2026 하계 세미나

KV Cache Compression

Sogang University
Vision & Display Systems Lab, Dept. of Electronic Engineering

Presented By
임나현

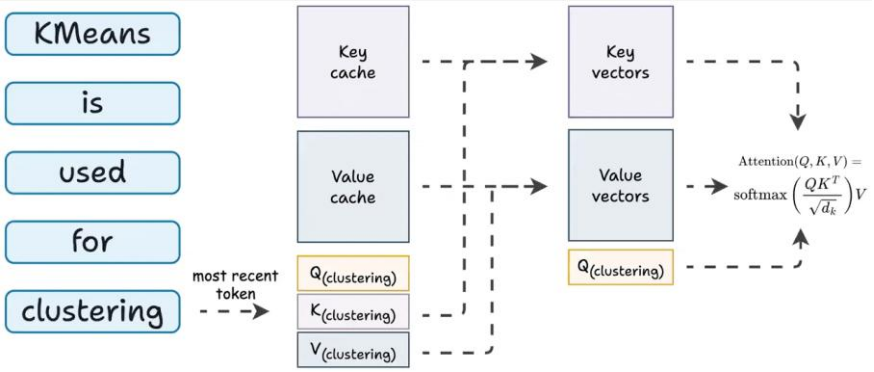
Outline

- Background
  - KV Cache
  - KV Cache Compression
  - Visual Autoregressive (VAR) Modeling

- Papers
  - FreqKV: Key-Value Compression in Frequency Domain for Context Window Extension [ICLR 2026]
  - HACK: Head-Aware KV Cache Compression for Efficient Visual Autoregressive Modeling [AAAI 2026]

Background

- KV Cache
  - Decoding 시 이전 token의 Key/Value를 cache에 저장하여 재사용
  - 한계: cache 크기가 sequence length에 선형으로 증가
    - LLaMA-2-7B 기준 32K context에서 KV cache만 수십 GB 수준까지 증가
    - Cache를 읽어오는 memory bandwidth가 decoding 속도의 병목이 됨
- KV Cache Compression
  - 제한된 memory budget에서 중요한 KV만 유지하여 cache를 압축하는 기술
    - 메모리 사용량과 attention 계산량을 줄이면서 성능을 최대한 유지하는 것이 목표



< Fig 1. KV cache >

Background

- 기존 KV Cache Compression 방식
  - Quantization
    - KV state를 FP16 대신 4-bit 등 저비트로 표현하여 cache 용량을 직접 축소
      - Token 수는 그대로이므로 attention 연산량은 줄지 않고, outlier에 의한 오차가 누적
  - Eviction
    - Attention score가 낮은 token의 KV를 삭제하여 cache 길이를 축소
      - 삭제된 token의 정보가 영구적으로 손실되어 이후 decoding에서 복원 불가
      - Context window를 넘어가면 position embedding이 범위를 벗어나 성능이 붕괴
  - Merging
    - 버려질 token을 유사한 token에 병합하여 정보를 일부 보존
      - Fine-tuning 없이는 근사 오차로 인해 성능 저하가 발생

Background

- Visual Autoregressive (VAR) Modeling
  - 기존 visual AR: 이미지를 discrete token으로 양자화한 뒤 next-token prediction
    - Token 수가 많아 decoding step이 길고 inference latency가 큼
  - VAR: next-token이 아닌 next-scale prediction 패러다임
    - Feature map을 K개의 multi-scale token map R = (r_1, …, r_K)로 양자화

$$p(r_1, \ldots, r_K) = \prod_{k=1}^{K} p(r_k \mid r_1, \ldots, r_{k-1})$$

    - Scale 내부 token은 병렬 생성 → K step만에 coarse-to-fine 생성
  - KV Cache in VAR
    - Scale k의 생성은 이전 모든 scale의 token을 contextual prefix로 사용

$$T_k = \sum_{i=1}^{k} h_i w_i, \qquad h_k w_k = a^{2(k-1)}$$

    - Generation이 진행될수록 이전 모든 scale의 KV cache가 누적 저장

FreqKV: Key-Value Compression in Frequency Domain for Context Window Extension [ICLR 2026]

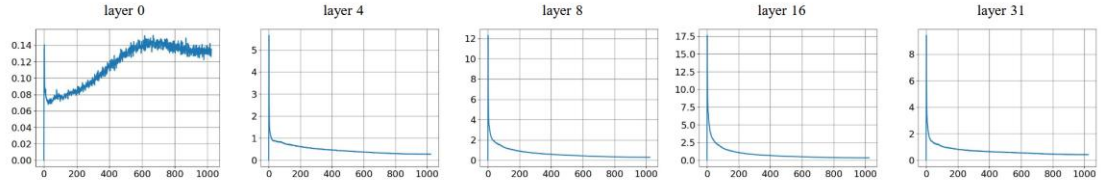
1) Kai, Jushi, et al. "FreqKV: Key-Value Compression in Frequency Domain for Context Window Extension." International Conference on Learning Representations. 2026.

FreqKV[1)]

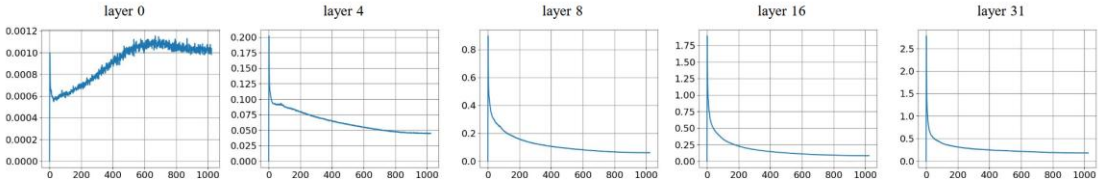
- Introduction
  - Problem statements
    - LLM은 pre-training 시 정해진 context window를 넘어서면 성능이 급격히 저하
    - 기존 KV compression은 주로 token eviction 또는 token merging에 기반
      - 중요도가 낮은 token을 제거하므로 해당 정보가 영구적으로 손실됨
      - 여러 KV state를 하나로 근사하므로 원래 full-attention 결과를 정확히 보존하기 어려움
  - Key contributions
    - KV state의 에너지가 frequency domain의 low-frequency 성분에 집중됨을 관찰
    - DCT 기반 frequency-domain KV compression 기법 FreqKV를 제안
      - Parameter-free: 추가 학습 파라미터나 압축 모듈 없이 KV cache 압축 가능
    - 8K context fine-tuning만으로 최대 256K context에서도 안정적인 성능 유지

FreqKV[1)]

- Observations
  - KV state의 frequency-domain power spectrum



(a) The average power spectrums of key states.



