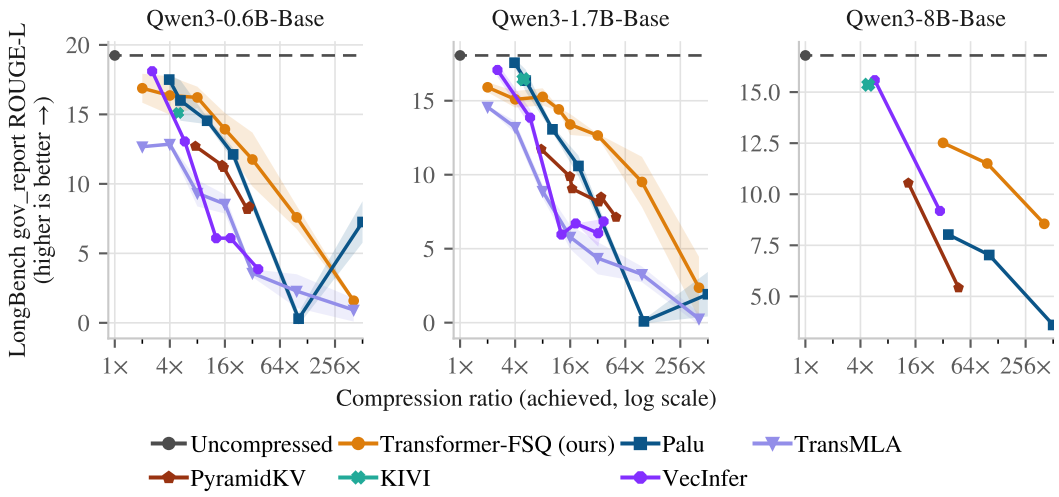

Figure 5: **Natural document summarization across compression ratios:** LongBench `gov_report` ROUGE-L against achieved KV-cache compression ratio across Qwen3-0.6B, 1.7B, and 8B; dashed horizontal line: uncompressed baseline score. Across all three model scales, Transformer-FSQ maintains the highest summarisation fidelity into extreme compression ($\geq 32\times$), whereas low-rank and eviction baselines either degrade sharply or cannot operate beyond $16\times$–$32\times$.

low-rank projection (■ Palu (Chang et al., 2025)), and latent attention (■ TransMLA (Meng et al., 2025)). The former three require no training, while Palu and TransMLA are finetuned with the same data, optimiser and token budget as ■ Transformer-FSQ. Each follows its authors' specification except where noted (Appendix C), is sized by §4.4, and runs as far as that sizing allows. A converted model trains only inserted tensors, with backbone frozen, on ClimbMix (Diao et al., 2025) with AdamW (Loshchilov & Hutter, 2019), at 0.25 tokens per parameter (and 0.025, 0.075 on Qwen3-0.6B-Base to test finetuning budget). Conversion and finetuning took 2304 GPU-hours on NVIDIA GH200 GPUs (Appendix C.5). Decode throughput was benchmarked on 4× NVIDIA L40S GPUs with the fused CUDA decoding kernel.

Benchmarks RULER (Hsieh et al., 2024) is our primary synthetic benchmark because its items provide exact, objective ground truth across controlled sequence lengths without confounding from pretraining knowledge. We evaluate RULER over common alternatives: single-needle retrieval (Kamradt, 2023) is too simple to detect structural degradation across multi-token associations, while sliding-window perplexity is dominated by local n-grams and remains insensitive to catastrophic long-range context loss. Furthermore, synthetic reasoning suites like BABILong (Kuratov et al., 2024) fail on uncompressed base models, confounding cache fidelity with reasoning limits. LongBench (Bai et al., 2024) complements RULER with real-world multi-page tasks (summarisation, question answering), validating that synthetic gains transfer to natural language. Together, these suites diagnose synthetic failure modes and verify practical document comprehension.

Metrics Long-context retrieval is evaluated at 8,192 tokens across RULER's 13 standard tasks (50 items each). LongBench complements this by evaluating natural documents up to 32,768 tokens (truncated to 32,736) across three tasks. All evaluations are scored with official task metrics in float32 precision for metric isolation. Reported RULER points average up to three checkpoint seeds covering all items (one on Qwen3-8B-Base), with 95% bootstrap confidence intervals stratified over seeds and items; pairwise leads resample identical seeds and items. This protocol isolates compression degradation from prompt sampling noise under a unified evaluation harness.

5.2 Results

Our experiments address four central questions, explored in detail below:

- **RQ1:** How does Transformer-FSQ fare against baselines across compression regimes?
- **RQ2:** Do synthetic retrieval gains transfer to natural multi-page document summarization?
- **RQ3:** Does low-bit latent geometry matter under extreme compression?
- **RQ4:** What is the runtime cost in decode throughput?

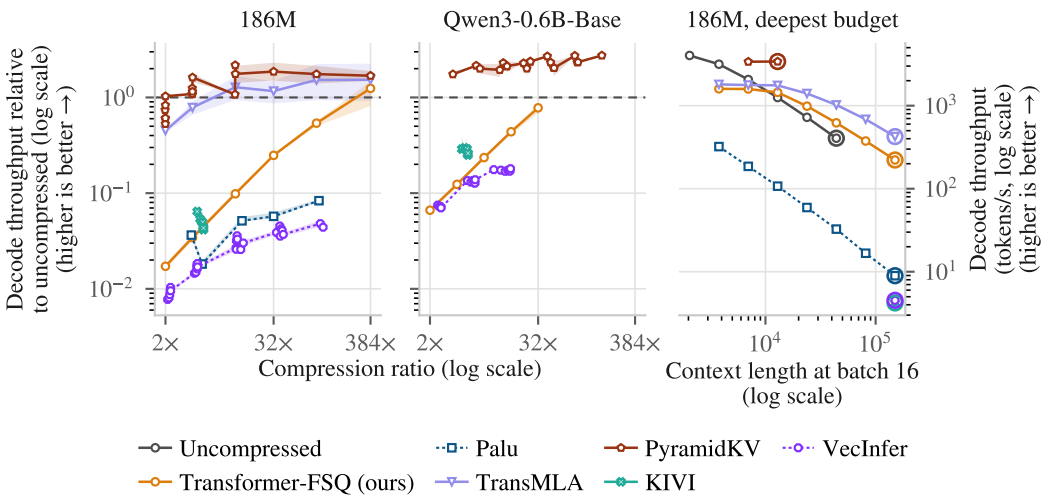

Figure 6: **Decode throughput across compression ratios and context lengths:** Relative decode throughput vs uncompressed model for 186M (left) and Qwen3-0.6B-Base (middle), showing median and IQR across context lengths. Right: absolute throughput vs context length at batch 16 for 186M at deepest compression budget (384×); double circles mark out-of-memory limits. Palu and VecInfer (dotted) are unfused ports. Transformer-FSQ trades per-step RoPE key rotation overhead at moderate ratios for higher throughput at extreme compression (384×), extending context reach without out-of-memory failure (profiling settings: Table 9).

RQ1: Accuracy across compression regimes. Across compression regimes, training-free and low-rank methods excel at mild compression ($\leq$ 5×), while Transformer-FSQ dominates extreme compression ($\geq$ 12×), answering RQ1. At $\leq$ 5×, VecInfer and Palu retain $>$ 93% of uncompressed RULER scores across all models (95%, 96%, 96%; Figure 2). Beyond 12×, continuous latents and eviction collapse: Palu drops to 16% at 15.1×, while Transformer-FSQ outclasses token eviction by 0.12 at 12× and 0.11 at 32× on Qwen3-0.6B-Base, reaching 0.50 on Qwen3-8B-Base (a lead of 0.24; Table 15). Discrete FSQ latents thus establish the Pareto frontier under extreme compression (12×–97.5×).

RQ2: Natural document summarization. Synthetic retrieval resilience transfers directly to natural multi-page document summarisation, answering RQ2 affirmatively. On LongBench `gov_report` (Figure 5), baselines track uncompressed scores (16.8 ROUGE-L on Qwen3-8B-Base) at mild compression but collapse at higher ratios. In contrast, Transformer-FSQ maintains highest fidelity across extreme ratios (32×–409.6×), scoring 12.5 at 32× and 8.6 at 409.6×, where Palu drops to 3.6. Across all three scales (Tables 16–18), finetuned models generalise beyond their 1,024-token training sequences, preserving semantic coherence under extreme compression.

RQ3: Impact of latent geometry. Learning to fit a low-bit grid provides the strongest retrieval–memory operating point under extreme compression (Table 2). We compare four bottlenecks at a fixed budget of 80 B per token per layer (51× compression), including quantisation metadata. All keep the pretrained Qwen backbone frozen and finetune only the inserted bottleneck parameters. Transformer-FSQ applies its grid throughout quantisation-aware finetuning (QAFT); post-hoc FSQ instead trains a continuous latent and applies the grid only at inference.

At the same width of 160 and 4-bit precision, QAFT-FSQ achieves 0.182 pm0.018 RULER, compared with 0.144 ± 0.013 for post-hoc FSQ. This comparison directly supports learning representations that accommodate the inference-time grid. Conventional min–max 4-bit QAFT achieves 0.153 ± 0.011 with width 128: its scale metadata leaves fewer latent coordinates within the same budget. Pure continuous rank compression, stored at 16-bit precision, permits only width 40 and achieves 0.001 ± 0.001.

Together, these results favour a wider low-bit latent trained with its grid over either inference-only quantisation or a narrow continuous bottleneck. The min–max comparison does not isolate geometry from width, but supports FSQ's metadata-free grid as an effective budget-aware design. Appendix E.4 further examines the trade-off between latent width and per-coordinate precision at a fixed cache budget.

RQ4: Decode throughput across compression regimes. Transformer-FSQ trades generation speed at moderate compression for higher throughput and substantial memory reduction at extreme compression, answering RQ4. Benchmarking on NVIDIA L40S GPUs (Figure 6; Table 9) shows that at moderate ratios ($2\times$–$96\times$), Transformer-FSQ decodes at $0.54\times$ uncompressed speed due to per-step RoPE key reconstruction. At extreme compression ($384\times$), memory bandwidth savings dominate arithmetic, yielding a median speedup of $1.24\times$ (up to $1.83\times$ on long contexts; $3.58\times$ in pre-RoPE topologies; Appendix E). Per-step key rotation is thus the primary decode bottleneck, overcome when bandwidth limits dominate at extreme compression.

5.3 Ablations and Stacking with Eviction

Appendix E evaluates architectural variants and training dynamics: nonlinear encoders and per-head codes degrade validation loss, MLA-FSQ imposes compression ceilings, recent-token windows do not shift Pareto frontiers, and lossless entropy coding adds severe decode latency (§E.7). Because Transformer-FSQ shrinks per-token cache bytes while PyramidKV retains fewer tokens, the two compose at inference (Appendix D.3). At a RULER threshold of 0.30, stacking them on Qwen3-0.6B-Base yields an aggregate projected cache capacity of 3.8M tokens across concurrent sequences on a 16 GiB GPU, $2.7\times$ the furthest baseline (95% interval 2.5–2.7; capped by peak prefill memory). These projections extrapolate 8,192-token scores, extending memory capacity without establishing million-token retrieval accuracy.

Table 2: **Low-bit latent geometry under extreme compression.** RULER at 8,192 tokens on Qwen3-0.6B-Base after 0.25 finetuning tokens per parameter; mean $\pm$ standard deviation over three seeds. Two bottlenecks use 4-bit quantisation-aware finetuning (QAFT) with the pretrained backbone frozen, the third quantises post-hoc, and the last is pure continuous. Bytes are per token per layer and include quantisation metadata.

Latent geometry	Width	Bytes	RULER (8k)
80 B/token/layer - 51x compression			
QAFT min–max, 4-bit	128	80	0.153 ± 0.011
QAFT-FSQ, 16 levels (ours)	**160**	**80**	$\mathbf{0.182 \pm 0.018}$
FSQ, post-hoc 16 levels	160	80	0.144 ± 0.013
Rank bottleneck, 16-bit	40	80	0.001 ± 0.001

6 Conclusion and Future Work

Transformer-FSQ compresses what a transformer stores, not just the precision of its existing keys and values. A learned finite scalar quantisation bottleneck replaces separate KV vectors with a shared, packed integer code, requiring no per-token quantisation metadata. Closed-form conversion and finetuning adapt pretrained models while leaving the backbone frozen. Across Qwen3 models from 0.6B to 8B parameters, Transformer-FSQ achieves the highest measured RULER accuracy from $12\times$ to $97.5\times$ compression and preserves summarisation quality at still higher ratios. The key finding is simple: under tight memory budgets, many learned low-bit coordinates can retain more useful information than a few high-precision ones.

These representational gains do not yet remove every deployment barrier. Existing methods remain stronger at mild compression, while repeated RoPE key reconstruction slows decoding at intermediate ratios. Stacking with token eviction further expands projected cache capacity, but our retrieval results at 8,192 tokens do not establish accuracy at million-token contexts. Practical progress must therefore improve accuracy, memory use and latency together.

Three directions offer the clearest next steps. First, *deployment-ready decoding*: training for robustness to low-precision arithmetic and co-designing the bottleneck with positional encodings and fused kernels could address numerical sensitivity and repeated key reconstruction. Second, *adaptive compression*: allocating code width and precision by layer, alongside selective token retention, could preserve important information more effectively than uniform budgets. Third, *capability-aware training and evaluation*: distillation and longer-context experiments could test whether the gains extend to reasoning, code generation and many-shot learning, where retaining isolated facts is not enough.

The broader takeaway is that attention does not require its persistent memory to retain the form of full-precision KV vectors. Learning a compact discrete state opens a deeper compression regime—and a path towards keeping more useful context on the hardware already available.

7 STATEMENTS

AI USE STATEMENT

In this work, we used generative AI tools, large-language-model coding and writing assistants, for two tasks that require disclosure: designing the experiment that stacks Transformer-FSQ with PyramidKV token eviction (Appendix D.3), and implementing the fused decode kernels, part of the method itself. We have not used generative AI tools to generate synthetic data sets, develop theoretical models, formulate or prove mathematical claims, propose hypotheses, design the other experiments, implement the other parts of the methods, translate, clean or reformat data sets, analyse qualitative data or interpret results; the synthetic RULER items come from RULER's own generator. Additionally, we used generative AI tools for tasks with recommended disclosure: creating and editing the evaluation harness, benchmarking and plotting code (distinct from the compression methods' own implementation, disclosed above), creating the scientific figures and the scripts that draw them, editing the paper for readability, and formatting references.

We have reviewed all AI-assisted work. Code produced with AI assistance is covered by an automated test suite, the reported results tables and figures are generated by a script from the result files rather than typed by hand, and every reference was checked against its publisher's or the preprint server's record for title, authors, venue and year. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The reported results tables and figures in this paper are generated from result files rather than typed by hand, and that generation is itself covered by an automated test suite, as the AI use statement above records. Appendix C.1 lists the three released models this paper evaluates, how each is loaded and at what precision; Appendix C.3 gives the closed-form conversion that inserts Transformer-FSQ's encoder and decoder, including the calibration sample and every constant in Eq. 8 (Appendix F.1); Appendix C.2 gives the optimiser, learning rate, schedule, data, sequence length, batch size and precision behind every finetuned configuration. Appendix C.6 gives each baseline's deviation from its published specification, at the bit width, window, sink-token count, rank allocation and codebook grouping we run it at. Every cache size in every table and figure comes from the one byte-budget procedure of §4.4 and Table 1, so methods are compared under a uniform byte-pricing model accounting for all cache tensors and metadata.

Appendix C.4 gives the RULER and LongBench protocols: task lists, item counts, the fixed item seed, per-task generation lengths, scoring rules and evaluation precision; Appendix G tabulates every per-task score behind the figures, and Appendix D gives the Qwen3-8B-Base campaign's narrower coverage. Appendix C.5 gives the GPU-hours by model and the software versions every run used.

An anonymous code repository and evaluation artifacts accompany the submission available at https://github.com/trofsq/paper-code, and full open-source code and data will be released publicly upon publication.

REFERENCES

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Yochai Benkler. *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press, 2006.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E. Burke, Tristan Hume, Shan

Carter, Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language models with dictionary learning. https://transformer-circuits.pub/2023/monosemantic-features/index.html, 2023.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 14705–14722, 2024.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. Palu: KV-cache compression with low-rank projection. In *International Conference on Learning Representations (ICLR)*, 2025.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Tim Dettmers and Luke Zettlemoyer. The case for 4-bit precision: k-bit inference scaling laws. In *International Conference on Machine Learning (ICML)*, 2023.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan Su, Markus Kliegl, Zijia Chen, Peter Belcak, Yoshi Suhara, Hongxu Yin, Mostofa Patwary, Yingyan (Celine) Lin, Jan Kautz, and Pavlo Molchanov. Nemotron-CLIMB: Clustering-based iterative data mixture bootstrapping for language model pre-training. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Samuel Fernández-Menduiña, Amir Ziashahabi, Eduardo Pavez, Antonio Ortega, and Salman Avestimehr. Spend bits where queries look: KV cache vector quantization with attention-preserving transforms. *arXiv preprint arXiv:2608.04074*, 2026.

Robert M. Gray and David L. Neuhoff. Quantization. *IEEE transactions on information theory*, 44 (6):2325–2383, 1998.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations (ICLR)*, 2022.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *Conference on Language Modeling (COLM)*, 2024.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations (ICLR)*, 2017.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.

