SurrogateKV: 어텐션 안정화 기반 의미보존 KV 캐시 압축

김준원[○1], 유정현[1], 김종만[1,2], 김영재[†1]
[1]서강대학교 컴퓨터공학과, [2](주)소테리아
{kimjunewon, jhryu, youkim}@sogang.ac.kr, jkim@soteria-sys.com

SurrogateKV: Semantic-Preserving KV Cache Compression via Attention Stabilization

Junwon Kim[○1], Junghyun Ryu[1], James J. Kim[1,2], Youngjae Kim[†1]
[1]Department of Computer Science and Engineering, Sogang University, [2]Soteria Co., Ltd.
{kimjunewon, jhryu, youkim}@sogang.ac.kr, jkim@soteria-sys.com

요 약

Long-context LLM inference에서 KV cache는 메모리 사용량, 워커 간 전송량, TTFT를 좌우하는 주요 병목이다. 기존 token pruning 기반 압축은 cache 크기를 직접적으로 줄이지만, 제거된 chunk의 token을 attention 대상에서 제외해 attention mass 재분배와 문맥 표현 왜곡을 유발할 수 있다. 본 연구는 중요도가 낮은 chunk를 삭제하지 않고 attention이 참조 가능한 surrogate KV token으로 대체하는 *SurrogateKV*를 제안한다. *SurrogateKV*는 중요 chunk와 recent suffix를 원본 KV로 유지하고, victim chunk는 하나의 surrogate token으로 치환하여 cache 길이를 줄이면서 정보 손실을 완화한다. long-context workload 실험에서 *SurKV-Global*은 대부분의 KV budget 구간에서 기존 pruning 방법보다 높은 점수를 유지했다. 25% remaining cache에서 PyramidKV 대비 9.73점 높은 점수를 달성했고, TTFT는 FullKV 대비 최대 22.9 ms 증가에 그쳤다.

1. 서론

Long-context LLM serving에서 KV cache는 GPU 메모리 사용량뿐 아니라 워커 간 전송량, 메모리 계층 간 데이터 이동, 그리고 전체 추론 지연을 좌우하는 핵심 병목이다. PD-disaggregated환경 [1] 에서는 prefill과 decode가 서로 다른 워커에서 수행되므로, prefill 단계에서 생성된 KV cache를 decode 워커로 전송해야 한다. DistServe [1]는 이러한 KV cache 전송이 분리형 LLM 서빙의 주요 성능 제약임을 지적한다. 또한 LMCache [2]는 KV cache를 GPU에 종속된 request 버퍼가 아닌, host memory, storage tier에서 이동 · 공유 · 재사용 가능한 자원으로 다룬다. 이 경우에도 KV cache의 크기는 계층 간 이동 비용과 공유 비용을 증가시킨다.

이러한 비용을 줄이기 위해 KV cache의 sequence length 자체를 줄이는 방법, 특히 pruning 기반 방법이 널리 연구되었다 [3, 4, 5, 6]. 그러나 pruning은 제거된 구간의 attention mass가 남은 KV entry로 재분배되어, 답변 정확도 하락을 유발할 수 있다 [7].

본 연구는 이러한 한계를 완화하기 위해 **SurrogateKV**를 제안한다. SurrogateKV는 중요한 chunk는 원본 KV로 유지하고, 중요도가 낮은 chunk는 attention이 참조할 수 있는 1개의 *surrogate token*으로 대체한다. 즉, SurrogateKV는 KV cache reduction을 단순한 keep-or-drop 문제가 아닌, 원본 KV와 surrogate 사이의 표현 자원 배분 문제로 재정의한다.

2. 연구 배경 및 관련연구

KV cache compression은 entry당 bit-width를 줄이는 quantization과, 유지 또는 참조되는 token-level KV entry 수를 줄이는 token reduction의 두 축에서 이해할 수 있다. KVQuant [8]와 같은 quantization 기반 연구는 KV cache를 low-bit로 저장하여 entry 별 메모리 사용량을 줄인다. 반면 본 연구는 decode단계에서 attention연산의 대상이되는 KV entry 수를 줄여 유효 cache 크기를 축소하는 token reduction에 초점을 둔다.

Token reduction 연구 중 pruning 기법은 전체 prefix KV 중 일부 entry만 선택해 제한된 KV budget 내에서 attention set을 구성하는 방식이다 [3, 4, 5, 6]. H_2O [3]와 StreamingLLM [4]은 heavy-hitter, recent, sink token과 같은 구조적 패턴을 활용하며, SnapKV [5]와 PyramidKV [6]는 attention pattern을 바탕으로 중요 token 을 추정하거나 layer별 budget을 결정한다. 이는 워커 간 KV cache 전송량과 decode 단계의 attention 연산량을 함께 줄일 수 있지만, 어떤 entry를 보존하는지가 성능을 좌우하기 때문에 victim selection 정책에 전체 성능이 의존하는 문제가 있다. 또한 이 과정에서 제거된 구간은 attention 대상에서 완전히 제외되므로, 남은 KV entry로 attention mass가 재분배되는 attention redistribution 문제가 발생한다. 이로 인한 답변 품질 및 정확도 하락 문제가 발견된다 [7].