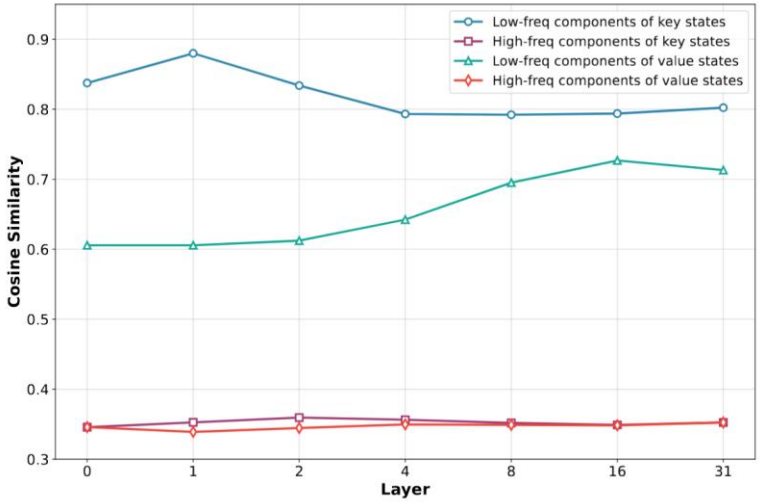
(b) The average power spectrums of value states.

< Fig 2. Power spectrums of KV states >

- K, V state를 sequence dimension을 따라 DCT 변환한 뒤 power spectrum 분석
- 초기 layer 제외, layer가 깊어질수록 K와 V 모두 low-frequency 영역에 energy가 집중
  - KV state에서 공통적으로 low-frequency-dominant spectral pattern이 관찰
- Low-frequency를 보존하는 frequency-domain compression의 가능성을 제시

FreqKV[1)]

- Observations
  - Low-frequency vs. High-frequency
    - 100개의 문서에서 전체 단어 1%를 동의어로 치환하는 perturbation 실험
      - Low-frequency 성분은 perturbation 이후에도 높은 cosine similarity를 유지
      - High-frequency 성분은 작은 입력 변화에 상대적으로 민감
    - Low-frequency는 global semantic, high-frequency는 local detail을 주로 반영
    - Summarization task에서 한쪽 성분만 유지하여 성능을 평가
      - Low-frequency만 유지한 경우 모든 데이터셋에서 더 높은 요약 성능을 보임



< Fig 3. Cosine similarity >

Components Retained	GovReport	QMSum	MultiNews
high-frequency	14.21	16.54	15.55
low-frequency	**25.51**	**21.81**	**26.9**

< Table 1. Performance on summarization >

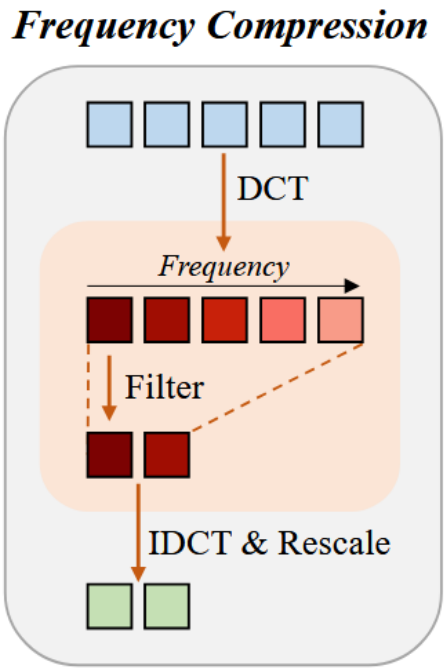
FreqKV[1)]

- Methods
  - Frequency-domain KV Compression
    - KV cache를 DCT로 frequency domain으로 변환

$$A(N) = \text{Softmax}\left(\frac{q_N[K_{0:N-1} \oplus k_N]^T}{\sqrt{d}}\right) \cdot [V_{0:N-1} \oplus v_N]$$

$$Z^K_{0:N-1} = \text{DCT}(K_{0:N-1}), \qquad Z^V_{0:N-1} = \text{DCT}(V_{0:N-1})$$

    - Low-frequency 성분만 유지
      - High-frequency 성분은 제거
    - IDCT로 다시 time domain으로 복원
      - Retaining ratio γ에 대해 cache length가 L = γN으로 축소
      - LLaMA-2 기준 N = 4092 → L = 2046 (γ = 0.5)
    - Token을 삭제하는 것이 아니라 전체 token의 정보를 압축
      - 모든 token의 기여가 compressed cache에 남아 있음
      - Eviction과의 근본적 차이



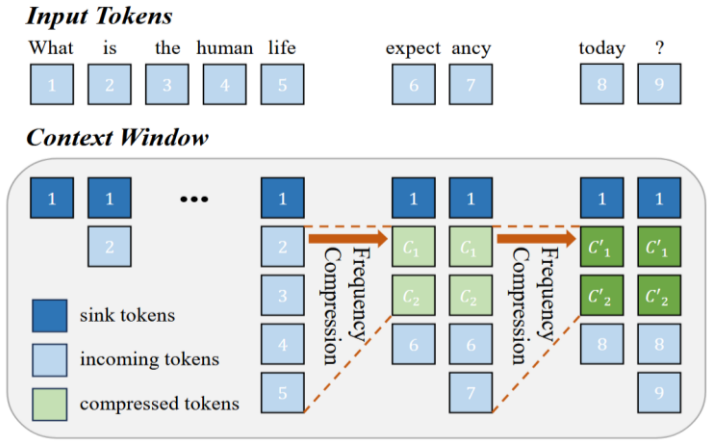
< Fig 4. Frequency compression >

FreqKV[1)]