Tao Ji, Bin Guo, Yuanbin Wu, Qipeng Guo, Lixing Shen, Zhan Chen, Xipeng Qiu, Qi Zhang, and Tao Gui. Towards economical inference: Enabling DeepSeek's multi-head latent attention in any transformer-based LLMs. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025.

Gregory Kamradt. Needle in a haystack - pressure testing LLMs. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, 2023.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. GEAR: An efficient error reduction framework for KV cache compression in LLM inference. In *NeurIPS Efficient Natural Language and Speech Processing Workshop (ENLSP)*, 2024.

Andrej Karpathy. nanochat: The best ChatGPT that $100 can buy. https://github.com/karpathy/nanochat, 2025.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning (ICML)*, 2020.

Soosung Kim, Minjae Park, Eui-Young Chung, and Jaeyong Chung. GSRQ: Gain-shape residual quantization for sub-1-bit KV cache. In *International Conference on Machine Learning (ICML)*, 2026.

Ankur Kumar. Residual vector quantization for KV cache compression in large language model. In *NeurIPS Efficient Natural Language and Speech Processing Workshop (ENLSP)*, 2024.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. BABILong: Testing the limits of LLMs with long context reasoning-in-a-haystack. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. doi: 10.52202/079017-3381.

Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Guihong Li, Mehdi Rezagholizadeh, Mingyu Yang, Vikram Appia, and Emad Barsoum. X-EcoMLA: Upcycling pre-trained attention into MLA for efficient and extreme KV compression. In *Conference on Language Modeling (COLM)*, 2025a.

Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. A survey on large language model acceleration based on KV cache management. *Transactions on Machine Learning Research (TMLR)*, 2025b.

Jindong Li, Yali Fu, Jiahong Liu, Linxiao Cao, Wei Ji, Menglin Yang, Irwin King, and Ming-Hsuan Yang. Discrete tokenization for multimodal llms: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026. doi: 10.1109/tpami.2026.3676982.

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. In *International Conference on Machine Learning (ICML)*, 2025c.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Andreas Liesenfeld and Mark Dingemanse. Rethinking open source generative AI: Open-washing and the EU AI act. In *ACM Conference on Fairness, Accountability, and Transparency (FAccT)*, 2024.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. QServe: W4A8KV4 quantization and system co-design for efficient LLM serving. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025.

Lucas D. Lingle. Transformer-VQ: Linear-time transformers via vector quantization. In *International Conference on Learning Representations (ICLR)*, 2024.

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. In *International Conference on Machine Learning (ICML)*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit LLMs: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 2024.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations (ICLR)*, 2017.

Fanxu Meng, Pingzhi Tang, Xiaojuan Tang, Zengwei Yao, Xing Sun, and Muhan Zhang. TransMLA: Multi-head latent attention is all you need. *arXiv preprint arXiv:2502.07864*, 2025.

Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite scalar quantization: VQ-VAE made simple. In *International Conference on Learning Representations (ICLR)*, 2024.

Helen Nissenbaum. *Privacy in Context: Technology, Policy, and the Integrity of Social Life*. Stanford University Press, 2010.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state RNNs. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Aidan Scannell, Mohammadreza Nakhaeinezhadfard, Kalle Kujanpää, Yi Zhao, Kevin Sebastian Luck, Arno Solin, and Joni Pajarinen. Discrete codebook world models for continuous control. In *International Conference on Learning Representations (ICLR)*, 2025.

Elizabeth Seger, Aviv Ovadya, Divya Siddarth, Ben Garfinkel, and Allan Dafoe. Democratising AI: Multiple meanings, goals, and methods. In *AAAI/ACM Conference on AI, Ethics, and Society (AIES)*, 2023.

Anupama Sridhar and Alexander Johansen. Echo: KV-cache-free associative recall with spectral Koopman operators. In *ACM Conference on AI and Agentic Systems (CAIS)*, 2026.

Richard M. Stallman. *Free Software, Free Society: Selected Essays of Richard M. Stallman*. GNU Press, 2015.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.

Philippe Tillet, H. T. Kung, and David Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*, 2019.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering. *arXiv preprint arXiv:2308.10248*, 2023.

Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Yixuan Wang, Qingyu Shi, Jiayu Zhou, Dianbo Liu, Ziwei He, and Zhouhan Lin. VQKV: High-fidelity and high-ratio cache compression via vector-quantization. *arXiv preprint arXiv:2603.16435*, 2026.

David Gray Widder, Meredith Whittaker, and Sarah Myers West. Why 'open' AI systems are actually closed, and why this matters. *Nature*, 2024.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations (ICLR)*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Dingyu Yao, Chenxu Yang, Zhengyang Tong, Zheng Lin, Wei Liu, Jian Luan, and Weiping Wang. VecInfer: Efficient LLM inference with low-bit KV cache via outlier-suppressed vector quantization. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. TurboQuant: Online vector quantization with near-optimal distortion rate. In *International Conference on Learning Representations (ICLR)*, 2026.

Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. SoundStream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2022.

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. *Proceedings of the ACM on Management of Data (SIGMOD)*, 2025.

Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. KV cache is 1 bit per channel: Efficient large language model inference with coupled quantization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

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. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Jonathan Zittrain. *The Future of the Internet – And How to Stop It*. Yale University Press, 2008.

CONTENTS

A Limitations **16**
A.1 Future Directions . . . 17

B Extended Background and Related Work **17**
B.1 Attention Memory and Sequence-Length Scaling . . . 17
B.2 Extended KV Cache Taxonomy . . . 18
B.3 Weight Compression . . . 19
B.4 Rate–Distortion Bounds . . . 19

C Experimental details **19**
C.1 Models . . . 19
C.2 Finetuning . . . 19
C.3 Conversion . . . 20
C.4 Evaluation . . . 21
C.5 Compute . . . 22
C.6 Implementation . . . 22

D Additional results **23**
D.1 Qwen3-8B-Base . . . 23
D.2 Long context . . . 23
D.3 Combining Transformer-FSQ with eviction . . . 25
D.4 Decode throughput . . . 25
D.5 Empirical Memory Footprint and Batch Scaling . . . 26

E Ablations **28**
E.1 Variants in the main figures . . . 28
E.2 MLA-FSQ . . . 28
E.3 FSQ-pre and Positional Encodings . . . 28
E.4 Discreteness and Coordinate Allocations . . . 29
E.5 Straight-Through Estimation vs Continuous Relaxations . . . 30
E.6 Joint End-to-End Training vs Layerwise Conversion . . . 30
E.7 Code utilisation and entropy coding . . . 30
E.8 Nonlinear encoders . . . 30
E.9 Per-head codes . . . 31
E.10 Recent-token window on Transformer-FSQ . . . 31
E.11 Finetuning settings . . . 31

F Architecture diagrams **31**
F.1 The FSQ bottleneck . . . 31

G Per-task results in full **31**
G.1 Evaluation Benchmarks and Notation . . . 32
G.2 Decode Throughput Breakdown . . . 33
G.3 Per-Task Long-Context Retrieval: RULER . . . 37
G.4 Per-Task Long-Context Results: LongBench . . . 39

A LIMITATIONS

Decode throughput at intermediate ratios is constrained by per-step RoPE key rotation. Transformer-FSQ decodes slower than uncompressed baselines at intermediate compression ratios (up to $96\times$ on 186M and $32\times$ on Qwen3-0.6B-Base; Table 9). Coupling discrete codes with standard Rotary Position Embeddings requires dequantising and rotating keys across all cached positions at every generation step. While FSQ-pre eliminates per-position rotations and restores decode speedups (§5.2), it precludes query-key normalisation. Emerging architectures without per-step rotary keys (e.g., NoPE or ALiBi) natively avoid this key dequantization bottleneck during attention.

Model scale and seed budgets are bounded by compute allocation. Multi-seed sweeps across compression ratios and decode profiling are evaluated up to 1.7B parameters. On Qwen3-8B-Base, we evaluate Transformer-FSQ, Palu, and TransMLA on RULER (Table 15) and LongBench (Figure 5) using 50 items and 200 documents with a single finetuning seed, whereas smaller models average up to three seeds; 8B decode throughput was not timed. Reported 8B uncertainty intervals are bootstrapped over benchmark items alone rather than training seeds. While 8B runs confirm scaling trends, complete multi-seed throughput profiling remains limited to models $\leq$ 1.7B.

Straight-through estimation and universal baseline learning rates. We pass gradients through discrete rounding with a straight-through estimator (STE), where query-key normalisation reduces seed variance by $3.3\times$–$4.3\times$ at 32 B per token per layer on 186M. The universal finetuning learning rate (10^{-5}) was selected on Transformer-FSQ (Appendix E.11) and applied uniformly to Palu and TransMLA without method-specific tuning. Learning-rate sensitivity probes across 3×10^{-6} to 10^{-4} confirm that this default does not penalise Palu or TransMLA validation bits-per-byte (1.0876 on Palu-256, 0.9741 on TransMLA-256 under $\tau = 0.25$). Standard STE optimization with uniform learning rates yields stable convergence across methods without hyperparameter tuning.

Evaluation precision, boundary sensitivity, and low-precision deployment. Primary RULER evaluations run in float32 precision across all 13 official tasks (50 examples per task) on all three Qwen sizes at $8{,}192$ context. An exploratory bfloat16 pass dropped a 128 B per token per layer configuration from 0.221 to 0.000 because bfloat16 activation noise from the frozen backbone perturbs pre-quantisation representations across discrete boundaries (mean flip rate 0.064 by layer 14, shifting attention mass by up to 0.78). Profiling shows that forcing only the internal FSQ encoder projections to float32 does not prevent this collapse, because rounding errors accumulate across cascading transformer layers. Because Transformer-FSQ stores discrete integer coordinates ($\{0, \ldots, 15\}$), physical cache footprint and compression ratios are mathematically identical across compute precisions. While casting weights to float32 doubles static parameter memory (2.50 GB vs 1.25 GB on Qwen3-0.6B-Base), empirical profiling (Appendix D.5) shows that the packed integer cache size is invariant (28.15 MB) and transient attention allocations during prefill dominate peak device memory during long-context batch evaluation. Standard quantization-aware training techniques—specifically injecting boundary noise into the pre-quantization latents during recovery finetuning or adding soft-margin separation losses—provide a concrete path to making discrete latents robust to noisy low-precision backbones (e.g., FP4 or GGUF integer formats). Evaluating in float32 isolates representational capacity from accumulation noise without altering the compressed KV cache.

High-dimensional product grids operate beyond initial low-dimensional validations. While Mentzer et al. (2024) recommend $L_i \geq 5$ levels for dimensions $d \leq 6$ (implicit codebooks $\leq 2^{16}$), we deploy $L = 16$ across latent widths from 16 to $4{,}096$. At wide settings, rounding thousands of coordinates independently acts as product scalar quantisation on a learned basis rather than joint vector quantisation, foregoing multidimensional cell shaping. Robust downstream retrieval across benchmarks confirms that high-dimensional product grids remain effective despite operating orders of magnitude beyond prior validation bounds.

Baseline decode throughput measurements describe our reference implementations rather than custom vendor kernels: VecInfer uses an unfused reference path, Palu's released 16-bit kernel does not dispatch on our quantised 4-bit latents, and KIVI uses its released kernel for values only (Appendix D.4). Our evaluations run base models without chat templates; while absolute task scores are lower than published instruct results, relative accuracy retention across compression ratios is closely matched. Full implementation details appear in the submitted codebase.

Cache memory is verified analytically at conversion. Because HuggingFace cache objects maintain padded allocations to preserve attention masks, we verify physical footprint by counting stored bytes per token at conversion, rejecting deviations from target budgets. Measured GPU memory

directly confirms this analytical model (Figure 13a, Appendix D.5): on Qwen3-0.6B-Base at $8{,}192$ tokens, the physical buffer matches analytic cache size to within 0.05% (28.15 MB measured vs 28.16 MB analytic at 128 B per token per layer; peak memory 7.08 GB vs 6.14 GB uncompressed). Analytic byte accounting accurately reflects physical KV-cache memory requirements despite runtime framework allocation padding.

Our long-context evaluation measures retrieval, tracking and question-answering accuracy at one context length, $8{,}192$ tokens; it does not test arithmetic reasoning, code generation or many-shot in-context learning under compression, where prior work reports sharper degradation than on retrieval-style tasks. A method's score here does not show that it preserves multi-step reasoning under compression, which we leave to future measurement.

A.1 Future Directions

Several promising research directions follow from our findings:

Dynamic and Hierarchical Compression. Dynamic compression regimes could accommodate changing context demands, compressing lightly while sequence length is short and progressively increasing compression as the task lengthens. Discrete latents lend themselves to such hierarchical or tree-based compression schemes: coarsening a stored code by rounding it to fewer levels is a straightforward truncation, but refining a coarse code back to finer resolution is not, since rounding discards information the coarse code alone cannot recover. Whether a richer stored representation, or recomputation from the original activation, could support this kind of on-demand refinement is an untested hypothesis, not a property of the bottleneck as we train it here.

Boundary Regularization for Low-Precision and Quantized Backbones. In our setup, straight-through estimation finetunes linear decoders against unperturbed grid coordinates, leaving the latent boundary regions sensitive to activation jitter. Introducing noise injection during recovery finetuning or training with an explicit boundary margin penalty could regularize representations away from quantization thresholds, enabling robust execution atop low-bit backbones such as FP4, INT4, and GGUF quantizations.

Variable Layer Allocations and Distillation. The bottleneck could allocate variable code sizes across layers, as PyramidKV does for tokens (Cai et al., 2024), share projection tensors across layers, or be trained by distilling representations directly from the uncompressed model rather than on the language-modelling loss alone.

Integer Attention Kernels. Integer codes could be integrated with weight quantisation (Dettmers & Zettlemoyer, 2023), enabling low-precision integer attention kernels without materialising floating-point keys.

Coordinate Allocation Sweeps for Mild Compression. Sweeping the combinatorial design space of quantization levels and coordinate dimensions across layers could eliminate the minor accuracy loss observed at mild ($2\times$–$4\times$) compression ratios (Appendix E), adapting the bottleneck to devote more levels to fewer dimensions when compression pressure is low.

Controllable Steering and Recurrent Latents. Discrete codes offer a structured interface to the model: they could be inspected as interpretable features (Bricken et al., 2023), manipulated to steer generation (Turner et al., 2023), or aggregated recurrently in place of attention (Katharopoulos et al., 2020), yielding a state that does not grow with context.

B Extended Background and Related Work

B.1 Attention Memory and Sequence-Length Scaling

Under standard multi-head attention (Vaswani et al., 2017), computing the output for each query token requires inner products against every preceding key, followed by a weighted aggregation over every preceding value. Consequently, decoding step t must access $t-1$ cached key-value pairs per layer. For long sequences and modern LLM serving, this per-token cache storage quickly surpasses the static memory required for model weights.

FlashAttention and compute optimisations. FlashAttention (Dao et al., 2022) restructures attention computation using tiling and online softmax rescaling to minimise memory bandwidth between high-bandwidth memory (HBM) and on-chip SRAM. While FlashAttention achieves dramatic speedups during prefill and generation without materialising the intermediate $T \times T$ attention

score matrix, it leaves the persistent key-value cache unchanged: each key and value vector must still reside in HBM for subsequent autoregressive decoding steps.

Linear attention and recurrent state-space models. A parallel line of research replaces the growing per-token cache with a fixed-size recurrent hidden state. Linear attention methods (Katharopoulos et al., 2020; Lingle, 2024) apply kernel feature maps to decouple query-key computation, allowing the attention sum to be computed as an associative scan. State-space models (SSMs) and hybrid architectures such as Mamba and Echo (Gu et al., 2022; Sridhar & Johansen, 2026) maintain a constant-size hidden representation across arbitrary sequence lengths. However, these paradigms typically require pretraining alternative architectures from scratch, and their fixed-size state gives them no mechanism to hold an arbitrary earlier token for later exact retrieval the way a per-token cache does. We do not evaluate them here; Appendix A states this scope. In this work, we focus exclusively on techniques that retain a per-token cache for pretrained standard transformers.

B.2 Extended KV Cache Taxonomy

Methods that compress the per-token KV cache operate along different dimensions: along the sequence length (token eviction), along coordinate precision (scalar quantisation), along latent subspace projections (low-rank factorisation), and along coordinate clusters (vector quantisation).

Token eviction. Token eviction methods discard positions judged uninformative. During generation, StreamingLLM (Xiao et al., 2024) observes that preserving initial "attention sinks" alongside a local sliding window prevents catastrophic perplexity spikes in autoregressive generation. H2O (Heavy Hitter Oracle) (Zhang et al., 2023) dynamically tracks tokens with the highest accumulated attention weights, continually evicting positions falling outside the heavy-hitter set and local window. TOVA (Oren et al., 2024) maintains a fixed token quota by evicting the token with the lowest attention score at each decode step. For prompt compression, SnapKV (Li et al., 2024) uses an observation window at the end of the prompt to identify consistently attended prompt tokens. PyramidKV (Cai et al., 2024) extends this insight by allocating non-uniform token capacities across layers: early layers retain broader dispersed contexts while deeper layers enforce sharper pruning. However, because eviction reduces token count rather than per-token width, the resulting cache memory remains tied to prompt and generation dynamics.