- Methods
  - Context Extension via Iterative Compression
    - Context window 내부에서는 일반적인 full attention을 그대로 수행
    - Cache가 가득 차면 frequency-domain compression을 수행
    - 이후 새로운 token을 compressed cache 뒤에 계속 추가
    - Cache가 다시 차면 compressed cache와 새 token을 함께 재압축

$$\tilde{A}(N,L) = \text{Softmax}\left(\frac{q_N\left[\tilde{K}_{0:L-1} \oplus k_N\right]^T}{\sqrt{d}}\right) \cdot \left[\tilde{V}_{0:L-1} \oplus v_N\right]$$

    - 오래된 token일수록 여러 번 압축되고 최근 token은 거의 손상되지 않음



< Fig 5. Context Extension >

FreqKV[1)]

- Methods
  - Rescaling
    - DCT/IDCT는 component 개수의 제곱근으로 signal을 normalize
      - 길이 N으로 정규화된 신호를 길이 L로 잘라 IDCT하면 amplitude가 어긋남
    - 이를 보정하기 위해 √(L/N)의 rescaling factor를 적용

$$\widetilde{K}_{0:L-1}^{0:N-1} = \sqrt{\frac{L}{N}}\,\mathrm{IDCT}(Z_{0:N-1}^{K}[0:L-1])$$

    - 압축된 cache는 이후 attention에서 일반 KV cache와 동일하게 사용
  - Computational cost
    - Compression은 N − L개의 token이 새로 유입될 때마다 한 번만 수행
    - 한 번의 compression 비용은 O(N log N)으로 전체 decoding latency 대비 0.3% 미만

FreqKV[1)]

- Methods
  - Sink Tokens
    - LLM은 초기 token에 높은 attention을 부여하는 attention sink 현상을 보임
    - FreqKV는 초기 S개의 sink token을 압축하지 않고 그대로 보존

$$\tilde{A}(S,N,L,M) = \text{Softmax}\left(\frac{q_M\left[K_{0:S-1} \oplus \tilde{K}_{0:L-1} \oplus K_{N:M}\right]^T}{\sqrt{d}}\right) \cdot \left[V_{0:S-1} \oplus \tilde{V}_{0:L-1} \oplus V_{N:M}\right]$$

  - RoPE Handling
    - Key state는 RoPE를 적용하기 전 상태로 압축되어 cache에 저장
    - Attention 계산 시점에 compressed key에 RoPE를 적용
    - Compressed token에는 원래 sequence position이 아니라 cache 내부의 position을 부여
      - Cache 길이가 항상 context window 이내이므로 position이 학습 범위를 벗어나지 않음

FreqKV[1)]

- Experiments
  - Long-context Language Modeling
    - FreqKV는 compressed cache를 사용하면서도 full cache와 동등하거나 더 낮은 PPL 달성
    - 8K 학습만으로 32K 평가 길이에서도 PPL 7.02로 안정적으로 유지
    - LongLoRA/LoCoCo는 원래 window(2K, 4K) 내에서 오히려 성능이 저하됨

Size	Training Length	Method	Inference Cache	Evaluation Context Length				
				2048	4096	8192	16384	32768
7B	8192	Full FT	Full	7.55	7.21	6.98	-	-
		LoCoCo*	Compressed	8.15	8.08	7.27	-	-
		LongLoRA	Full	7.70	7.35	7.14	-	-
		FreqKV	Compressed	**7.45**	**7.12**	**7.04**	**7.02**	**7.02**
	16384	Full FT	Full	OOM during Training				
		LoCoCo	Compressed	OOM during Training				
		LongLoRA	Full	7.65	7.28	**7.02**	**6.86**	-
		FreqKV	Compressed	**7.46**	**7.13**	7.03	6.99	**6.98**
	32768	LongLoRA	Full	8.29	7.83	7.54	7.35	7.22
		FreqKV	Compressed	**7.47**	**7.14**	**7.04**	**7.00**	**6.98**
13B	8192	Full FT	Full	6.95	6.60	6.43	-	-
		LongLoRA	Full	7.03	6.73	6.58	-	-
		FreqKV	Compressed	**6.82**	**6.52**	**6.44**	**6.42**	**6.41**

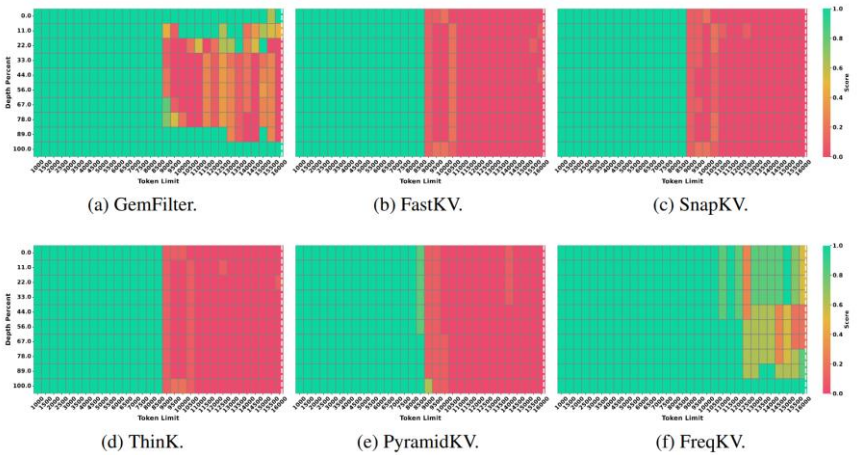
< Table 2. Perplexity evaluation >

FreqKV[1)]

- Experiments
  - Long-context Understanding
    - LLaMA-3 context window를 넘는 16K 평가에서 기존 방법은 모두 붕괴 (≈0)
      - 압축 이후 out-of-bound position embedding 사용이 원인
    - FreqKV는 16K에서도 평균 47.30으로 유일하게 정상 동작
    - Needle-in-a-Haystack에서도 원래 window를 넘어선 구간에서 FreqKV만 정답을 검색