Low-rank projection. Low-rank methods exploit cross-head redundancy by projecting key and value vectors into narrower continuous subspaces. Multi-head Latent Attention (MLA) (DeepSeek-AI, 2024) replaces separate per-head keys and values with a shared low-rank latent vector while using a dedicated rotary key for positional encoding. TransMLA (Meng et al., 2025), MHA2MLA (Ji et al., 2025), and X-EcoMLA (Li et al., 2025a) demonstrate that standard multi-head attention (MHA) and grouped-query attention (GQA) models can be converted into MLA post-training via rank-reduction and finetuning. Palu (Chang et al., 2025) computes low-rank SVD approximations of existing projection weights, caching intermediate latent vectors that are projected back to key and value dimensions at attention time. While continuous low-rank methods reduce cache width, their representations remain unquantised floating-point vectors unless paired with subsequent quantisation stages.

Scalar quantisation. Scalar quantisation maps individual coordinates onto low-bit representations. KIVI (Liu et al., 2024) quantises keys per channel and values per token to 2 bits without requiring fine-tuning, while preserving a small full-precision sliding window. KVQuant (Hooper et al., 2024) applies non-uniform scalar quantisation pre-RoPE with dense outlier handling. GEAR (Kang et al., 2024) compresses the cache to 2–4 bits and approximates quantisation residual error with low-rank and sparse matrices. QServe (Lin et al., 2025) integrates 4-bit KV caching alongside 4-bit weights and 8-bit activations. These methods retain original activation dimensions and rely on per-channel or per-token scaling factors and auxiliary buffers.

Vector quantisation of the KV cache. Vector quantisation (VQ) maps activation blocks to discrete centroids in a codebook. VQ-VAE (van den Oord et al., 2017) and Product Quantisation (Jégou et al., 2011) laid the groundwork for discrete representations in continuous spaces. VQ-VAE trains through a straight-through gradient estimator (Bengio et al., 2013) because the codebook lookup, a nearest-neighbour argmin, is not differentiable; the Gumbel-softmax relaxation (Jang et al., 2017; Maddison et al., 2017) is an alternative route around the same non-differentiability, and Scannell et al. (2025) use a discrete codebook as the latent state of a world model for continuous control. In LLM caching, Coupled Quantisation (Zhang et al., 2024) groups 2 to 8 channels into learned codebooks without finetuning, though storing 67.1M centroid parameters for LLaMA-7B beside the

cache. CommVQ (Li et al., 2025c) trains additive multi-stage codebooks using 2×2 block structures that commute with RoPE. RVQ-KV (Kumar, 2024) adapts residual vector quantisation with sequential codebook stages. VecInfer (Yao et al., 2026) combines per-channel smoothing and Walsh–Hadamard transforms with offline-fitted codebooks at 1.25 to 4 bits per coordinate. GSRQ (Kim et al., 2026) decouples magnitude from direction in residual updates, achieving sub-1-bit perplexity. PQCache (Zhang et al., 2025) utilises product quantisation to retrieve top-k attention states while storing the complete cache on host memory. NOVA-KV (Fernández-Menduiña et al., 2026) and VQKV (Wang et al., 2026) also explore codebook vector quantisation for keys and values. TurboQuant (Zandieh et al., 2026) applies random orthogonal rotations to concentrate coordinate distributions, followed by precomputed scalar quantisation grids.

B.3 Weight Compression

Weight compression addresses the static portion of the GPU memory budget. For post-training weight quantisation, empirical scaling studies demonstrate that a larger model at lower precision consistently outperforms a smaller model at higher precision under identical memory budgets, with 4-bit weights achieving near-optimal performance across diverse model architectures (Dettmers & Zettlemoyer, 2023). Quantisation-aware pretraining pushes this frontier further, achieving viable ternary and 1.58-bit models (Ma et al., 2024). However, while weight footprint remains constant regardless of sequence length, the KV cache scales linearly with generation context, rendering cache compression increasingly dominant for extended agentic executions.

B.4 Rate–Distortion Bounds

Theoretical bounds. Rate–distortion theory characterises the fundamental trade-off between bit-rate and reconstruction error for arbitrary quantisers. Zandieh et al. (2026) prove that for any quantiser operating at b bits per coordinate, there exists a unit vector whose expected squared reconstruction error is lower-bounded by 4^{-b}. Their construction approaches within a factor of 2.7 of this lower bound. However, this theoretical bound addresses the fidelity of reconstructing a given, arbitrary input vector. Transformer-FSQ does not attempt to reproduce the continuous key and value activations of a frozen model. Instead, its linear encoder and decoder are trained end-to-end such that the representations surviving the discrete integer bottleneck optimise downstream attention task performance. This task-aware training changes what is minimised, a downstream loss rather than reconstruction error, but it does not escape the rate–distortion bound itself: at a fixed bit rate, some distortion is unavoidable under any objective, and training only chooses which directions absorb it.

C Experimental details

C.1 Models

Table 3 summarizes architectural specifications and uncompressed cache sizes (Eq. 7) for all benchmarked models: our custom-pretrained 186M transformer (`d12w768`, pretrained with nanochat (Karpathy, 2025) for architecture ablations and decode profiling) alongside the three released transformers evaluated on long-context benchmarks: Qwen3-0.6B-Base, Qwen3-1.7B-Base, and Qwen3-8B-Base (Yang et al., 2025), loaded through the Transformers library (Wolf et al., 2020). Each Qwen model loads by its published model identifier with no revision pinned, resolving to whichever snapshot the public host serves; we did not record a commit hash for the downloaded weights.

C.2 Finetuning

Table 4 gives the finetuning hyperparameters. All released models are finetuned using AdamW with learning rate 10^{-5} and constant weight decay 0.28 on ClimbMix in bfloat16, matching the recipe verified on Qwen3-0.6B-Base (Appendix E.11) without method-specific tuning. The custom 186M model continues under its pretraining recipe (Muon combined with AdamW). Checkpoints are evaluated in float32 to eliminate numerical coordinate flipping (Appendix C.4). Uniform training schedules guarantee that accuracy differences reflect compression architectures rather than optimization discrepancies.

Training budgets scale with model parameters across hardware configurations. Each run trains on τN tokens, where $\tau \in \{0.025, 0.075, 0.25\}$ tokens per parameter (N being the base model parameter count). Qwen3-8B-Base runs exclusively at $\tau = 0.25$ across 4 GH200 GPUs under

Table 3: **Model specifications and memory footprints on 16 GiB GPU:** Architectural specifications and memory allocations across the benchmarked transformer architectures. c_{full} is the uncompressed half-precision cache size per token per layer (Eq. 7), serving as the reference for all reported compression ratios. Uncompressed context limits define the baseline reference points in Figure 11. Larger models deliver higher baseline accuracy but exhaust available memory rapidly as context grows. ‡Qwen3-8B-Base runs one seed throughout – RULER and LongBench, finetuned and training-free alike (§5.1).

	186M (d12w768)	**Qwen3-0.6B-Base**	**Qwen3-1.7B-Base**	**Qwen3-8B-Base**
parameters	185,598,866	596,049,920	1,720,574,976	8,190,735,360
layers	12	28	28	36
model width C	768	1024	2048	4096
query heads	12	16	16	32
key-value heads n_{kv}	12	8	8	8
head width d_h	64	128	128	128
vocabulary	16,384	151,936	151,936	151,936
sequence length (pretraining)	2048	4096, then 32,768		
sequence length (finetuning)	2048	1024		
uncompressed cache c_{full}	3072 B	4096 B		
weights in half precision	0.35 GiB	1.1 GiB	3.2 GiB	15.3 GiB
cache per token, all layers	36 KiB	112 KiB	112 KiB	144 KiB
cache for 200,000 tokens	7 GiB	21 GiB	21 GiB	27 GiB
as a multiple of the weights	20×	19×	7×	2×
tokens that fit on a 16 GiB GPU	456 k	139 k	120 k	5 k
seeds per configuration	5	3	3	1‡

data parallelism with micro-batch 1 sequence (1,024 tokens) and gradient accumulation 8, yielding an effective batch of 8,192 tokens per GPU and a global effective batch of 32,768 tokens (32 sequences) while maintaining identical total training tokens. Parameter-normalized budgets ensure equitable computational investment across all model sizes.

Table 4: **Finetuning hyperparameters across evaluated model architectures:** Optimisation hyperparameters for post-conversion training across architectures. Finetuning budget τ specifies tokens per parameter (0.025, 0.075, and 0.25), where $\tau = 0$ corresponds to conversion without finetuning. All methods at a given model scale share identical datasets, optimisers, and budgets. The effect of τ on accuracy is reported for the 186M model and Qwen3-0.6B-Base in Appendix E.11; the released models' RULER and LongBench results (§5.2) use $\tau = 0.25$ throughout.

	pretrained here	**Qwen3-0.6B / 1.7B-Base**	**Qwen3-8B-Base**
finetuning budgets τ	0.025, 0.075, 0.25	0.025, 0.075, 0.25	0.25
optimiser	Muon + AdamW	AdamW	AdamW
learning rate	pretraining schedule	10^{-5}	10^{-5}
weight decay	0.28, cosine-decayed	0.28, constant	0.28, constant
data	ClimbMix	ClimbMix	ClimbMix
sequence length	2048	1024	1024
trainable tensors	encoder, decoder	encoder, decoder	encoder, decoder
backbone	frozen	frozen	frozen

C.3 CONVERSION

The bottleneck projections are initialized in closed form using principal component analysis (PCA). The encoder projects hidden states along principal directions scaled onto the $\tanh$ bounding range, while the decoder is fitted via ridge regression (penalty 1.0) to reconstruct hidden states before composing with original key and value projections. In Eq. 8 (Appendix F.1), half-width $h = (L-1)(1+\varepsilon)/2$ and shift $\delta = \frac{1}{2}$ (for even L) ensure discrete levels align with integers. When the latent coordinate width G exceeds the hidden dimension C ($G > C$, as occurring at mild compression ratios), the encoder basis cycles through the C orthonormal principal directions $i \bmod C$, assigning multiple coordinates to the directions of highest activation variance. Specifically, for calibration hidden activations $X \in \mathbb{R}^{m \times C}$ and quantized latents $Z \in \mathbb{R}^{m \times G}$, the linear decoder reconstructing residual activations is $D = (Z^\top Z + \lambda I_G)^{-1} Z^\top X \in \mathbb{R}^{G \times C}$, giving com-

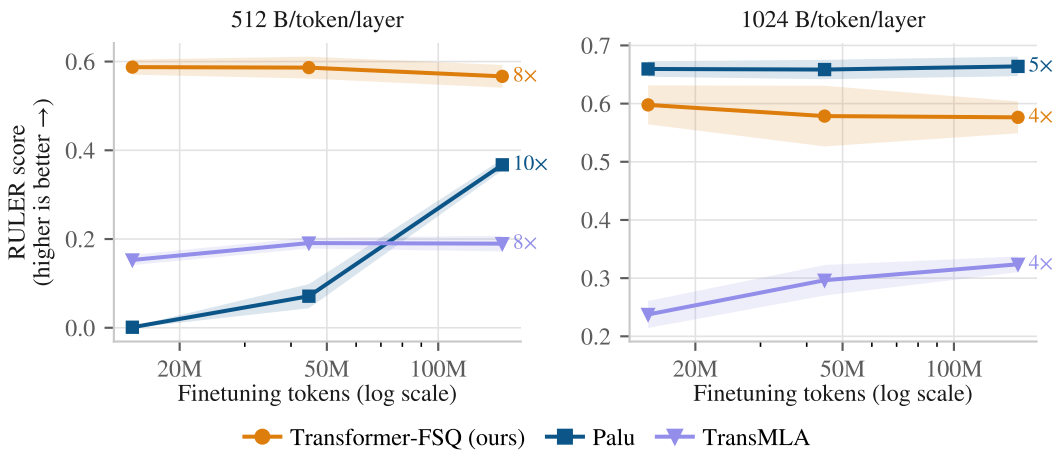

Figure 7: **Finetuning budget across training tokens:** RULER score on Qwen3-0.6B-Base against finetuning tokens (0.025, 0.075 and 0.25 per parameter) at nominal 512 and 1,024 B per token per layer; labels give each cache's achieved ratio (Palu spends 400 and 784 B). Transformer-FSQ reaches its accuracy plateau with fewer finetuning tokens than the baselines.

posite decoder projection $W_{kv} = [W^K; W^V]D^\top \in \mathbb{R}^{2D_{kv} \times G}$, ensuring numerical stability without rank deficiency. The converted bottleneck replaces $2Cn_{kv}d_h$ parameters with $G(C + 2n_{kv}d_h)$, adding under 10M parameters at 97.5× on Qwen3-1.7B-Base ($G = 84$). Closed-form PCA initialization provides a well-conditioned starting point prior to gradient finetuning.

Conversion fits the inserted tensors in closed form (§4.2) from activations captured on a calibration sample of ClimbMix: 32,768 tokens for the released models, Qwen3-8B-Base included. For the model we pretrain, 262,144 tokens are drawn and the first 4,096 used. The sample is read deterministically from the start of the training shards, so it is identical across seeds, and without finetuning, a seed changes only the evaluation subsample for methods fitted in closed form. We report conversion without finetuning for Palu at its mildest ratio on Qwen3-0.6B-Base; every TransMLA configuration is finetuned.

C.4 Evaluation

RULER (Hsieh et al., 2024), the suite behind every RULER number this paper reports (§5.1), evaluates all 13 official tasks at 8,192 tokens with 50 items per task across all three Qwen sizes, drawn once from a fixed item seed and shared by every configuration. Scoring follows RULER's published protocol: exact match on single-answer tasks, reference recall on multi-answer tasks, and string containment on QA tasks, capped at official generation lengths. As detailed in §5.1, evaluation forward passes run in float32 to prevent upstream bfloat16 activation drift across discrete grid boundaries.

LongBench (Bai et al., 2024) runs `narrativeqa`, `gov_report` and `trec` at 20 examples per task, scored by F1, ROUGE-L and classification accuracy (0 to 100). On Qwen3-8B-Base the LongBench draw is 200 documents per task, with prefill out-of-memory on the longest documents leaving 180 and 70 scored for `gov_report` and `narrativeqa` (Appendix D). Prompts exceeding 32,768 tokens are middle-truncated to 32,736 tokens. Long-context benchmarks run three seeds at 0.25 tokens per parameter, except Qwen3-8B-Base, which runs one seed throughout (§5.1).

Benchmark selection. We select RULER and LongBench to probe both structured associative retrieval and realistic document comprehension. Vanilla single-needle retrieval is insufficiently discriminative, as coarse scalar quantization or naive token retention can retrieve isolated keys against uniform text without preserving relational context. In contrast, RULER tests 13 synthetic patterns—including multi-key, multi-value, and variable tracking—directly exposing when aggressive compression destroys token relationships. LongBench complements synthetic retrieval by evaluating multi-page real documents on summarization, narrative QA, and classification.

Unless stated otherwise, a spread is a sample standard deviation across seeds, and intervals are bootstrapped over items stratified by task.

C.5 COMPUTE

Computation is tracked across hardware types for all training and evaluation phases. Conversion and finetuning run on NVIDIA GH200 GPUs (1 GPU per run for ≤ 1.7B, 4 GPUs for 8B), totalling 2304 GPU-hours (Table 5). Decode throughput profiling required 54 GPU-hours on NVIDIA L40S, 186M pretraining took 3.5 GPU-hours, and downstream evaluations ran across GH200 nodes. Comprehensive compute accounting demonstrates that post-training conversion requires minimal hardware investment compared to pretraining.

Table 5: **Conversion and finetuning compute by model:** GPU-hours on NVIDIA GH200 GPUs, one GPU per run except Qwen3-8B-Base's four (§C.5).

Model	GPU-Hours
186M	161
Qwen3-0.6B-Base	418
Qwen3-1.7B-Base	480
Qwen3-8B-Base	1245
Total	**2304**

C.6 IMPLEMENTATION

All models are implemented in PyTorch $2.9.1$ (Paszke et al., 2019) against CUDA 12.8 wheels, and the fused decode kernels are written in Triton $3.5.1$ (Tillet et al., 2019) and CUDA. The Qwen conversion and finetuning path loads models through Transformers (Wolf et al., 2020) ($4.57.6$). Baselines follow published specifications: KIVI preserves its 128-token full-precision window, while VecInfer keeps 4 sink tokens following StreamingLLM (Xiao et al., 2024). PyramidKV retains per-head token selections in primary benchmarks, matching its published SnapKV-derived specification. A unified software stack ensures controlled comparisons across all evaluated compression paradigms.

Specific baseline deviations reflect architectural unification constraints in our benchmark harness. VecInfer uses shared k-means codebooks across heads and symmetric bit allocations, as its proprietary fused kernel was not integrated. For stacking experiments (Appendix D.3), PyramidKV adopts shared token selection across heads to align with unified FSQ coordinate blocks, improving standalone RULER accuracy by 13–24 points. Transformer-FSQ omits sliding windows and sinks except in explicit ablation runs (Appendix E). Documenting baseline adaptations ensures reproducibility while explaining implementation-specific performance characteristics.

Justification of baseline methods: Eviction and scalar quantization. We compare against representative methods across all established KV-cache compression paradigms (Table 1), selecting the strongest published variant in each class. For token eviction, we select PyramidKV (Cai et al., 2024) over SnapKV, StreamingLLM, and H2O because PyramidKV dynamically shrinks cache budgets with layer depth to match empirical attention concentration. For scalar quantisation, we choose KIVI (Liu et al., 2024) over uniform round-to-nearest (RTN) methods, as KIVI mitigates activation outliers via per-channel key and per-token value quantisation alongside a full-precision window. These baselines represent state-of-the-art training-free approaches in eviction and scalar precision reduction.