Evaluation Length	Method	CWE	FWE	QA	VT	NIAH	Avg.
8K	LLaMA-3	96.40	82.33	62.50	95.00	97.81	86.81
	+D2O	93.20	81.00	62.00	88.80	47.53	74.51
	+GemFilter	**96.40**	82.33	62.50	95.00	**97.81**	86.81
	+FastKV	86.10	77.67	63.50	93.00	87.03	81.46
	+SnapKV	96.00	80.33	62.50	93.60	92.28	84.94
	+ThinK	95.7	78.67	62.00	96.8	94.91	85.62
	+PyramidKV	**96.40**	82.33	62.50	95.00	**97.81**	86.81
	+FreqKV	91.80	**84.67**	**64.50**	**100.00**	97.44	**87.68**
16K	+D2O	-	-	-	-	-	OOM
	+GemFilter	0.00	0.00	0.50	0.00	0.00	0.10
	+FastKV	0.00	0.00	0.00	0.00	0.00	0.00
	+SnapKV	0.00	0.00	0.50	0.00	0.00	0.10
	+ThinK	0.00	0.00	0.00	0.00	0.00	0.00
	+PyramidKV	0.00	0.00	0.50	0.00	0.00	0.10
	+FreqKV	**45.10**	**87.33**	**34.00**	**44.60**	**25.47**	**47.30**

< Table 3. RULER results >



< Fig 6. Needle-in-a-Haystack results >

FreqKV[1)]

- Analysis
  - Efficient Extrapolation
    - 8K context로만 학습한 모델을 최대 256K의 문맥에 적용하여 일반화 성능을 평가
    - Compression latency는 전체 decoding latency의 0.3% 미만
  - Training Cost
    - 학습 시간 28.88h → 17.07h, 메모리 44.04 GB → 24.58 GB (약 40% 절감)

Evaluation Length	8K	16K	32K	64K	128K	256K
Compression Count	3	7	15	31	63	127
Decoding Latency	214.96	445.69	907.49	1832.77	3692.03	7422.64
Compression Latency	0.62 (0.29%)	1.09 (0.24%)	2.18 (0.24%)	4.00 (0.22%)	8.31 (0.23%)	16.39 (0.22%)
PPL	7.76	7.74	7.73	7.73	7.73	7.73

< Table 4. Extrapolation performance >

Method	Training Length	Time (hours)	Memory (GB)
Full FT	8K	28.88	44.04
FreqKV	8K	17.07	24.58
	16K	38.57	34.00
	32K	89.35	45.37

< Table 5. Training cost >

HACK: Head-Aware KV Cache Compression for Efficient Visual Autoregressive Modeling [AAAI 2026]

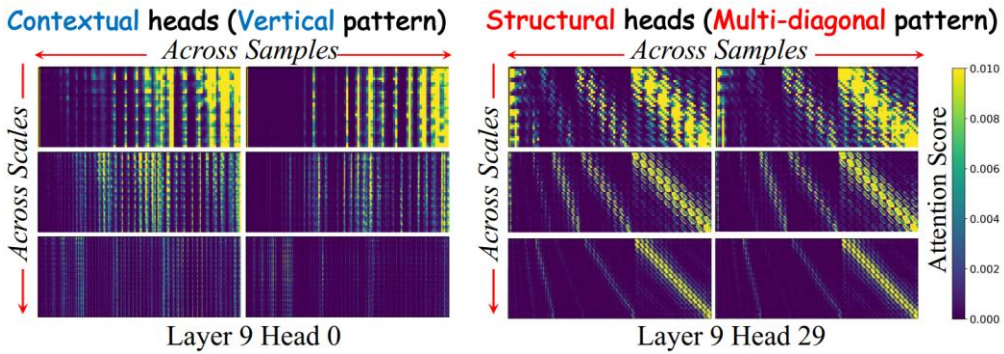
1) Qin, Ziran, et al. "Head-Aware KV Cache Compression for Efficient Visual Autoregressive Modeling." *Proceedings of the AAAI Conference on Artificial Intelligence*. 2026.

HACK[1)]

- Introduction
  - Problem statements
    - VAR 모델은 next-scale prediction을 통해 적은 수의 step만에 고품질 이미지를 생성
    - Scale이 증가할수록 이전 모든 scale의 KV cache가 지속적으로 누적
      - Attention 복잡도 $O(n^4)$로 고해상도 이미지 생성 시 연산량이 급격하게 증가
        - ✓ Infinity: 1024×1024 이미지 생성 시 10K개 이상의 token 처리 필요
    - LLM용 KV compression을 그대로 적용하면 VAR에서 성능이 크게 저하
  - Key contributions
    - VAR의 attention head를 contextual 과 structural 두 유형으로 분리
    - Training-free head-aware compression 기법 HACK 제안
      - Asymmetric budget allocation과 pattern-specific compression을 결합
      - Attention 복잡도: $O(n^4) \rightarrow O(Bn^2)$로 감소, 70% 압축에서도 품질 유지

HACK[1)]

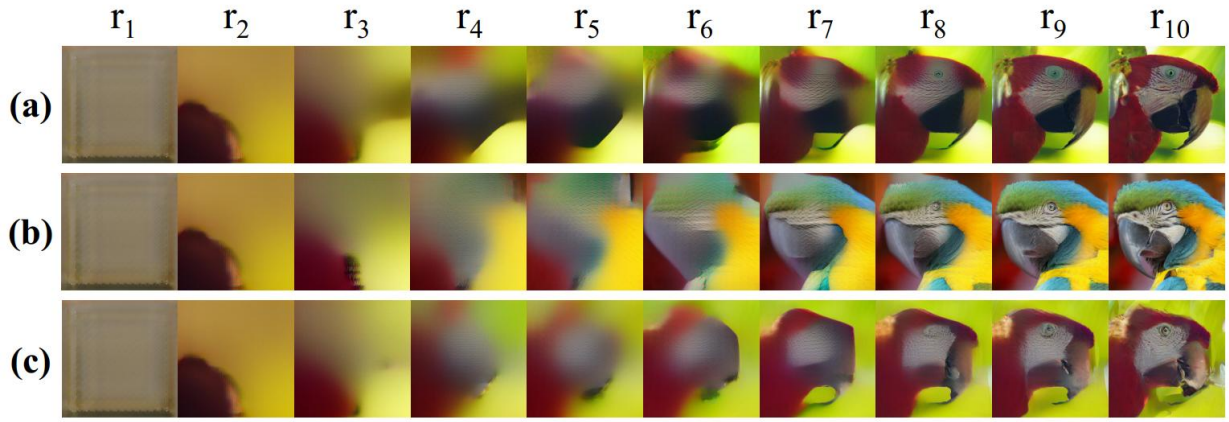
- Observations
  - Two distinct attention patterns
    - Contextual heads
      - 소수의 key token에 집중하는 vertical attention pattern
      - Global semantic consistency를 유지하는 역할
    - Structural heads
      - 공간적으로 인접한 token에 attention하는 multi-diagonal pattern
      - Spatial coherence와 geometry를 보존하는 역할
    - 두 pattern 모두 sample과 scale이 바뀌어도 일관되게 유지