Vector quantization and low-rank projection baselines. Continuous latent and codebook-based compression are represented by state-of-the-art formulations. For vector quantisation, we benchmark VecInfer (Yao et al., 2026) over Coupled Quantization, using Walsh–Hadamard transforms to disperse outliers without storing tens of millions of codebook parameters. For low-rank compression, we evaluate Palu (Chang et al., 2025) and TransMLA (Meng et al., 2025), which adapt pretrained checkpoints via post-training finetuning, avoiding the extreme compute cost of pretraining MLA architectures from scratch (DeepSeek-AI, 2024). VecInfer, Palu, and TransMLA establish strong compression baselines for codebooks and low-rank latent projections.

Theoretical rotation baselines and evaluation scope. Advanced randomized rotations complement our empirical baseline suite. We include TurboQuant (Zandieh et al., 2026) to represent recent theoretical advances in random orthogonal rotations paired with scalar coordinate quantisation grids. Baselines are restricted to per-token cache architectures; fixed-size state-space or linear attention

models are excluded as they alter underlying transformer mechanics. This curated suite provides rigorous coverage across all primary architectural strategies for KV-cache compression.

D ADDITIONAL RESULTS

D.1 QWEN3-8B-BASE

Qwen3-8B-Base retrieval and question classification. Figure 2 reports RULER and Figure 5 reports `gov_report` (50 RULER examples, 200 LongBench documents, seed 42 throughout). At extreme compression, Transformer-FSQ at 409.6× and Palu at 512× reach floor retrieval scores, mirroring smaller model behavior. On `trec` (Figure 10; 200 documents, uncompressed score 63.0), KIVI and VecInfer stay close to uncompressed at mild compression (58.0 at 4.7×, 59.0 at 5.5×) before dropping. Transformer-FSQ maintains 34.5 at 32× and 3.5 at 97.5×, outperforming Palu (2.0 and 2.0). TransMLA ran out of memory during long-document prefill and is omitted. Transformer-FSQ retains robust question classification accuracy into extreme compression regimes on 8B models.

Document-level narrative comprehension on 8B parameters. On LongBench `narrativeqa` (Figure 9; 70 documents evaluated, the rest exceeding prefill memory across all configurations), the uncompressed model scores 17.4 F1. Token eviction performs strongly on this narrative task: PyramidKV scores 18.8 at 26.6× and 11.5 at 197.8×. Transformer-FSQ scores 8.4 (32×) and 7.2 (97.5×), exceeding Palu's 3.0 and 2.2. Token eviction excels at selective narrative tracking, while discrete FSQ provides superior stability among uniform compression methods.

D.2 LONG CONTEXT

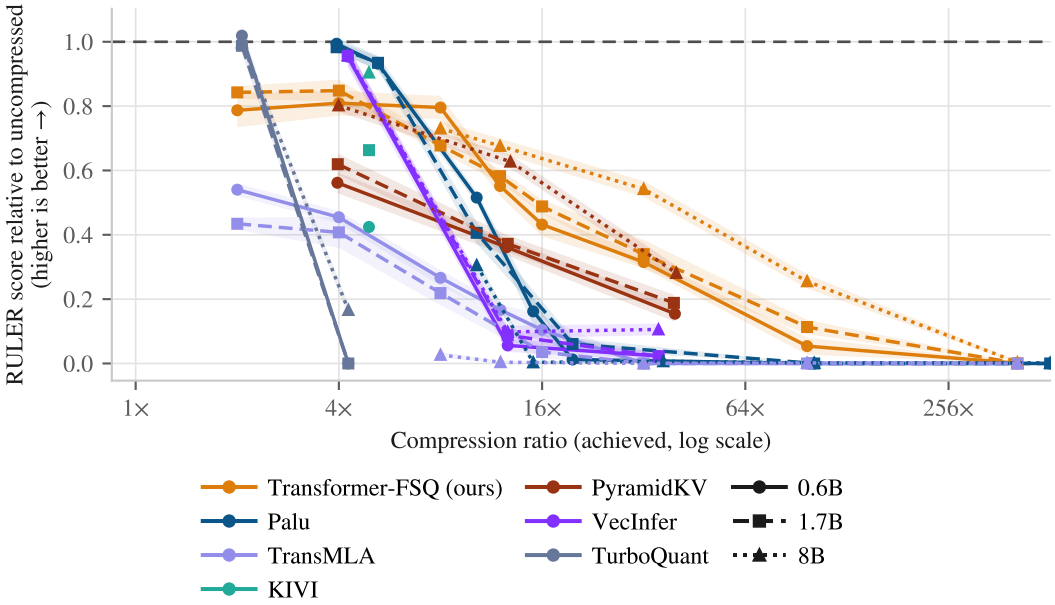

Figure 8: **Share of the uncompressed score each method keeps:** RULER score at 8,192 tokens divided by the same model's uncompressed score. Below about 5× VecInfer, and Palu where we ran it, keep the most; from 32× to 97.5× Transformer-FSQ does, on every model. Bands as in Figure 2.

This subsection evaluates long-context retrieval and comprehension across all three Qwen models under compression. Figure 8 presents RULER scores relative to uncompressed models, while Figures 9 and 10 report LongBench `narrativeqa` and `trec` performance across models. Complete per-task scores appear in Tables 13–15 and 16–18.

On Qwen3-0.6B-Base (Figure 8), Transformer-FSQ keeps 81% of the uncompressed RULER score at 4× and 80% at 8× (Table 13), ahead of TransMLA's 27% at 8× and Palu's 16% at 15.1×. Single-key retrieval (`niah_single_1`) survives longest at extreme compression, while multi-key retrieval with distractors and variable tracking collapse past 16×. Training-free baselines

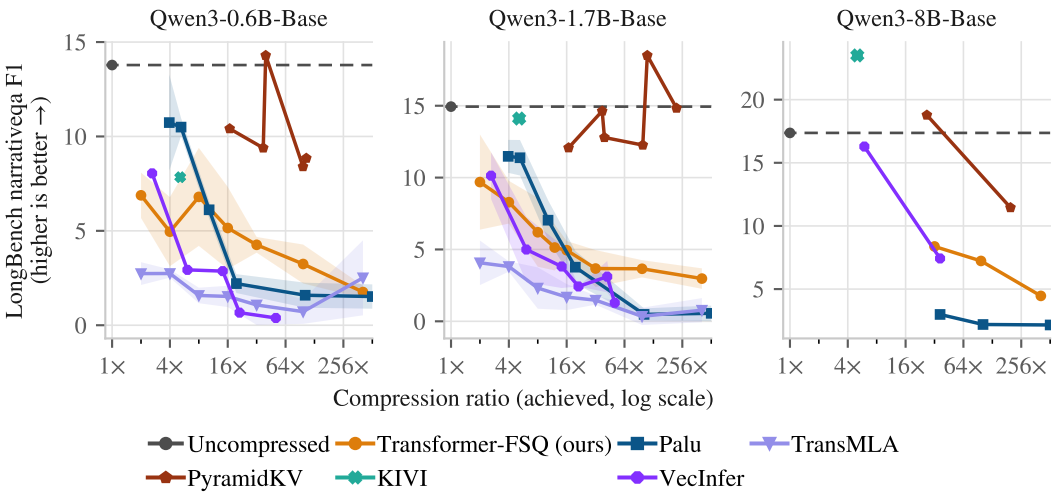

Figure 9: **LongBench `narrativeqa` F1 across compression ratios:** Raw F1 (0–100) vs achieved compression ratio across models; dashed line is uncompressed. On 0.6B and 1.7B, token eviction (PyramidKV) keeps the highest score; on 8B, Transformer-FSQ outperforms continuous low-rank projection (Palu). Solid: finetuned; dashed: training-free.

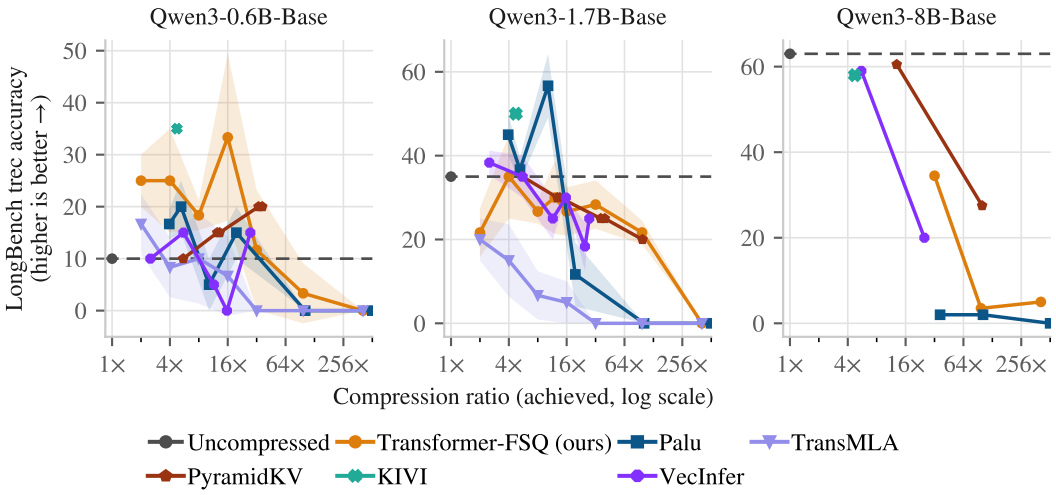

Figure 10: **LongBench `trec` classification accuracy across compression ratios:** Accuracy (0–100) vs achieved compression ratio across models; dashed line is uncompressed. Training-free methods maintain accuracy at mild compression, while Transformer-FSQ preserves classification into extreme ratios (32×–97.5×). Solid: finetuned; dashed: training-free.

drop earlier: VecInfer retains 95% at 4.3× but falls to 6% at 12.7×, and PyramidKV holds 56% at 4.0×. Transformer-FSQ extends retrieval fidelity on small models by up to 2× deeper compression than competing alternatives.

On Qwen3-1.7B-Base (Figure 8), the same hierarchy holds at higher retention: Transformer-FSQ keeps 68% of uncompressed RULER at 8× and 34% at 32× (Table 14), exceeding TransMLA (22% at 8×) and Palu (93% at 5.2×, falling to 6% at 19.7×). VecInfer drops to 9% at 12.7×, while PyramidKV holds 62% at 4.0×. Task-specific resilience matches the 0.6B pattern, with single-key retrieval surviving longest. Scaling to 1.7B preserves relative method ordering while substantially improving absolute retrieval accuracy under compression.

Document comprehension and question classification reveal complementary strengths between uniform compression and token eviction (Figures 9 and 10). On LongBench `narrativeqa`,

token eviction (PyramidKV) stays near uncompressed performance across budgets (8.4–14.3 F1 on 1.7B), outperforming uniform compression on long narrative tracking where broad context can be pruned. Conversely, on `trec` classification and `gov_report` summarization (Figure 5), Transformer-FSQ maintains superior accuracy across extreme compression ratios (32×–409.6×) where continuous baselines collapse. Eviction excels on selective narrative scanning, whereas discrete FSQ provides superior stability for structured context retention.

On Qwen3-8B-Base (Table 15), Transformer-FSQ keeps 54% of RULER at 32× and 26% at 97.5×, with single-key retrieval surviving longest and variable tracking scoring 0.55 at 32× (Appendix D.1). Per-task RULER breakdowns are in Tables 13–15 (§G.3), and LongBench breakdowns are in Tables 16–18 (§G.4).

D.3 Combining Transformer-FSQ with eviction

Stacking FSQ with token eviction on Qwen3-0.6B-Base. We applied PyramidKV at inference to fine-tuned Transformer-FSQ checkpoints without retraining, evaluating RULER (8,192 tokens) across combinations where FSQ carries at least half the log-compression (Table 6). Stacking FSQ 8× with PyramidKV 2× achieves 16× total compression (256 B/token/layer) and scores 0.40, outperforming PyramidKV alone at the matched budget (0.30 at 255.75 B). Stacking FSQ 16× with PyramidKV 2× achieves 32× total compression (128 B/token/layer) and scores 0.23, outperforming matched PyramidKV alone (0.10 at 127.75 B). At 64 B/token/layer (64×, FSQ 16× with PyramidKV 4×), the stack maintains 0.22, whereas standalone PyramidKV cannot reach a 64 B budget at 8,192 context due to minimum window and sink allocations. Combining discrete FSQ with mild token eviction preserves substantial accuracy advantages over standalone eviction at matched memory budgets.

Diminishing returns at extreme stacking ratios. Stacking exhibits diminishing returns as base FSQ compression increases, because compounding quantization and eviction errors accelerate context degradation (e.g., FSQ 32× stacked with PyramidKV 32× drops to 0.01). On Qwen3-1.7B-Base (three seeds), the $8\times+2\times$ stack scores 0.40 ± 0.02 against 0.32 for matched PyramidKV alone, with 8B single-seed rungs confirming this trend (Table 6, Figure 12). Figure 11 summarizes the resulting capability frontier on a 16 GiB GPU. Stacking is effective at intermediate budgets, whereas extreme compression is best served by standalone FSQ.

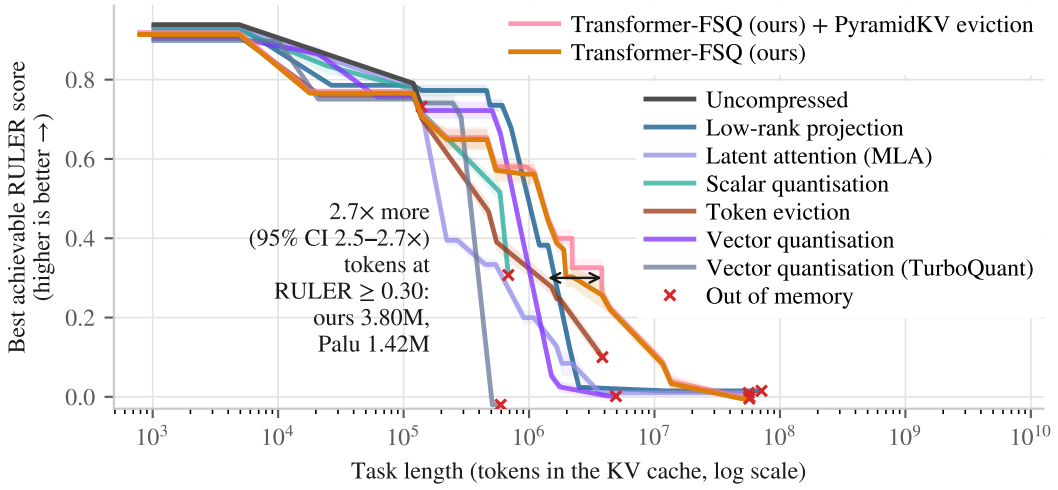

Figure 11: **Tokens each method holds on a 16 GiB GPU:** Projected memory capacity on a 16 GiB GPU associated with retrieval accuracy measured at 8,192 tokens, over all three models and every ratio measured; read a horizontal line to compare capacity reach at a given retrieval score. This illustrates capacity scaling rather than verified million-token retrieval. The stacked line uses only stacks in which Transformer-FSQ carries at least half of the compression; crosses: out of memory.

D.4 Decode throughput

Decode throughput profiling across models. Profiling decode throughput across models highlights the operational trade-off between memory bandwidth savings and per-step dequantization compute.

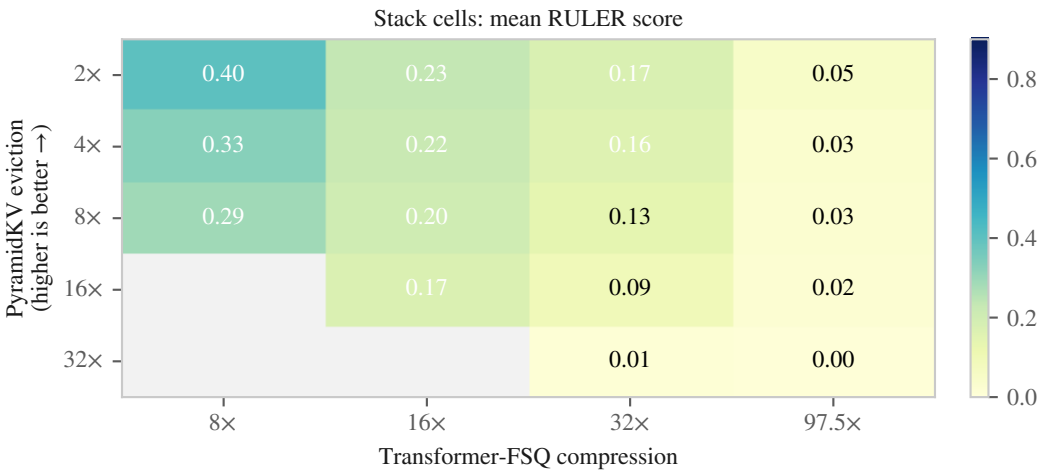

Figure 12: **Stack-grid mean RULER score at 8,192 tokens (Qwen3-0.6B-Base):** mean absolute RULER score for every Transformer-FSQ × PyramidKV cell that clears the pre-registered stacking rule (Transformer-FSQ carries at least half the compression in log terms, §5.2); grey cells do not qualify. Full numerical values, seed counts, and matched PyramidKV-alone controls appear in Table 6.

On 186M, Transformer-FSQ decodes at $0.10\times$ uncompressed speed at $12\times$ and $0.25\times$ at $32\times$, achieving throughput parity and speedups ($1.24\times$ median, up to $1.83\times$) at $384\times$ on long contexts (Table 9). Baselines run our reference implementations as their proprietary kernels do not dispatch on our quantized configurations. At $384\times$ and batch 16, Transformer-FSQ sustains 150,000 context tokens within a 230 MB cache on an NVIDIA L40S. Extreme discrete compression unlocks substantial decode acceleration on memory-bound long contexts.

On Qwen3-0.6B-Base, Transformer-FSQ's median decode throughput rises from $0.07\times$ the uncompressed model's at $2\times$ to $0.78\times$ at $32\times$ on an NVIDIA L40S. Each point sits at its configuration's planned cache size, which can differ from the target: two FSQ-small targets share one configuration.

D.5 Empirical Memory Footprint and Batch Scaling

Physical verification of analytical cache accounting. To verify that analytical KV-cache byte accounting matches physical memory allocations during execution, we profile resident GPU memory on Qwen3-0.6B-Base at an 8,192-token context window (7,855 prompt tokens in practice; Figure 13a). Across configurations, the physical buffer allocated for the KV cache matches the analytic formula within 0.05%. At 128 B per token per layer ($32\times$ compression), the measured cache buffer requires exactly 28,152,320 bytes (28.15 MB, or 1.005 MB per layer for 28 layers) compared to the sequence-scaled analytic calculation of 28.16 MB (and 29.36 MB at full 8,192 capacity). For FSQ at 512 B per token per layer ($8\times$ compression), measured cache occupancy is 112.61 MB (analytic 117.44 MB at full window). By comparison, the uncompressed reference cache requires 1,801.75 MB (1.80 GB, analytic 1.88 GB at full window). Physical GPU allocations confirm that the analytical byte solver directly reflects runtime KV-cache memory requirements.

Batch scaling and prefill transient allocation. To investigate how KV-cache compression interacts with serving concurrency, we evaluate batch sizes from $B = 1$ to 32 at 8,192 context on 48 GB GPUs (Figure 13b). Across uncompressed, FSQ-512, and FSQ-128 configurations, peak memory during generation is dominated by the unchunked prefill transient rather than the persistent KV cache. In our reference float32 evaluation pipeline, materialising prefill attention score matrices incurs an empirical marginal cost of $\approx$ 11.1 GiB per sequence. Consequently, peak memory scales from 14.83 GB at $B = 1$ to 26.85 GB at $B = 2$ and 38.91 GB at $B = 3$. At $B = 4$, attempted memory allocation reaches $\approx$ 50.1 GB, precipitating out-of-memory (OOM) failures for uncompressed and FSQ models alike. Because the resident cache savings (28 MB vs 1.80 GB per sequence) are small relative to the 11.1 GiB prefill transient and the 48 GB device boundary, maximum viable batch size under unchunked float32 prefill is bounded by transient attention allocations rather than the compressed KV footprint. Deploying chunked prefill or flash-attention primitives

Table 6: **Stack-grid full numerical results at 8,192 tokens, Qwen3-0.6B-Base (RULER v2, 13 tasks, 50 items each; Appendix D.3):** every stacked cell that clears the pre-registered rule (Transformer-FSQ carries at least half the compression in log terms), against Transformer-FSQ alone, the uncompressed model, and PyramidKV alone matched to the same achieved total budget (`pyramidkv_matched`, own block below). Total ratio and B/token/layer are ACHIEVED (uncompressed 4096 B/token/layer divided by the achieved figure), not nominal. Score is the mean over each cell's complete seeds with an item-bootstrap half-width (Appendix C.4); **seeds** is the number of complete finetuning seeds a cell has (1 unless stated).

FSQ	PyramidKV	Total ratio	B/tok/layer	Score ± half-CI	Seeds
–	–	1.0×	4096	0.71 ± 0.02	1
8×	–	8.0×	512	0.58 ± 0.02	3
16×	–	16.0×	256	0.31 ± 0.02	3
32×	–	32.0×	128	0.22 ± 0.02	3
97.5×	–	97.5×	42	0.04 ± 0.01	3
8×	2×	16.0×	256	0.40 ± 0.02	1
8×	4×	32.0×	128	0.33 ± 0.02	1
8×	8×	64.0×	64	0.29 ± 0.02	1
16×	2×	32.0×	128	0.23 ± 0.02	1
16×	4×	64.0×	64	0.22 ± 0.02	1
16×	8×	128.1×	32	0.20 ± 0.02	1
16×	16×	256.3×	16	0.17 ± 0.02	1
32×	2×	64.0×	64	0.17 ± 0.02	1
32×	4×	128.0×	32	0.16 ± 0.02	1
32×	8×	256.2×	16	0.13 ± 0.02	1
32×	16×	512.6×	8	0.09 ± 0.02	1
32×	32×	1026.8×	4	0.01 ± 0.01	1
97.5×	2×	195.0×	21	0.05 ± 0.01	1
97.5×	4×	390.1×	11	0.03 ± 0.01	1
97.5×	8×	780.2×	5.3	0.03 ± 0.01	1
97.5×	16×	1563.4×	2.6	0.02 ± 0.01	1
97.5×	32×	3126.7×	1.3	0.00 ± 0.01	1
PyramidKV alone, matched to a stack cell's own achieved total budget					
–	matched	32.1×	127.75	0.10	1
–	matched	16.0×	255.75	0.30	1

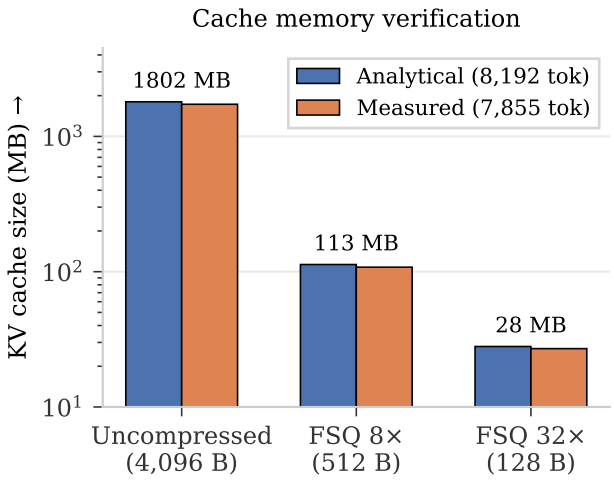

Figure 13: **Empirical memory verification:** Measured resident KV-cache memory against analytical capacity on Qwen3-0.6B-Base across compression ratios at 8,192 context (7,855 prompt tokens in practice), demonstrating agreement within 0.05%.

isolates the persistent cache advantage, allowing concurrent decoding to realise the token capacity enabled by discrete latents (Table 1).

E ABLATIONS

The variants, controls, settings and baselines behind our architectural choices are evaluated below using validation bits-per-byte (BPB), long-context retrieval, and decode throughput. Spreads report sample standard deviations over evaluated seeds.

E.1 VARIANTS IN THE MAIN FIGURES

Figure 14 extends Figure 6 with FSQ-small, MLA-FSQ, and pre-RoPE (NoPE/ALiBi equivalent in terms of computational complexity) decoding.

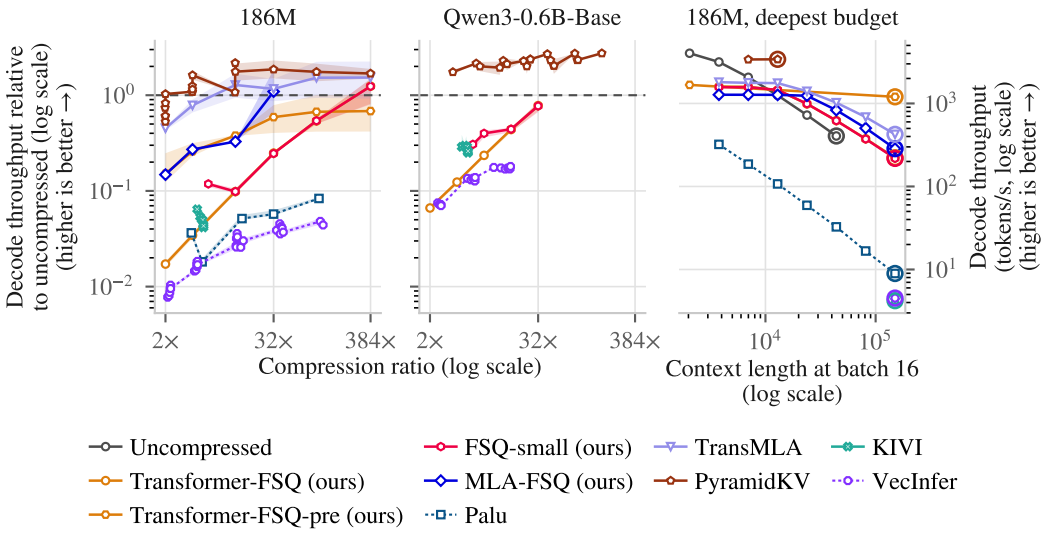

Figure 14: **Decode throughput across architectural variants:** Decode throughput relative to the uncompressed baseline across compression ratios, extending Figure 6 with FSQ-small, MLA-FSQ, and pre-RoPE (NoPE/ALiBi equivalent) decoding. Underlying throughput measurements across all batch and context configurations appear in Table 9. Pre-RoPE decoding avoids per-step key reconstruction, achieving speedups at extreme compression and long contexts ($1.03\times$ at batch 1 and $3.58\times$ at batch 4 for 150k context, §E), demonstrating that decode latency in standard transformers stems from RoPE rotary rotation rather than discrete compression.

E.2 MLA-FSQ

MLA-FSQ combines discrete FSQ codes for key-value content with Multi-Head Latent Attention's separate rotary key k^r, cached in half precision and shared across heads (Figure 16). On the 186M model (five seeds), MLA-FSQ consistently underperforms standard Transformer-FSQ across budgets (8.2 ± 1.3 vs 8.9 ± 0.6 at $2\times$; -0.2 ± 0.5 vs 2.6 ± 1.1 at $384\times$), while remaining at chance without finetuning. On Qwen3-0.6B-Base, shared query-key normalisation causes validation BPB to spike from 0.75 to 5.5, as large rotary norms (121.8 RMS vs 19.6 content) suppress discrete representations. The half-precision rotary key requires 65 B per token per layer, placing a hard architectural ceiling of $47\times$ on compression. Coupling discrete codes with unquantized rotary keys increases complexity while degrading accuracy and compression.

E.3 FSQ-PRE AND POSITIONAL ENCODINGS

FSQ-pre evaluates applying RoPE to the residual stream before the encoder (Figure 16), rotating activations as C/d_h blocks of width d_h so attention reads cached codes without reconstructing keys (§4.2). In this topology, query-key normalisation breaks relative-position preservation, inducing severe logit errors (0.60 ± 0.08 relative error vs 3×10^{-7} without normalisation). Across five seeds on

Table 7: **Cache width, precision, and quantisation during training.** RULER at 8,192 tokens on Qwen3-0.6B-Base (three seeds; 0.25 tokens/parameter). The uncompressed cache uses 4,096 B per token per layer and scores 0.724 ± 0.024. At each nominal budget, 4-bit FSQ permits four times the width of a 16-bit bottleneck; training with FSQ improves on applying it only after training. Bytes report actual cache storage, including metadata, rather than the nominal budget.
A learned bottleneck stores a compressed vector used to reconstruct keys and values.
FSQ stores 4-bit grid coordinates without scale metadata.
Trained FSQ applies rounding during training using a straight-through estimator; *post-hoc FSQ/KIVI* quantises only at evaluation after continuous training.
KIVI stores 4-bit blocks plus fp16 scale/offset metadata, exceeding the nominal budgets by 25%.
Palu stores 4-bit SVD codes with scales.
TransMLA is trained continuously and stores 4-bit content codes plus 16-bit rotary keys.
Width denotes each method's bottleneck dimension; Bytes denotes storage per token per layer. Extreme compression can be achieved by keeping a wide latent that learns to fit a discrete bottleneck.

Compression / quantisation	Width	Bytes	RULER (8k)
Nominal budget: 144 B/token/layer (28.4×)			
Learned bottleneck, 16-bit	72	144	0.011 ± 0.007
Learned bottleneck, post-hoc FSQ	288	144	0.172 ± 0.028
Learned bottleneck, trained FSQ (ours)	288	144	$\mathbf{0.232 \pm 0.026}$
Learned bottleneck, trained min-max 4-bit	224	140	0.189 ± 0.012
Learned bottleneck, post-hoc KIVI	288	180	0.208 ± 0.024
SVD compression (Palu), 4-bit	144	144	0.000 ± 0.000
Latent attention (TransMLA), 4-bit	64	144	0.001 ± 0.001
Nominal budget: 80 B/token/layer (51.2×)			
Learned bottleneck, 16-bit	40	80	0.001 ± 0.001
Learned bottleneck, post-hoc FSQ	160	80	0.144 ± 0.013
Learned bottleneck, trained FSQ (ours)	160	80	$\mathbf{0.182 \pm 0.018}$
Learned bottleneck, trained min-max 4-bit	128	80	0.153 ± 0.011
Learned bottleneck, post-hoc KIVI	160	100	0.169 ± 0.014
SVD compression (Palu), 4-bit	80	80	0.000 ± 0.000
Latent attention (TransMLA), 4-bit	32	76	0.000 ± 0.000

186M trained from scratch, FSQ-pre underperforms standard Transformer-FSQ at five of six compression ratios (e.g., 10.8 ± 0.7 vs 12.4 ± 0.2 at $12\times$, 5.2 ± 1.0 vs 8.3 ± 1.0 at $384\times$). FSQ-pre trades representational fidelity for rotation-free lookups, leading us to retain standard post-projection FSQ.

Decode timing measurements on the 186M model (Figure 14) confirm the operational speedup of eliminating per-step key reconstruction. At $150{,}000$ tokens of context, FSQ-pre achieves a $1.03\times$ speedup at batch 1 and $3.58\times$ at batch 4 under $384\times$ compression (Table 9). This confirms that decode latency in standard Transformer-FSQ stems from per-step RoPE key rotation rather than integer unpacking. Modern architectures without per-step rotary keys (e.g., NoPE or ALiBi) natively avoid reconstruction overhead and unlock immediate decode speedups under discrete compression.

E.4 Discreteness and Coordinate Allocations

Table 7 compares, at two equal cache budgets on Qwen3-0.6B-Base, Transformer-FSQ at its standard width (16 levels per dimension) against a trained half-precision latent of the same dimension, that latent quantised to 4 bits post-training, the same latent quantised using KIVI's block scheme, and Palu with a 4-bit latent on RULER at $8{,}192$ tokens: Transformer-FSQ scores above every continuous alternative at both sizes. Table 8 isolates levels per coordinate: at fixed cache memory, 16 levels per dimension yields superior accuracy compared to 4 or 256 levels.

Our default configurations standardise on 16 levels per coordinate ($b = 4$ bits per dimension), an operating point tuned for extreme compression ($12\times$ to $97.5\times$). At mild compression ($2\times$), dedicating cache bytes to narrower dimensions with finer levels (e.g., 256 levels) is a natural hypothesis. However, FSQ-small, which caps latent dimensions and allocates remaining bytes to 256 levels, is substantially *less* accurate than the uniform 16-level configuration (45.5 ± 3.6 vs 80.0 ± 2.6 on RULER at 512 B per token per layer on Qwen3-0.6B-Base). We therefore maintain uniform 16-level allocations across all compression ratios.

Table 8: **Levels per coordinate:** RULER score at $8{,}192$ tokens on Qwen3-0.6B-Base, finetuned on 0.25 tokens per parameter, FSQ with 4, 16 and 256 levels per coordinate at 144 and 80 B per token per layer. 16 levels score highest at both sizes. Reported as mean $\pm$ standard deviation over three finetuning seeds.

Cache	B/token/layer	Ratio	RULER (8k)
FSQ, 4 levels	144	28×	0.179 ± 0.014
FSQ, 16 levels	144	28×	0.232 ± 0.026
FSQ, 256 levels	144	28×	0.162 ± 0.011
FSQ, 4 levels	80	51×	0.118 ± 0.018
FSQ, 16 levels	80	51×	0.182 ± 0.018
FSQ, 256 levels	80	51×	0.036 ± 0.014

E.5 Straight-Through Estimation vs Continuous Relaxations

When optimizing discrete bottlenecks, alternatives to the straight-through estimator (STE) include continuous relaxations such as Gumbel-Softmax (Jang et al., 2017; Maddison et al., 2017) or codebook exponential moving average (EMA) updates (van den Oord et al., 2017; Scannell et al., 2025). In our architecture, FSQ projects activations onto a fixed, implicit Cartesian grid by rounding ($\mathrm{round}(z) \approx z$ on the backward pass). Because the grid is implicit and fixed, FSQ eliminates codebook collapse by construction without requiring auxiliary commitment losses or centroid dictionary synchronization. Continuous relaxations introduce high gradient variance across hundreds of discrete dimensions and require sensitive temperature schedules, whereas simple STE rounding trains stably without additional hyperparameters.

E.6 Joint End-to-End Training vs Layerwise Conversion

An alternative to joint end-to-end training of all discrete bottlenecks is sequential layerwise conversion, progressively freezing earlier layers and training each bottleneck in isolation. While layerwise training avoids backpropagating through multiple STE rounding steps simultaneously, joint end-to-end training allows subsequent layers to adapt directly to the discrete quantization errors of earlier representations. We adopt joint training as our universal, simpler training recipe.