< Fig 7. Attention patterns >

HACK[1)]

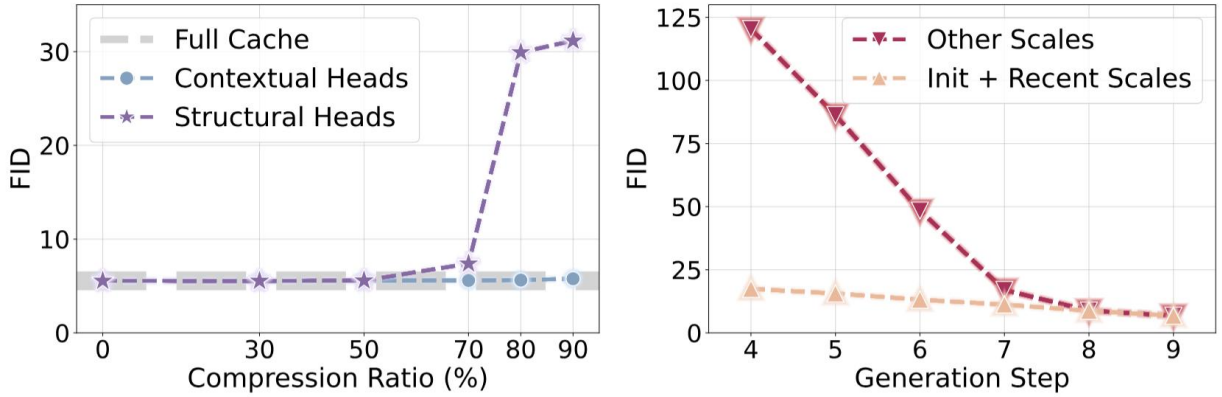
- Observations
  - Head masking 분석
    - 각 head type의 10%를 masking하여 생성 결과를 비교
    - Contextual head masking (b)
      - 생성 객체의 semantic consistency가 크게 변화
      - 전체적인 공간 구조는 비교적 유지
    - Structural head masking (c)
      - 생성 객체의 의미는 대부분 유지
      - 형태 왜곡 및 spatial distortion 발생



< Fig 8. Head masking >

HACK[1)]

- Observations
  - Compression sensitivity & Scale importance
    - Structural head에 더 많은 cache budget을 할당
      - Contextual head: 90% compression에서도 이미지 품질 거의 유지
      - Structural head: 압축률이 50%를 넘으면 품질 급격히 저하
  - Scale importance
    - LLM에서 초기에 등장한 token과 가장 최근 token이 더 높은 attention을 받음
    - Initial scale과 recent scale의 token을 보존하는 전략이 중간 scale 균등 보존보다 우수



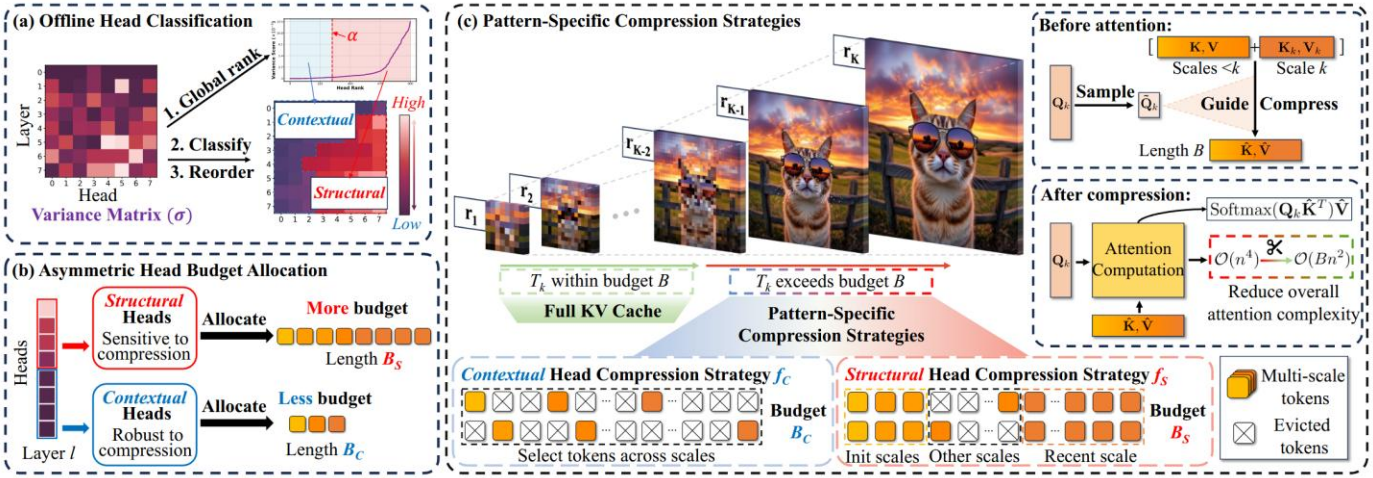
< Fig 9. Analysis on VAR-d30 >

HACK[1)]

- Methods
  - Framework

(a) Offline head classification → (b) Asymmetric budget → (c) Pattern-specific compression

- KV cache 길이가 해당 head group의 budget을 초과할 때만 압축 수행
- Attention 계산 직전에 압축하므로 메모리와 계산량 모두 bounded
- 평균 per-head budget B를 유지하며 attention 복잡도를 O(Bn^2)로 감소



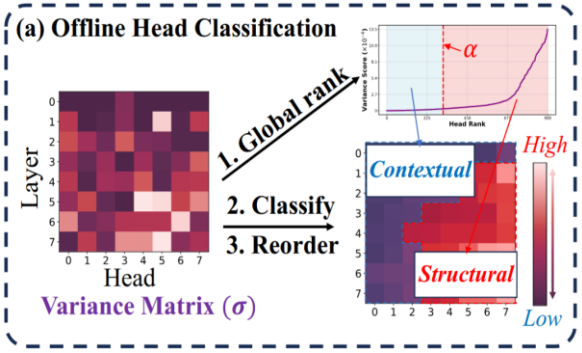
< Fig 10. HACK framework >

HACK[1)]

- Methods
  - Offline Head Classification via Attention Variance
    - 최종 generation scale의 attention matrix를 사용하여 head를 사전 분류
    - 각 head마다 column-wise variance 합을 계산

$$\sigma_{l,h} = \sum_{j=1}^{T_K} Var(A_K^{(l,h)}[:, j])$$

    - 작은 validation set 평균으로 variance matrix $(L \times H)$를 구성
    - 전체 head를 variance 기준으로 global ranking
    - Contextual head ratio α로 head type을 결정



< Fig 11. Head classification >

- Contextual head
  - ✓Query가 바뀌어도 유사한 token에 집중
  - ✓Low variance
- Structural head
  - ✓Query 위치에 따라 attention이 이동
  - ✓High variance

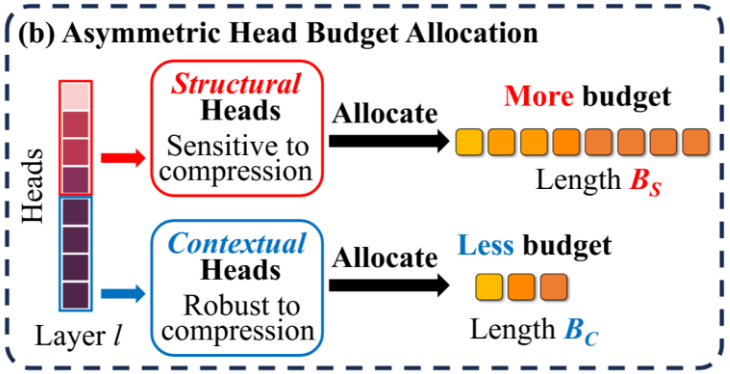
HACK[1)]

- Methods
  - Asymmetric Head Budget Allocation
    - 평균 per-head budget B 를 유지하면서 head type 별로 비대칭적으로 budget 을 분배

$$B = \alpha B_C + (1-\alpha)B_S, \qquad B_C << B_S$$

      - α는 전체 head 중 contextual head 의 비율
      - Contextual head에는 작은 budget B_C, structural head에는 큰 budget B_S할당
    - 전체 query 대신 N_{obs}=32 개의 query를 sampling 하여 attention score 를 근사

$$\widetilde{A}_k^{(p)} = \text{Softmax}\left(\widetilde{Q}_k^{(p)}\left(K^{(p)}\right)^T\right)$$



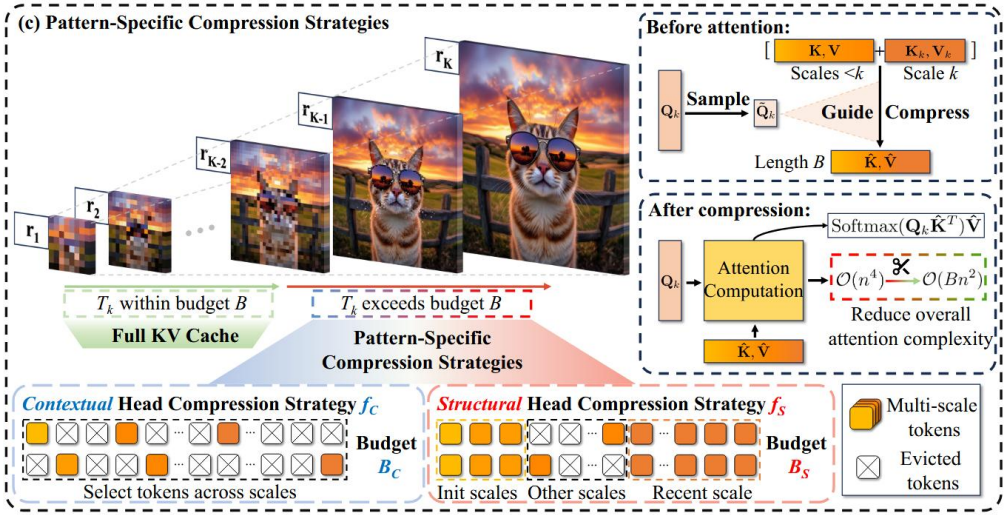
< Fig 12. Head budget >

HACK[1)]

- Methods
  - Contextual Head Compression Strategy
    - Subset attention score를 누적하여 상위 B_C개의 token을 Top-K로 선택
    - 선택된 token의 KV cache만 유지

$$I^{(C)} = \mathrm{TopK}\left(\sum_{i=1}^{N_{\mathrm{obs}}} \widetilde{A}_{k}^{(C)}[i,:], B_C\right)$$

$$\widehat{K^{(C)}} = K^{(C)}\left[I^{(C)},:\right], \qquad \widehat{V^{(C)}} = V^{(C)}\left[I^{(C)},:\right]$$



< Fig 13. Compression strategies >

HACK[1)]

- Methods
  - Structural Head Compression Strategy
    - 단순 Top-K token selection은 공간 구조를 훼손하므로 부적절
    - Scale-aware 압축 : initial scale 의 N_{init}개와 최근 scale 의 N_k개 token 을 항상 보존
    - 남은 budget M 으로 중간 scale 에서 중요도가 높은 token 선택

$$M = B_S - N_{init} - N_K$$

$$I^{(S)} = \mathrm{TopK}\left(\sum_{i=1}^{N_{\mathrm{obs}}} \widetilde{A_k^{(S)}}[i,:], M\right)$$

$$\hat{K}^{(S)} = \mathrm{Concat}\left(K^{(S)}[:N_{\mathrm{init}},:], K^{(S)}[I^{(S)},:], K^{(S)}[-N_k:,:]\right)$$
$$\hat{V}^{(S)} = \mathrm{Concat}\left(V^{(S)}[:N_{\mathrm{init}},:], V^{(S)}[I^{(S)},:], V^{(S)}[-N_k:,:]\right)$$