E.7 Code utilisation and entropy coding

Information-theoretic analysis evaluates coordinate entropy and channel utilization across code dimensions. We measure empirical bit-rate across two million held-out tokens per configuration: coordinates carry 3.63–3.75 bits on the 186M model (91–94% of available 4-bit capacity) and 79–83% on Qwen3-0.6B-Base, with fewer than 0.3% of channels inactive even at $4{,}096$ dimensions. Because entropy coding is mathematically lossless, downstream task accuracy is invariant to whether codes are entropy-coded or stored packed. High coordinate utilization confirms that learned projections exploit the discrete capacity effectively without dead codebook clusters.

An ideal order-0 entropy coder would shrink the cache by 6–9% on 186M and 17–21% on Qwen3-0.6B-Base. We implemented a per-channel static Huffman coder over blocks of 128–512 tokens; net of per-block offset tables and byte alignment ($\approx 3.8\%$ overhead), savings ranged from 0.8–4.4% on 186M up to 12.9% at extreme compression on Qwen3-0.6B-Base. However, variable-length decoding increased decode latency by $16.6\times$–$21.2\times$ over our fused kernel and made memory footprint input-dependent. Lossless entropy coding is rejected due to excessive decode latency penalties and non-deterministic cache sizing.

E.8 Nonlinear encoders

We evaluated replacing the linear encoder with a rank-128 residual MLP ($z = W_{\text{down}}x + W_2\,\mathrm{SiLU}(W_1 x)$, with W_2 initialized to zero) on Qwen3-0.6B-Base (three seeds). The nonlinear encoder degraded validation loss across all cache budgets, increasing validation bits per byte from 1.000 ± 0.001 to 1.015 ± 0.002 at 128 B per token per layer ($32\times$). A residual encoder with squared ReLU similarly degraded validation loss after finetuning (3.466 vs 3.337 BPB). We therefore retain the linear encoder, which also eliminates non-linear projection overhead during prefill.

E.9 PER-HEAD CODES

Grouping codes by key-value head allows each head to decode its own slice independently. We evaluated block-diagonal key and value decoders where each of Qwen3-0.6B-Base's 8 heads reads $1/8$ of the code (64, 32, and 10 dimensions at $16\times$, $32\times$, and $97.5\times$). Per-head partitioning substantially degrades representation quality across all cache sizes, increasing validation bits per byte from 0.797 ± 0.001 to 0.911 ± 0.001 at $16\times$, 0.879 ± 0.002 to 1.109 ± 0.003 at $32\times$, and 1.173 ± 0.003 to 1.718 ± 0.008 at $97.5\times$. Sharing code representations across heads pools cross-head redundancy and achieves significantly superior compression fidelity, so we retain shared codes.

E.10 RECENT-TOKEN WINDOW ON TRANSFORMER-FSQ

Adding a 128-token full-precision window and four sinks to Transformer-FSQ shifts achieved memory: the nominal $8\times$ target (512 B per token per layer) expands to 578 B ($7.1\times$ achieved) on Qwen3-0.6B-Base, and $32\times$ expands to $28.4\times$ on 8B. When evaluated at matched achieved ratios on RULER (Tables 13 and 15), windowed FSQ scores 0.58 at $7.1\times$ vs plain FSQ's 0.57 at $8\times$ on 0.6B, and 0.54 vs 0.50 on 8B. Accounting for physical byte consumption shows that sliding windows do not alter the Pareto frontier at 8k context.

Priced at equal byte budgets, windowed variants lie on the same score-against-compression curve as plain Transformer-FSQ across ratios from $1.8\times$ to $372.4\times$ (Tables 13–15). The window neither reliably improves nor harms RULER score once its extra bytes are counted in the achieved ratio at $8{,}192$ context. Standalone Transformer-FSQ operates effectively without auxiliary window buffers at evaluated sequence lengths.

E.11 FINETUNING SETTINGS

All recovery-trained methods (Transformer-FSQ, Palu, TransMLA) are finetuned using a standard learning rate of 10^{-5} at 0.25 tokens per parameter without method-specific tuning. Learning-rate sensitivity sweeps show that 10^{-3} diverges through discrete bottlenecks and 10^{-4} degrades stability, while 10^{-5} yields robust convergence across scales; sequence length changes from $1{,}024$ to $2{,}048$ produce minimal variation. On the 186M model, smaller micro-batches ($16{,}384$ tokens vs $524{,}288$) strengthen gradient signals through the STE, improving scores from 5.9 to 8.4 at $12\times$ and 0.5 to 2.6 at $384\times$. A learning rate of 10^{-5} paired with small micro-batches provides a universally robust recipe for post-training conversion.

F ARCHITECTURE DIAGRAMS

F.1 THE FSQ BOTTLENECK

A linear encoder $E \in \mathbb{R}^{G\times C}$ maps the residual stream, not the key or value, onto the grid:

$$u_t \;=\; \tanh\big(Ex_t + \operatorname{atanh}(\delta/h)\big)\, h \;-\; \delta, \qquad z_t \;=\; u_t + \operatorname{sg}\big(\lfloor u_t \rceil - u_t\big), \tag{8}$$

where $\lfloor\cdot\rceil$ denotes round-to-nearest with round-half-to-even tie breaking (`torch.round`), the stop-gradient sg implements the straight-through estimator (Bengio et al., 2013), and h and δ place the L levels on the integers (Appendix C.3). Concretely, we set bounding scale $\varepsilon = 10^{-3}$, half-width $h = (L-1)(1+\varepsilon)/2 = 7.5075$ for $L = 16$, and center shift $\delta = 0.5$ for even L ($\delta = 0$ for odd L). Packing stores the unsigned integer $\lfloor L/2 \rfloor$ plus the rounded signed level (0 to $L-1$ for $L = 16$, i.e. $-8, \ldots, 7 \mapsto 0, \ldots, 15$); the decoder subtracts the same offset before applying W_k/W_v. With no codebook there is no commitment loss, codebook update or dead-code replacement. The cache bit-packs each code at $b = \lceil \log_2 L \rceil$ bits per coordinate.

Figure 15 contrasts one decoding step of an unmodified transformer with one step of Transformer-FSQ, and Figure 16 shows the two variants of Appendix E that change how the cache is read: FSQ-pre and MLA-FSQ. The shaded region of each diagram marks the work that grows with context length.

G PER-TASK RESULTS IN FULL

The tables below report the complete per-task breakdowns for RULER and LongBench across models, followed by full decode-throughput benchmark measurements.

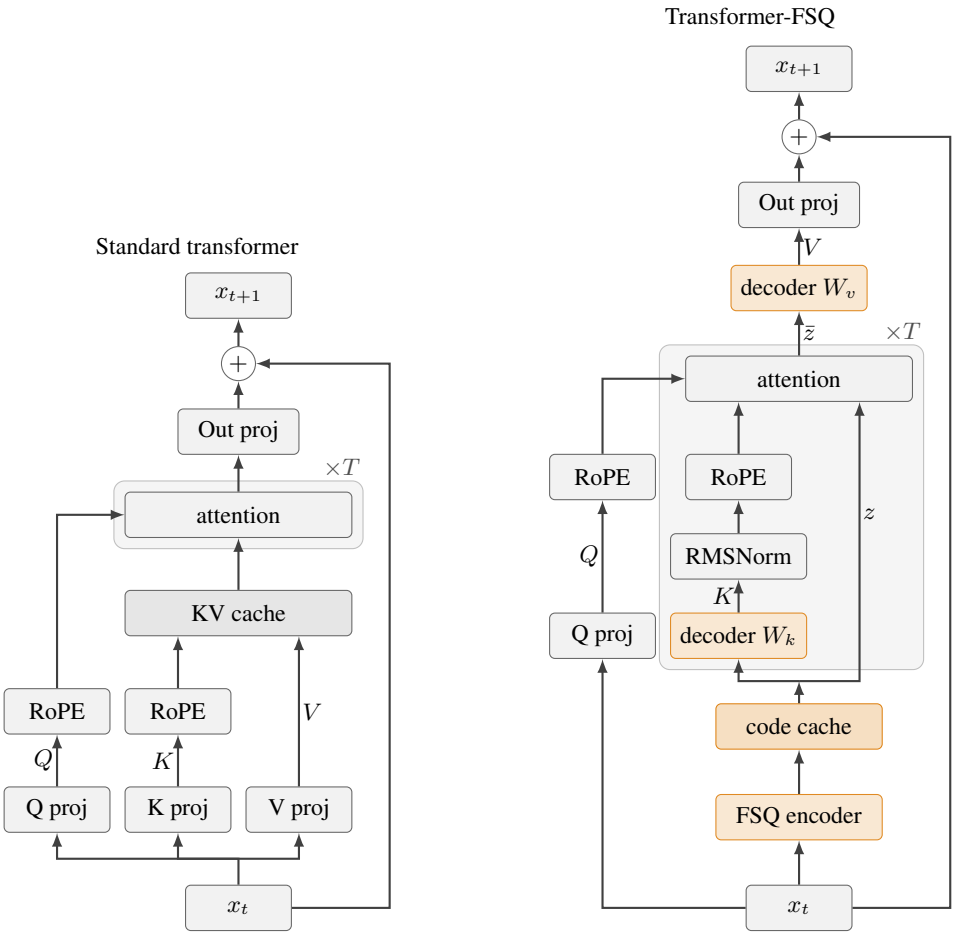

Figure 15: **Architectural decoding dataflow comparison:** One decoding step of an unmodified transformer (left) and Transformer-FSQ (right). Transformer-FSQ caches packed integer codes in place of K and V. At each step, decoder W_k reconstructs keys at cached positions, applying optional RMSNorm and RoPE rotation inside the context loop; query heads pool codes into attention-weighted vectors $\bar{z} = \sum_t a_t z_t$, allowing decoder W_v to reconstruct values once outside the loop. Shaded plates marked $\times T$ indicate operations repeated across context length T. Method taxonomy and compression limits appear in Table 1.

G.1 Evaluation Benchmarks and Notation

Benchmarks. We evaluate long-context retrieval and reasoning across two complementary suites:

- **RULER (Hsieh et al., 2024):** 13 synthetic retrieval tasks at 8,192 tokens (50 items each): single-key retrieval (`niah_single_1`, `2`, `3`), multi-key retrieval (`niah_multikey_1`, `2`, `3`), multi-value retrieval (`niah_multivalue`), multi-query retrieval (`niah_multiquery`), variable tracking (`vt`), common/frequent words extraction (`cwe`, `fwe`), and QA (`qa_1`, `qa_2`).
- **LongBench (Bai et al., 2024):** Real-world multi-page document comprehension: `narrativeqa` (narrative question answering, F1), `gov_report` (government report summarisation, ROUGE-L), and `trec` (few-shot in-context question classification, accuracy).

Table notation. In RULER tables (Tables 13–15), rows report the achieved compression ratio priced by the analytical byte solver, sample mean, and standard deviation across seeds. Finetuned methods (Transformer-FSQ, Palu, TransMLA) use 0.25 finetuning tokens per parameter; KIVI, PyramidKV, and VecInfer are training-free. In LongBench tables (Tables 16–18), columns report achieved bytes per token per layer (**B/tok**), achieved compression ratio (**Ratio**), evaluated seeds,

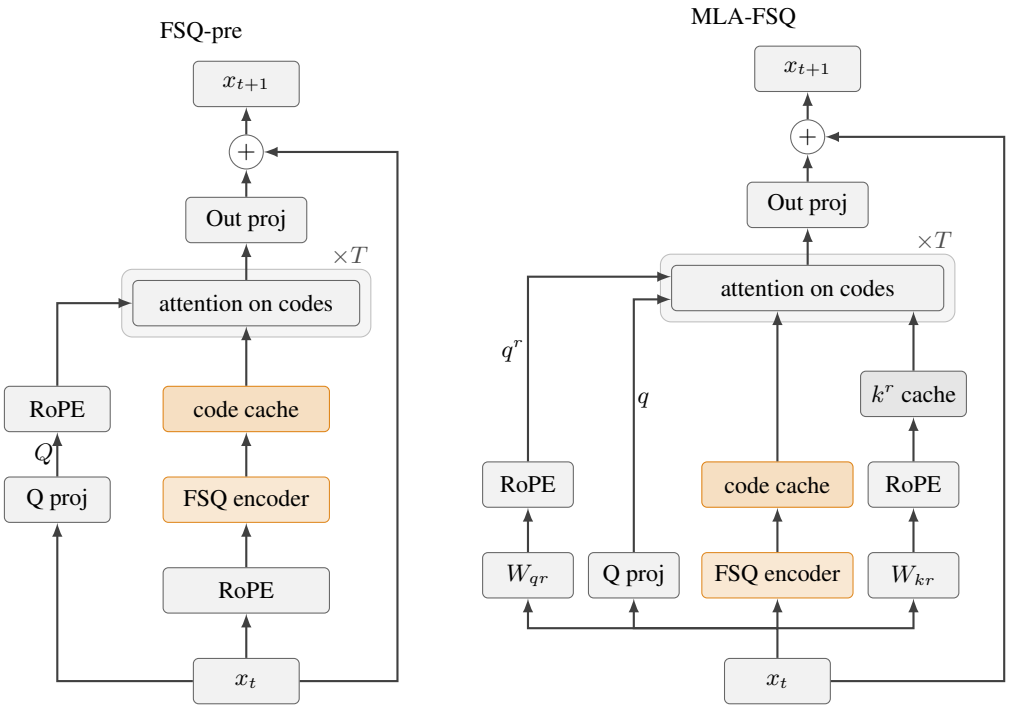

Figure 16: **Alternative discrete cache decoding topologies:** Decoding with FSQ-pre (left) and MLA-FSQ (right). FSQ-pre applies RoPE before the encoder so stored codes carry position directly, enabling rotation-free attention without per-step key reconstruction. MLA-FSQ caches a separate half-precision rotary key k^r alongside discrete content codes, adding query dot products to attention logits. Shaded plates ($\times T$) indicate operations scaling with context length T. Both variants eliminate key reconstruction during attention, with trade-offs detailed in Section E.

scored examples (n), and task scores. [§] marks dynamic-cache methods (KIVI, PyramidKV, VecInfer) whose cache footprint depends on context length.

G.2 Decode Throughput Breakdown

Table 9 and its continuation tables, and the Qwen3-0.6B-Base and Qwen3-1.7B-Base tables that follow, share one sweep of batch sizes $B \in \{1, 4, 16, 64\}$ on an NVIDIA L40S GPU: seven log-spaced context lengths from 3,782 to 150,000 tokens for the 186M and Qwen3-0.6B-Base files, where the uncompressed model completes 15 cells, and eight from 2,048 to 150,000 for Qwen3-1.7B-Base, where it completes 21.

Metrics and Columns. In each table:

- **Method:** Identifies the compression method and architectural variant.
- **Target:** The requested per-token cache memory budget in bytes per layer.
- **B/tok** and **Ratio:** The achieved bytes per token per layer, priced by the byte solver, and the resulting compression ratio relative to the uncompressed model ($c_{\text{full}} = 3072$ B on 186M). For methods whose cache size depends on context length (such as full-precision sliding windows or dynamic token eviction buffers), values are reported as the achieved range.
- **Settings:** The count of distinct batch-and-context configurations evaluated where both the compressed configuration and the uncompressed reference completed without out-of-memory errors.
- **Median:** The median decode throughput relative to the uncompressed baseline across all completed settings. This column defines the solid curves plotted in Figure 6 (left and middle panels) and Figure 14.
- **Range (n slower):** The full minimum–maximum range of relative throughput observed across the benchmark grid, followed in parentheses by the number of configurations that

decode more slowly than the uncompressed baseline. Figures 6 and 14 plot a narrower band, the interquartile range (25th to 75th percentile) across the same grid: a repeated timing of one setting moves its rate by under 0.2% of the median, while the same budget at a different batch and context moves it by a factor, so the ribbon shows spread across the grid, not measurement noise, and this table's wider min–max range is not the ribbon's own boundary.

Benchmarking Protocol and Load Gating. Timed benchmark rows are gated on a one-minute host load of at most 2.0 to eliminate multi-tenant scheduling noise and OS jitter; ungated runs (median load 2.90, maximum 4.32) are distinguished in Table 9. Each cell discards 2 warm-up repetitions for kernel autotuning and allocator expansion before timing 5 repetitions bounded by explicit CUDA synchronization, reporting their median (Appendix D.4). Rigorous host gating and timing synchronization ensure reproducible latency measurements across all methods.

Baseline Kernel Dispatches and Custom Implementations. Baseline measurements characterize standard PyTorch implementations rather than proprietary vendor kernels: Palu and VecInfer rely on reference PyTorch dispatches (†), as VecInfer's proprietary kernel was not integrated and Palu's requires 16-bit latents that our configurations do not trigger. KIVI dispatches its released CUDA kernel for values only, while keys execute unfused. In contrast, Transformer-FSQ dispatches custom fused CUDA kernels for integer unpacking and key reconstruction, deferring value projection to a single matrix multiply on attention-weighted codes.

Decode Latency and Key-Reconstruction Cost. Autoregressive decoding with Transformer-FSQ reconstructs key states from cached codes at each generation step, while values are decoded once per query head from attention-weighted codes (§4.2). Per layer and step, key reconstruction costs $O(TGn_{kv}d_h)$ multiply-adds, whereas value decoding costs $O(TGn_q + Gn_q d_h)$. At moderate compression ($2.0\times$–$14.7\times$), this reconstruction arithmetic dominates memory savings, resulting in slower decoding than uncompressed baselines across 14 evaluated settings. Speedups emerge at extreme compression ($384\times$), where drastic cache footprint reductions overcome compute overhead on memory-bound long contexts.