    - 마지막 generation scale에서만 discarded token을 nearest retained token에 merge

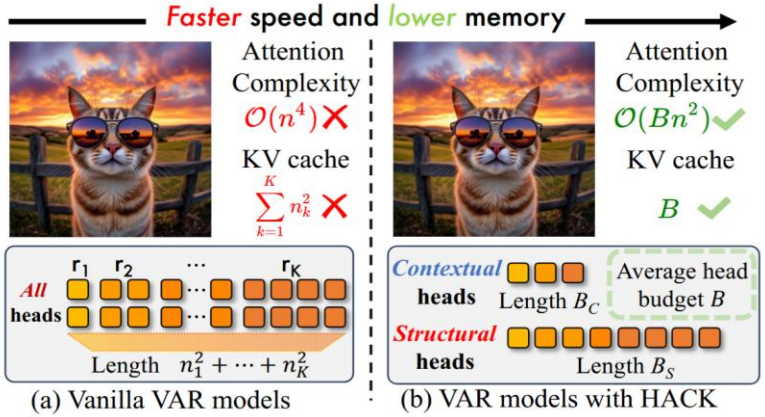
HACK[1)]

- Methods
  - Complexity analysis
    - Vanilla VAR
      - 모든 이전 scale의 KV cache를 유지
    - HACK
      - 평균 per-head budget B 이하로 KV cache 유지
      - Attention 계산 비용:

$$\sum_{k=1}^{K} N_k \cdot \min(T_k, B) \le B \sum_{k=1}^{K} a^{2(k-1)} \sim O(Bn^2)$$

      - Compression 추가 오버헤드:

$$\sum_{k=k_s}^{K} N_{obs}\,(B + N_k) = N_{obs} \sum_{k=k_s}^{K} \left(B + a^{2(k-1)}\right) \sim O(N_{obs}\, n^2)$$



< Fig 14. Attention complexity >

HACK[1)]

- Experiments
  - Quantitative Results
    - Infinity-2B/8B, HART(text-to-image), VAR-d30(class-conditional)에서 평가
    - 70% 압축에서도 GenEval/HPS/IR/FID/CLIP 전 지표에서 거의 손실 없음
    - VAR-d30 70% 압축 기준 FID 2.78로, LOOK-M(18.88) 대비 압도적으로 우수

Method	GenEval ↑	HPS↑	IR↑	FID↓	CLIP↑
Infinity-2B	0.68	30.49	0.946	10.34	27.52
+Streaming	0.68	29.76	0.901	11.20	27.54
+H2O	0.68	29.60	0.910	10.68	27.57
+SnapKV	0.68	29.60	0.904	10.60	27.56
+LOOK-M	0.67	28.73	0.864	11.14	27.59
+CAKE	0.68	29.46	0.906	10.59	27.56
+MEDA	0.67	28.70	0.867	11.14	27.59
+HACK	**0.68**	**30.18**	**0.933**	**10.56**	**27.62**
Infinity-8B	0.81	30.99	1.049	8.75	28.73
+Streaming	0.81	30.52	1.016	8.98	29.05
+H2O	0.81	30.53	1.020	9.04	29.02
+SnapKV	0.81	30.21	1.015	9.45	29.06
+LOOK-M	0.81	29.99	0.994	10.84	29.04
+CAKE	0.81	30.04	1.002	9.80	29.05
+MEDA	0.79	29.57	0.954	11.56	29.01
+HACK	**0.82**	**30.69**	**1.043**	**8.62**	**29.08**
Hart	0.50	28.75	0.658	10.70	27.69
+Streaming	0.48	25.57	0.458	11.16	27.38
+H2O	0.48	25.69	0.475	11.13	27.32
+SnapKV	0.47	25.62	0.469	11.33	27.33
+LOOK-M	0.37	20.83	0.140	25.64	25.85
+CAKE	0.46	25.89	0.458	12.88	27.21
+MEDA	0.36	20.57	0.117	28.37	25.71
+HACK	**0.50**	**28.36**	**0.658**	**8.58**	**27.76**

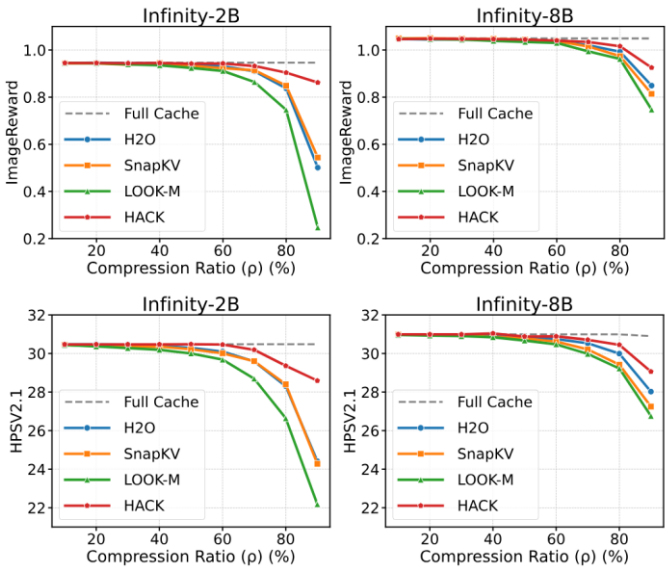
< Table 6. Text-to-image generation >

Method	ρ	FID↓	IS↑	Precision↑	Recall↑
VAR-d30	0%	1.96	302.23	0.81	0.60
+Streaming	30%	2.04	297.09	0.80	0.61
+H2O	30%	2.10	295.01	0.81	0.60
+SnapKV	30%	2.13	294.85	0.81	0.60
+LOOK-M	30%	3.32	255.36	0.76	0.61
+HACK	30%	**1.97**	**299.98**	0.81	0.61
+Streaming	50%	2.36	281.30	0.79	0.61
+H2O	50%	3.04	262.68	0.77	0.62
+SnapKV	50%	3.09	261.63	0.77	0.62
+LOOK-M	50%	6.89	203.47	0.70	0.62
+HACK	50%	**2.06**	**293.60**	0.80	0.61
+Streaming	70%	4.84	228.43	0.74	0.62
+H2O	70%	8.81	182.84	0.68	0.62
+SnapKV	70%	7.31	196.62	0.70	0.62
+LOOK-M	70%	18.88	116.70	0.58	0.63
+HACK	70%	**2.78**	**268.69**	0.78	0.62