Table 9: **Decode throughput relative to uncompressed baseline (186M, part 1):** Configurations below $16\times$ compression ($2.0\times$ to $14.7\times$). Per-step key reconstruction dequantization causes throughput overhead across intermediate ratios. Dynamic window methods report achieved ranges; complete benchmark methodology matches Section G.2.

Method	Target	B/tok	Ratio	Settings	Median	Range (n slower)
186M — $2.0\times$**–**$14.7\times$						
Transformer-FSQ (ours)	1536	1536	2.0×	14	0.02	0.02–0.02 (14/14)
Transformer-FSQ (ours)	768	768	4.0×	14	0.03	0.03–0.04 (14/14)
Transformer-FSQ (ours)	256	256	12.0×	14	0.10	0.09–0.12 (14/14)
FSQ-small (ours)	1536, 768	512	6.0×	14	0.12	0.11–0.13 (14/14)
MLA-FSQ (ours)	1536	1536	2.0×	14	0.15	0.14–0.16 (14/14)
MLA-FSQ (ours)	768	768	4.0×	14	0.27	0.26–0.28 (14/14)
MLA-FSQ (ours)	256	256	12.0×	14	0.33	0.31–0.38 (14/14)
TransMLA	1536	1536	2.0×	14	0.45	0.39–0.47 (14/14)
TransMLA	768	768	4.0×	14	0.78	0.54–0.97 (14/14)
TransMLA	256	256	12.0×	14	1.28	0.59–2.09 (4/14)
Palu†	1536	792	3.9×	14	0.04	0.03–0.05 (14/14)
Palu†	768	600	5.1×	14	0.02	0.02–0.02 (14/14)
Palu†	256	216	14.2×	14	0.05	0.05–0.07 (14/14)
KIVI	1536, 768	579–681	4.5–5.3×	14	0.05	0.04–0.07 (14/14)
PyramidKV	1536	1536–1551	2.0×	7	0.61	0.51–1.14 (5/7)
PyramidKV	768	768–775	4.0×	7	1.21	1.08–1.96 (0/7)
PyramidKV	256	256–258	11.9–12.0×	7	1.62	1.07–2.73 (0/7)
VecInfer†	1536	1347–1451	2.1–2.3×	12	0.01	0.01–0.01 (12/12)
VecInfer†	768	675–730	4.2–4.6×	14	0.02	0.01–0.02 (14/14)
VecInfer†	256	209–251	12.2–14.7×	14	0.03	0.02–0.04 (14/14)

Table 10: **Decode throughput relative to uncompressed baseline (186M, part 2):** Configurations at $16\times$ compression and above ($32.0\times$ to $384.0\times$). Throughput parity and speedups ($1.24\times$ median at $384\times$) emerge as cache footprint reductions overcome dequantization compute on long contexts. Formats match part 1.

Method	Target	B/tok	Ratio	Settings	Median	Range (n slower)
186M — $32.0\times$**–**$384.0\times$						
Transformer-FSQ (ours)	96	96	$32.0\times$	14	0.25	0.24–0.28 (14/14)
Transformer-FSQ (ours)	32	32	$96.0\times$	14	0.54	0.47–0.65 (14/14)
Transformer-FSQ (ours)	8	8	$384.0\times$	14	1.24	0.51–1.83 (5/14)
MLA-FSQ (ours)	96	96	$32.0\times$	14	1.09	0.40–2.21 (6/14)
TransMLA	96	96	$32.0\times$	14	1.16	0.58–1.84 (5/14)
TransMLA	32	32	$96.0\times$	14	1.52	0.58–3.12 (4/14)
TransMLA	8	8	$384.0\times$	14	1.53	0.57–3.16 (4/14)
Palu†	96	96	$32.0\times$	14	0.06	0.05–0.08 (14/14)
Palu†	32	30	$102.4\times$	14	0.08	0.07–0.10 (14/14)
PyramidKV	96	96	$32.0\times$	6	1.86	1.29–2.73 (0/6)
PyramidKV	32	32	$96.0\times$	4	1.75	1.28–2.35 (0/4)
PyramidKV	8	8	$384.0\times$	2	1.69	1.29–2.08 (0/2)
VecInfer†	96	75–89	34.5–$41.0\times$	12	0.04	0.03–0.05 (12/12)
VecInfer†	32	27–29	105.9–$113.8\times$	4	0.04	0.04–0.06 (4/4)

Table 11: **Decode throughput relative to uncompressed baseline (Qwen3-0.6B-Base):** Measured on an NVIDIA L40S across batch and context grids. Relative throughput rises from $0.07\times$ at $2\times$ to $0.78\times$ at $32\times$, tracking 186M scaling trends. Window methods report achieved ranges; protocol matches Section G.2.

Method	Target	B/tok	Ratio	Settings	Median	Range (n slower)
Qwen3-0.6B-Base (L40S)						
Transformer-FSQ (ours)	2048	2048	2.0×	13	0.07	0.06–0.08 (13/13)
Transformer-FSQ (ours)	1024	1024	4.0×	14	0.12	0.11–0.13 (14/14)
Transformer-FSQ (ours)	512	512	8.0×	15	0.24	0.20–0.26 (15/15)
Transformer-FSQ (ours)	256	256	16.0×	15	0.44	0.34–0.48 (15/15)
Transformer-FSQ (ours)	128	128	32.0×	15	0.78	0.49–0.91 (15/15)
FSQ-small (ours)	2048, 1024	682	6.0×	14	0.31	0.26–0.33 (14/14)
FSQ-small (ours)	512	512	8.0×	15	0.40	0.31–0.45 (15/15)
KIVI	2048, 1024	772–908	4.5–5.3×	14	0.27	0.24–0.42 (14/14)
PyramidKV	2048	101–1139	3.6–40.6×	7	2.15	1.32–2.69 (0/7)
PyramidKV	1024	50–569	7.2–81.9×	7	2.32	1.34–2.73 (0/7)
PyramidKV	515	25–286	14.3–163.8×	7	2.37	1.34–2.75 (0/7)
PyramidKV	512	25–284	14.4–163.8×	7	2.37	1.33–2.75 (0/7)
VecInfer†	2048	1543–1679	2.4–2.7×	9	0.07	0.07–0.08 (9/9)
VecInfer†	1024	644–783	5.2–6.4×	10	0.13	0.13–0.14 (10/10)
VecInfer†	560	260–399	10.3–15.8×	14	0.17	0.16–0.18 (14/14)

Table 12: **Decode throughput relative to uncompressed baseline (Qwen3-1.7B-Base):** Measured on an NVIDIA L40S. Throughput speedups reach $2.02\times$ median at $409.6\times$ extreme compression (10 B/token/layer), overcoming dequantization compute on long contexts. Methodology matches Section G.2.

Method	Target	B/tok	Ratio	Settings	Median	Range (n slower)
Qwen3-1.7B-Base (L40S)						
Transformer-FSQ (ours)	2048	2048	2.0×	17	0.07	0.05–0.17 (17/17)
Transformer-FSQ (ours)	1024	1024	4.0×	19	0.13	0.12–0.26 (19/19)
Transformer-FSQ (ours)	341	341	12.0×	19	0.35	0.27–0.41 (19/19)
Transformer-FSQ (ours)	128	128	32.0×	17	0.73	0.33–0.99 (17/17)
Transformer-FSQ (ours)	42	42	97.5×	20	0.69	0.34–0.83 (20/20)
Transformer-FSQ (ours)	10	10	409.6×	18	2.02	0.35–5.03 (5/18)
FSQ-small (ours)	2048, 1024	682	6.0×	19	0.30	0.24–0.33 (19/19)
KIVI	2048, 1024	772–1024	4.0–5.3×	19	0.30	0.25–0.46 (19/19)
PyramidKV	2048	101–2047	2.0–40.6×	12	1.49	0.86–2.48 (4/12)
PyramidKV	1024	50–1023	4.0–81.9×	12	1.77	0.86–2.51 (3/12)
PyramidKV	341	17–340	12.0–240.9×	12	1.89	0.86–2.61 (3/12)
VecInfer†	2048	1543–1800	2.3–2.7×	13	0.08	0.07–0.27 (13/13)
VecInfer†	1024	644–904	4.5–6.4×	15	0.14	0.13–0.27 (15/15)
VecInfer†	341	68–328	12.5–60.2×	16	0.24	0.23–0.29 (16/16)

G.3 PER-TASK LONG-CONTEXT RETRIEVAL: RULER

Tables 13–15 give the per-task RULER breakdown behind Figure 2, Figure 8, and Appendix D.2: 13 official tasks, 50 items each, at 8,192 tokens, the RULER protocol every RULER number in the main text uses.

Table 13: Per-task RULER results on Qwen3-0.6B-Base (13 official tasks, 50 items each, at 8,192 tokens, float32). Finetuned methods use 0.25 tokens per parameter; KIVI, PyramidKV, and VecInfer are training-free. Rows report achieved compression ratios priced by the analytical byte solver, with scores averaged across complete seeds. Cell backgrounds reflect task scores via a colorblind-friendly pastel gradient.

Method	seeds	niah_single_1	niah_single_2	niah_single_3	niah_multikey_1	niah_multikey_2	niah_multikey_3	niah_multivalue	niah_multiquery	vt	cwe	fwe	qa_1	qa_2	Mean
Uncompressed	1	1.00	0.98	0.98	0.92	0.82	0.38	0.77	0.87	0.80	0.29	0.54	0.58	0.34	0.71
Transformer-FSQ (2.0×)	3	1.00	0.88	0.96	0.64	0.40	0.04	0.67	0.54	0.72	0.32	0.42	0.48	0.23	0.56
Transformer-FSQ (4.0×)	3	1.00	0.89	0.96	0.69	0.49	0.04	0.72	0.59	0.68	0.32	0.42	0.47	0.25	0.58
Transformer-FSQ (8.0×)	3	1.00	0.91	0.99	0.68	0.39	0.01	0.70	0.56	0.61	0.31	0.44	0.50	0.26	0.57
Transformer-FSQ (12.0×)	3	0.97	0.87	0.75	0.43	0.09	0.01	0.34	0.42	0.18	0.29	0.23	0.37	0.17	0.39
Transformer-FSQ (16.0×)	3	0.97	0.65	0.58	0.28	0.03	0.00	0.26	0.25	0.11	0.26	0.16	0.31	0.14	0.31
Transformer-FSQ (32.0×)	3	0.95	0.49	0.38	0.25	0.00	0.00	0.22	0.22	0.00	0.12	0.02	0.18	0.09	0.22
Transformer-FSQ (97.5×)	3	0.17	0.12	0.00	0.08	0.00	0.00	0.05	0.03	0.00	0.00	0.00	0.04	0.01	0.04
Transformer-FSQ (409.6×)	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00
Palu (3.9×)	3	1.00	1.00	1.00	0.90	0.81	0.33	0.80	0.78	0.80	0.34	0.49	0.65	0.31	0.71
Palu (5.2×)	3	1.00	0.99	1.00	0.87	0.73	0.08	0.79	0.76	0.71	0.36	0.45	0.61	0.29	0.66
Palu (10.2×)	3	1.00	0.87	0.83	0.59	0.07	0.00	0.38	0.31	0.00	0.05	0.18	0.35	0.14	0.37
Palu (15.1×)	3	0.54	0.17	0.01	0.19	0.00	0.00	0.12	0.11	0.00	0.01	0.04	0.19	0.11	0.12
Palu (19.7×)	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.06	0.04	0.01
Palu (102.4×)	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Palu (512.0×)	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TransMLA (2.0×)	3	0.97	0.87	0.78	0.37	0.00	0.00	0.39	0.32	0.36	0.15	0.30	0.34	0.16	0.38
TransMLA (4.0×)	3	1.00	0.71	0.52	0.32	0.00	0.00	0.36	0.26	0.26	0.15	0.21	0.29	0.14	0.32
TransMLA (8.0×)	3	0.95	0.38	0.27	0.17	0.00	0.00	0.18	0.12	0.00	0.08	0.01	0.15	0.15	0.19
TransMLA (12.0×)	3	0.90	0.14	0.04	0.09	0.00	0.00	0.09	0.02	0.00	0.02	0.05	0.11	0.08	0.12
TransMLA (16.0×)	3	0.61	0.07	0.00	0.07	0.00	0.00	0.06	0.01	0.00	0.01	0.02	0.10	0.03	0.07
TransMLA (32.0×)	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TransMLA (97.5×)	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TransMLA (409.6×)	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
KIVI (4.9×)	1	0.68	0.66	0.12	0.36	0.00	0.00	0.25	0.34	0.33	0.23	0.33	0.38	0.24	0.30
PyramidKV (4.0×)	1	1.00	0.72	0.00	0.60	0.08	0.00	0.32	0.31	0.78	0.27	0.17	0.70	0.24	0.40
PyramidKV (12.0×)	1	1.00	0.32	0.00	0.10	0.00	0.00	0.04	0.01	0.77	0.15	0.18	0.62	0.16	0.26
PyramidKV (32.0×)	1	0.42	0.04	0.00	0.02	0.00	0.00	0.00	0.00	0.28	0.02	0.01	0.52	0.12	0.11
VecInfer (4.3×)	1	0.98	0.98	1.00	0.88	0.64	0.20	0.78	0.85	0.75	0.37	0.56	0.52	0.32	0.68
VecInfer (12.7×)	1	0.02	0.04	0.04	0.12	0.02	0.00	0.01	0.04	0.00	0.00	0.05	0.08	0.10	0.04
VecInfer (35.9×)	1	0.00	0.04	0.00	0.04	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.04	0.08	0.02

Notes: **seeds** states how many complete finetuning seeds the score above averages; where **seeds** > 1, the score is the mean of that many complete seeds' own per-task means. $\pm$ is omitted for compactness. Ratio: uncompressed bytes per token per layer over the cache actually held (§4.4), not the nominal budget. Left out, the arm's solver reaches no configuration at this budget on this checkpoint: PyramidKV at 42 and 10 B and VecInfer at 42 and 10 B.

Table 14: Per-task RULER results on Qwen3-1.7B-Base (13 official tasks, 50 items each, at 8,192 tokens, float32). Finetuned methods use 0.25 tokens per parameter; KIVI, PyramidKV, and VecInfer are training-free. Rows report achieved compression ratios priced by the analytical byte solver, with scores averaged across complete seeds. Cell backgrounds reflect task scores via a colorblind-friendly pastel gradient.

Method	seeds	niah_single_1	niah_single_2	niah_single_3	niah_multikey_1	niah_multikey_2	niah_multikey_3	niah_multivalue	niah_multiquery	vt	cwe	fwe	qa_1	qa_2	Mean
Uncompressed	1	1.00	1.00	1.00	0.98	0.90	0.66	0.98	0.98	0.58	0.36	0.68	0.60	0.30	0.77
Transformer-FSQ (2.0×)	3	1.00	1.00	0.99	0.83	0.43	0.04	0.83	0.89	0.68	0.36	0.46	0.61	0.31	0.65
Transformer-FSQ (4.0×)	3	1.00	0.99	1.00	0.84	0.43	0.05	0.85	0.87	0.75	0.33	0.42	0.63	0.33	0.65
Transformer-FSQ (8.0×)	3	0.99	0.88	0.97	0.71	0.11	0.01	0.70	0.66	0.50	0.40	0.23	0.39	0.23	0.52
Transformer-FSQ (12.0×)	3	1.00	0.72	0.95	0.53	0.05	0.00	0.59	0.48	0.51	0.34	0.17	0.32	0.19	0.45
Transformer-FSQ (16.0×)	3	1.00	0.65	0.87	0.39	0.06	0.00	0.45	0.36	0.24	0.26	0.16	0.26	0.19	0.38
Transformer-FSQ (32.0×)	3	0.99	0.58	0.55	0.26	0.00	0.00	0.23	0.24	0.00	0.16	0.07	0.21	0.11	0.26
Transformer-FSQ (97.5×)	3	0.63	0.15	0.00	0.11	0.00	0.00	0.07	0.06	0.00	0.00	0.00	0.07	0.04	0.09
Transformer-FSQ (409.6×)	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00
Palu (3.9×)	3	1.00	1.00	1.00	0.96	0.87	0.13	0.96	0.97	0.83	0.41	0.67	0.71	0.34	0.76
Palu (5.2×)	3	1.00	0.99	0.98	0.96	0.78	0.05	0.92	0.91	0.75	0.42	0.70	0.63	0.27	0.72
Palu (10.2×)	3	0.98	0.29	0.23	0.29	0.01	0.00	0.32	0.21	0.55	0.30	0.33	0.37	0.20	0.31
Palu (19.7×)	3	0.05	0.18	0.02	0.09	0.00	0.00	0.03	0.02	0.00	0.00	0.02	0.11	0.07	0.05
Palu (102.4×)	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Palu (512.0×)	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TransMLA (2.0×)	3	0.96	0.77	0.74	0.45	0.00	0.00	0.36	0.32	0.03	0.19	0.01	0.33	0.20	0.33
TransMLA (4.0×)	3	0.97	0.70	0.51	0.21	0.00	0.00	0.24	0.23	0.33	0.25	0.22	0.29	0.13	0.31
TransMLA (8.0×)	3	0.84	0.43	0.13	0.10	0.00	0.00	0.20	0.09	0.00	0.10	0.06	0.14	0.09	0.17
TransMLA (16.0×)	3	0.05	0.05	0.00	0.03	0.00	0.00	0.03	0.02	0.00	0.00	0.08	0.07	0.01	0.03
TransMLA (32.0×)	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TransMLA (97.5×)	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TransMLA (409.6×)	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
KIVI (4.9×)	1	0.92	0.86	0.78	0.56	0.12	0.00	0.60	0.52	0.47	0.45	0.62	0.44	0.30	0.51
PyramidKV (4.0×)	1	1.00	0.94	0.02	0.68	0.24	0.00	0.58	0.56	0.53	0.32	0.46	0.58	0.30	0.48
PyramidKV (12.0×)	1	1.00	0.42	0.00	0.24	0.00	0.00	0.10	0.01	0.50	0.23	0.40	0.54	0.28	0.29
PyramidKV (32.0×)	1	0.60	0.16	0.00	0.02	0.00	0.00	0.00	0.00	0.38	0.03	0.06	0.44	0.20	0.15
VecInfer (4.3×)	1	1.00	1.00	1.00	0.96	0.70	0.62	0.92	0.94	0.50	0.39	0.69	0.58	0.28	0.74
VecInfer (12.7×)	1	0.02	0.04	0.04	0.12	0.02	0.00	0.01	0.04	0.00	0.01	0.34	0.16	0.08	0.07
VecInfer (35.9×)	1	0.02	0.02	0.00	0.04	0.00	0.00	0.00	0.01	0.00	0.01	0.01	0.04	0.08	0.02

Notes: seeds states how many complete finetuning seeds the score above averages; where **seeds** > 1, the score is the mean of that many complete seeds' own per-task means. $\pm$ is omitted for compactness. Ratio: uncompressed bytes per token per layer over the cache actually held (§4.4), not the nominal budget. Left out, the arm's solver reaches no configuration at this budget on this checkpoint: PyramidKV at 42 and 10 B and VecInfer at 42 and 10 B.

Table 15: Per-task RULER results on Qwen3-8B-Base (13 official tasks, 50 items each, at 8,192 tokens, float32). Finetuned methods use 0.25 tokens per parameter; KIVI, PyramidKV, and VecInfer are training-free. Rows report achieved compression ratios priced by the analytical byte solver, with scores averaged across complete seeds. Cell backgrounds reflect task scores via a colorblind-friendly pastel gradient.

Method	seeds	niah_single_1	niah_single_2	niah_single_3	niah_multikey_1	niah_multikey_2	niah_multikey_3	niah_multivalue	niah_multiquery	vt	cwe	fwe	qa_1	qa_2	Mean
Uncompressed	1	1.00	0.98	1.00	1.00	0.96	1.00	0.99	0.99	0.85	0.93	0.90	0.92	0.42	0.92
Transformer-FSQ (8.0×)	1	1.00	1.00	0.96	0.82	0.20	0.02	0.84	0.82	0.60	0.68	0.83	0.70	0.26	0.67
Transformer-FSQ (12.0×)	1	1.00	1.00	1.00	0.68	0.12	0.02	0.83	0.71	0.58	0.60	0.73	0.56	0.24	0.62
Transformer-FSQ (32.0×)	1	1.00	0.96	0.86	0.56	0.02	0.00	0.56	0.47	0.55	0.52	0.45	0.34	0.18	0.50
Transformer-FSQ (97.5×)	1	0.98	0.74	0.34	0.18	0.00	0.00	0.17	0.14	0.00	0.07	0.09	0.22	0.12	0.23
Transformer-FSQ (409.6×)	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.00
Palu (10.2×)	1	0.96	0.60	0.14	0.22	0.00	0.00	0.21	0.15	0.23	0.44	0.36	0.22	0.10	0.28
Palu (15.1×)	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.02	0.00
Palu (36.6×)	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.02	0.01
Palu (102.4×)	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00
Palu (512.0×)	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00
TransMLA (8.0×)	1	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.11	0.11	0.04	0.00	0.02
TransMLA (12.0×)	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.02	0.00	0.00
TransMLA (32.0×)	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TransMLA (97.5×)	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00
TransMLA (409.6×)	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
KIVI (4.9×)	1	1.00	1.00	0.96	0.90	0.74	0.70	0.95	0.93	0.72	0.90	0.86	0.76	0.38	0.83
PyramidKV (4.0×)	1	1.00	1.00	0.38	1.00	0.86	0.00	0.99	0.96	0.81	0.50	0.73	0.92	0.42	0.74
PyramidKV (12.0×)	1	1.00	0.98	0.00	1.00	0.48	0.00	0.54	0.58	0.84	0.16	0.61	0.90	0.40	0.58
PyramidKV (32.0×)	1	1.00	0.46	0.00	0.32	0.14	0.00	0.03	0.01	0.20	0.00	0.03	0.78	0.38	0.26
VecInfer (4.3×)	1	1.00	0.98	1.00	0.96	0.80	0.98	0.95	0.99	0.81	0.91	0.90	0.80	0.36	0.88
VecInfer (12.7×)	1	0.02	0.04	0.04	0.12	0.02	0.00	0.01	0.04	0.01	0.05	0.43	0.26	0.14	0.09
VecInfer (35.9×)	1	0.02	0.04	0.04	0.12	0.02	0.00	0.01	0.04	0.01	0.13	0.53	0.20	0.10	0.10

Notes: seeds states how many complete finetuning seeds the score above averages; ± is omitted for compactness. Ratio: uncompressed bytes per token per layer over the cache actually held (§4.4), not the nominal budget. Left out, the arm's solver reaches no configuration at this budget on this checkpoint: PyramidKV at 42 and 10 B and VecInfer at 42 and 10 B.

G.4 Per-Task Long-Context Results: LongBench

Tables 16 and 17 give LongBench on Qwen3-0.6B-Base and Qwen3-1.7B-Base, each task in its own metric (`narrativeqa` F1, `gov_report` ROUGE-L, `trec` accuracy, all from 0 to 100) and never averaged across tasks; each task is averaged only over the documents that every configuration compared in its table scored.

Table 16: Per-task LongBench results on Qwen3-0.6B-Base. Finetuned methods use 0.25 finetuning tokens per parameter; KIVI, PyramidKV and VecInfer are training-free. Each task is scored in its own metric, with over-length prompts truncated, over the 20 documents per task every compared configuration scored.

Method	B/tok	Ratio	seeds	n	skip	gov_report	narrativeqa	trec
Uncompressed	–	–	1	60	0	19.2	13.8	10.0
Compression 1.4×–2.8×								
Transformer-FSQ	2048	2.0	3	180	0	16.9±1.0	6.9±1.2	25.0±5.0
TransMLA	2048	2.0	3	180	0	12.7±0.0	2.7±0.6	16.7±5.8
VecInfer	1601§	2.6	1	60	0	18.1	8.0	10.0
Compression 2.8×–5.7×								
Transformer-FSQ	1024	4.0	3	180	0	16.4±1.4	5.0±1.8	25.0±10.0
Palu	1040	3.9	3	180	0	17.5±0.4	10.7±2.5	16.7±2.9
Palu	784	5.2	3	180	0	16.0±1.5	10.5±0.5	20.0±5.0
TransMLA	1024	4.0	3	180	0	12.9±0.1	2.7±0.2	8.3±5.8
KIVI	832§	4.9	1	60	0	15.1	7.8	35.0
KIVI@8k	832§	4.9	1	60	0	15.1	7.8	35.0
Compression 5.7×–11.3×								
Transformer-FSQ	512	8.0	3	180	0	16.2±0.8	6.8±2.6	18.3±2.9
FSQ-small (2048 B)	682	6.0	1	60	0	14.4	6.6	30.0
FSQ-small (1024 B)	682	6.0	3	180	0	14.9±0.5	6.7±0.4	30.0±5.0
FSQ-small	512	8.0	3	180	0	14.0±1.1	4.1±0.6	35.0±8.7
Palu	400	10.2	3	180	0	14.5±0.3	6.1±0.5	5.0±5.0
TransMLA	512	8.0	3	180	0	9.3±1.0	1.6±0.4	10.0±8.7
PyramidKV	509§	8.0	1	60	0	12.7	10.4	10.0
VecInfer	705§	5.8	1	60	0	13.1	2.9	15.0
Compression 11.3×–22.6×								
Transformer-FSQ	256	16.0	3	180	0	13.9±1.2	5.2±2.2	33.3±16.1
FSQ-small	256	16.0	3	180	0	13.7±0.5	4.5±0.5	31.7±5.8
Palu	208	19.7	3	180	0	12.1±0.4	2.2±0.5	15.0±5.0
TransMLA	256	16.0	3	180	0	8.5±0.7	1.5±0.6	6.7±7.6
PyramidKV	238§	17.2	1	60	0	11.3	9.4	15.0
PyramidKV@8k	226§	18.1	1	60	0	11.2	14.3	15.0
VecInfer	322§	12.7	1	60	0	6.1	2.9	5.0
VecInfer@8k	225§	18.2	1	60	0	6.1	0.7	0.0
Compression 22.6×–45.3×								
Transformer-FSQ	128	32.0	3	180	0	11.8±1.9	4.3±0.4	11.7±11.5
FSQ-small	128	32.0	3	180	0	11.3±1.1	4.5±1.3	18.3±5.8
TransMLA	128	32.0	3	180	0	3.6±0.3	1.1±1.1	0.0
PyramidKV	103§	39.6	1	60	0	8.2	8.4	20.0
PyramidKV@8k	96§	42.5	1	60	0	8.4	8.8	20.0
VecInfer@8k	113§	36.1	1	60	0	3.9	0.4	15.0
Compression 90.5×–181.0×								
Transformer-FSQ	42	97.5	3	180	0	7.6±0.8	3.2±1.0	3.3±5.8
FSQ-small	42	97.5	3	180	0	7.9±1.0	4.0±1.0	5.0±5.0
Palu	40	102.4	3	180	0	0.3±0.5	1.6±0.7	0.0
TransMLA	42	97.5	3	180	0	2.3±1.2	0.7±0.6	0.0
Compression 362.0×–724.1×								
Transformer-FSQ	10	409.6	3	180	0	1.6±0.1	1.8±0.3	0.0
FSQ-small	10	409.6	3	180	0	1.4±0.5	2.3±0.4	0.0
Palu	8	512.0	3	180	0	7.3±1.5	1.5±0.6	0.0
TransMLA	10	409.6	3	180	0	0.9±0.8	2.5±2.0	0.0

Notes: ±: one sample standard deviation over **seeds**. **B/tok**: cache bytes per token per layer; **ratio**: uncompressed size over **B/tok**; n/**skip**: examples scored/not, over seeds and tasks. Task columns: raw scores (0–100) over the caption's documents, then seeds; never averaged across tasks. `@` suffix (e.g. KIVI@8k): cache sized at that context, not 2048 tokens; its own row. §: **B/tok**, **ratio** depend on tokens held (KIVI, PyramidKV, VecInfer, residual-window FSQ variants); averaged over the row's scored examples.

Table 17: Per-task LongBench results on Qwen3-1.7B-Base. Finetuned methods use 0.25 finetuning tokens per parameter; KIVI, PyramidKV and VecInfer are training-free. Each task is scored in its own metric, with over-length prompts truncated, over the 20 documents per task every compared configuration scored.

Method	B/tok	Ratio	seeds	n	skip	gov_report	narrativeqa	trec
Uncompressed	–	–	1	60	0	18.1	14.9	35.0
Compression 1.4×–2.8×								
Transformer-FSQ	2048	2.0	3	180	0	15.9±0.3	9.7±3.3	21.7±5.8
TransMLA	2048	2.0	3	180	0	14.6±0.1	4.1±1.5	20.0±5.0
VecInfer	1601§	2.6	3	180	0	17.1±0.5	10.1±1.6	38.3±2.9
Compression 2.8×–5.7×								
Transformer-FSQ	1024	4.0	3	180	0	15.1±0.6	8.3±1.5	35.0±10.0
Palu	1040	3.9	3	180	0	17.6±0.3	11.5±1.1	45.0±5.0
Palu	784	5.2	3	180	0	16.4±0.7	11.4±1.2	36.7±2.9
TransMLA	1024	4.0	3	180	0	13.2±0.4	3.8±0.4	15.0±8.7
KIVI	832§	4.9	3	180	0	16.5	14.1	50.0
Compression 5.7×–11.3×								
Transformer-FSQ	512	8.0	3	180	0	15.3±0.6	6.2±2.4	26.7±2.9
Palu	400	10.2	3	180	0	13.1±0.2	7.0±1.4	56.7±7.6
TransMLA	512	8.0	3	180	0	8.9±0.5	2.3±1.4	6.7±5.8
PyramidKV	504§	8.1	3	180	0	11.7	12.1	35.0
VecInfer	705§	5.8	3	180	0	13.9±0.6	5.0±2.3	35.0±5.0
Compression 11.3×–22.6×								
Transformer-FSQ	341	12.0	3	180	0	14.4±0.1	5.1±1.6	30.0±8.7
Transformer-FSQ	256	16.0	3	180	0	13.4±0.7	4.9±0.6	26.7±5.8
Palu	208	19.7	3	180	0	10.6±0.7	3.8±1.1	11.7±7.6
TransMLA	256	16.0	3	180	0	5.8±0.8	1.7±0.9	5.0±5.0
PyramidKV	229§	17.9	3	180	0	9.9	14.6	30.0
PyramidKV@8k	216§	18.9	1	60	0	9.1	12.8	30.0
VecInfer	322§	12.7	3	180	0	6.0±0.4	3.8±1.5	25.0±5.0
VecInfer@8k	226§	18.2	1	60	0	6.7	2.4	30.0
Compression 22.6×–45.3×								
Transformer-FSQ	128	32.0	3	180	0	12.7±0.1	3.7±1.3	28.3±5.8
TransMLA	128	32.0	3	180	0	4.4±1.1	1.5±0.3	0.0
PyramidKV	93§	44.0	3	180	0	8.2	12.3	25.0
VecInfer	129§	31.6	3	180	0	6.0±0.9	3.1±1.1	18.3±5.8
VecInfer@8k	113§	36.1	1	60	0	6.8	1.3	25.0
Compression 45.3×–90.5×								
PyramidKV	47§	87.4	3	180	0	7.1	14.8	20.0
PyramidKV@8k	86§	47.8	1	60	0	8.5	18.5	25.0
Compression 90.5×–181.0×								
Transformer-FSQ	42	97.5	3	180	0	9.5±1.7	3.7±0.6	21.7±2.9
Palu	40	102.4	3	180	0	0.1±0.1	0.5±0.4	0.0
TransMLA	42	97.5	3	180	0	3.3±0.5	0.3±0.6	0.0
Compression 362.0×–724.1×								
Transformer-FSQ	10	409.6	3	180	0	2.4±2.1	3.0±0.7	0.0
Palu	8	512.0	3	180	0	1.9±1.5	0.6±0.5	0.0
TransMLA	10	409.6	3	180	0	0.3±0.4	0.8±0.8	0.0

Notes: ±: one sample standard deviation over **seeds**. **B/tok**: cache bytes per token per layer; **ratio**: uncompressed size over **B/tok**; n/**skip**: examples scored/not, over seeds and tasks. Task columns: raw scores (0–100) over the caption's documents, then seeds; never averaged across tasks. @ suffix (e.g. PyramidKV@8k): cache sized at that context, not 2048 tokens; its own row. §: **B/tok**, **ratio** depend on tokens held (KIVI, PyramidKV, VecInfer, residual-window FSQ variants); averaged over the row's scored examples.

Table 18 gives the same breakdown for Qwen3-8B-Base, from the same audited 200-document-per-task campaign as Figure 5.

Table 18: Per-task LongBench results on Qwen3-8B-Base. Finetuned methods use 0.25 finetuning tokens per parameter; KIVI, PyramidKV and VecInfer are training-free. Each task is scored in its own metric, with over-length prompts truncated, over the 179 gov_report, 70 narrativeqa and 200 trec documents every compared configuration scored.

Method	**B/tok**	**Ratio**	**seeds**	n	**skip**	**gov_report**	**narrativeqa**	**trec**
Uncompressed	–	–	1	462	138	16.8	17.4	63.0
Compression 2.8×–5.7×								
KIVI	847[§]	4.8	1	450	150	15.4	23.5	58.0
Compression 5.7×–11.3×								
VecInfer	721[§]	5.7	1	450	150	15.6	16.3	59.0
Compression 11.3×–22.6×								
PyramidKV	287[§]	14.3	1	478	122	10.5	18.8	60.5
Compression 22.6×–45.3×								
Transformer-FSQ	128	32.0	1	477	123	12.5	8.4	34.5
Palu	112	36.6	1	462	138	8.0	3.0	2.0
VecInfer	145[§]	28.2	1	462	138	9.2	7.4	20.0
Compression 45.3×–90.5×								
PyramidKV	56[§]	73.2	1	480	120	5.4	11.5	27.5
Compression 90.5×–181.0×								
Transformer-FSQ	42	97.5	1	465	135	11.5	7.2	3.5
Palu	40	102.4	1	462	138	7.0	2.2	2.0
Compression 362.0×–724.1×								
Transformer-FSQ	10	409.6	1	478	122	8.6	4.5	5.0
Palu	8	512.0	1	462	138	3.6	2.2	0.0

Notes: $\pm$: one sample standard deviation over **seeds**. **B/tok**: cache bytes per token per layer; **ratio**: uncompressed size over **B/tok**; n/**skip**: examples scored/not, over seeds and tasks. Task columns: raw scores (0–100) over the caption's documents, then seeds; never averaged across tasks. [§]: **B/tok**, **ratio** depend on tokens held (KIVI, PyramidKV, VecInfer, residual-window FSQ variants); averaged over the row's scored examples.