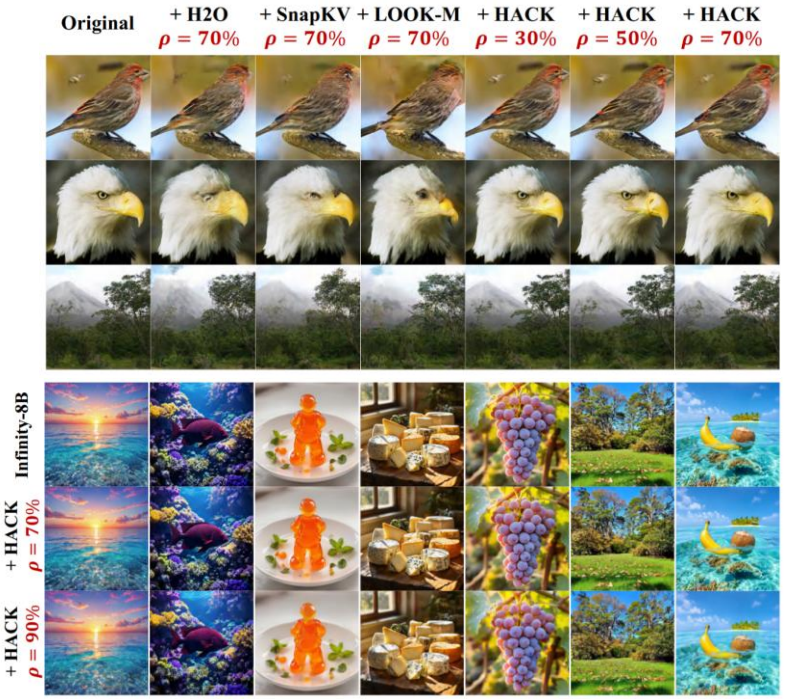
< Table 7. Class-conditional generation >

HACK[1)]

- Experiments
  - Compression Ratio & Qualitative Results
    - 압축률이 70%를 넘어서면 기존 방법은 지각 품질이 급격히 저하
    - HACK은 ρ = 90%의 극단적 압축에서도 pixel-level fidelity와 scene layout을 유지



< Fig 15. Performance comparison >



< Fig 16. Qualitative results >

HACK[1)]

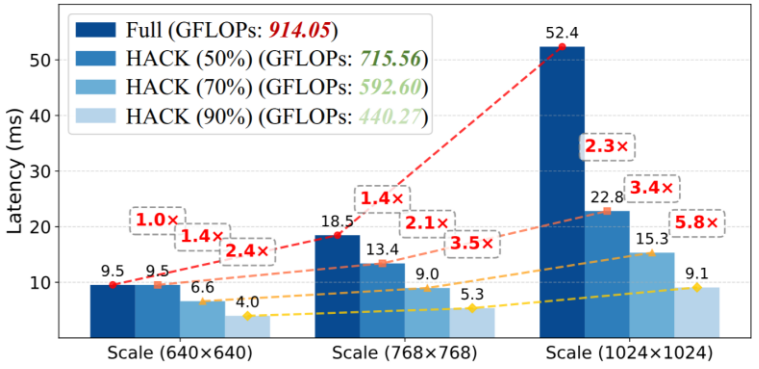
- Experiments
  - Efficiency Analysis & Ablation Study
    - Infinity-8B: memory 60.42GB → 34.44GB (1.75×), latency 8.14s → 5.17s (1.57×)
    - Compression overhead는 전체 latency 대비 약 6–8% 수준
    - 고해상도일수록 효과가 커져 1024×1024에서 최대 5.8× speedup (ρ=90%)
    - Uniform query sampling이 가장 높은 성능을 달성

Model	Random	Init	Recent	Uniform	Full
Infinity-2B	0.927	0.923	0.927	**0.933**	0.944
Infinity-8B	1.038	1.028	1.027	**1.043**	1.045

< Table 8. Query-subset selection >

Model	Memory	Com. O	Latency	Speedup
Infinity-2B	28.96GB	–	3.85s	1.00×
+HACK (70%)	14.64GB	0.16s	2.61s	1.48×
Infinity-8B	60.42GB	–	8.14s	1.00×
+HACK (70%)	34.44GB	0.44s	5.17s	1.57×
Hart	35.47GB	–	2.43s	1.00×
+HACK (70%)	23.82GB	0.15s	1.38s	1.76×

< Table 9. Efficiency analysis >



< Fig 17. Efficiency analysis >

1) Kai, Jushi, et al. "FreqKV: Key-Value Compression in Frequency Domain for Context Window Extension." International Conference on Learning Representations. 2026.
2) Qin, Ziran, et al. "Head-Aware KV Cache Compression for Efficient Visual Autoregressive Modeling." *Proceedings of the AAAI Conference on Artificial Intelligence*. 2026.

Conclusion

- FreqKV[1)]
  - Key contributions
    - KV state의 에너지가 frequency domain의 low-frequency에 집중됨을 규명
      - Low-frequency: global semantic context
      - High-frequency: local detail, perturbation에 민감
    - Token 삭제 없이 frequency component만 줄이는 DCT 기반 압축 제안
- HACK[2)]
  - Key contributions
    - VAR attention head를 기능적으로 분리
      - Contextual head: vertical pattern, semantic consistency 담당
      - Structural head: multi-diagonal pattern, spatial coherence 담당
    - Head type별 압축 민감도 차이를 정량화
      - 민감도에 따라 cache를 분배하는 Asymmetric Budget Allocation 제안
      - Pattern별로 다른 전략을 적용하는 Pattern-Specific Compression 도입

감사합니다.