따라서 본 연구는 제거될 구간을 단순 삭제하지 않고, attention이 참조 가능한 표현으로 대체하여, attention mass의 과도한 재분배를 완화하고 압축된 문맥 정보를 보존하고자 한다.

3. SurrogateKV

본 연구에서 제안하는 SurrogateKV는 KV cache를 압축 대상 과거 구간과 recent suffix로 나누어 처리한다. 그림 1과 같이 과거 chunk 중 중요도가 높은 chunk는 원본 KV로 보존하고, 중요도가 낮은 victim chunk는 하나의 surrogate pair로 대체한다. 따라서 decode step의 query는 kept chunk, surrogate KV, recent suffix KV로 구성된 압축 cache를 참조한다.

*본 연구는 2026년 과학기술정보통신부 및 정보통신기획평가원의 AI중심대학 사업 지원을 받아 수행되었음(2026-0-00036).
†Corresponding author.

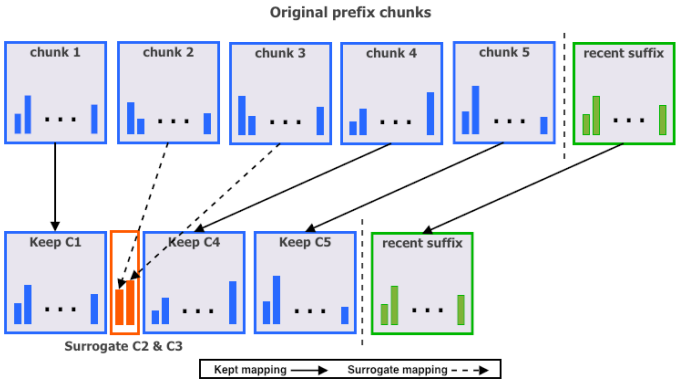

그림 1: Overview of SurrogateKV.

SurrogateKV는 먼저 과거 구간을 $C_1, \ldots, C_M$의 chunk로 나누고, recent suffix의 token position 집합 $\mathcal{R}$이 과거 token에 부여한 attention mass를 중요도 신호로 사용한다 [3, 5]. 여기서 $a_{h,r,t}$는 head h에서 recent suffix position r의 query가 과거 token t에 부여한 attention weight이다. Token 중요도 s_t는 recent suffix query들의 attention을 합산하고, token 축 local pooling을 적용한 뒤 head 평균을 취한다:

$$s_t = \frac{1}{H}\sum_{h=1}^{H}\frac{1}{|\mathcal{N}(t)|}\sum_{\tau\in\mathcal{N}(t)}\sum_{r\in\mathcal{R}} a_{h,r,\tau}. \tag{1}$$

여기서 $\mathcal{N}(t)$는 token t 주변의 local pooling window를 의미한다.

다음으로 token score를 chunk 단위로 모아 chunk 중요도 u_i를 정의한다:

$$u_i = \frac{1}{|C_i|}\sum_{t\in C_i} s_t. \tag{2}$$

Token reduction 과정에서는 u_i가 낮은 chunk부터 victim set $\mathcal{V}$에 추가한다. Target reduction rate를 ρ라 할 때, chunk C_i를 하나의 surrogate token으로 대체하면 $|C_i|-1$개의 KV entry가 줄어든다. 따라서 다음 조건을 만족하면 victim selection을 종료한다:

$$\sum_{i\in\mathcal{V}}(|C_i|-1) \geq \left\lceil \rho \sum_{i=1}^{M}|C_i| \right\rceil. \tag{3}$$

마지막으로 선택된 victim chunk의 원본 KV를 하나의 surrogate pair $(\hat{k}_i, \hat{v}_i)$로 치환한다. Surrogate construction은 surrogate pair를 구성하는 방식에 따라 다음 세 가지로 구분된다:

$$(\hat{k}_i, \hat{v}_i) = \begin{cases} (\mathbf{0}, \mathbf{0}), & \textsc{Null}, \\ (\mu_K(C_i),\ \mu_V(C_i)), & \textsc{Local}, \\ (\mu_K(\bigcup_{j\in\mathcal{V}} C_j),\ \mu_V(\bigcup_{j\in\mathcal{V}} C_j)), & \textsc{Global}. \end{cases} \tag{4}$$

$$\text{where } \mu_K(S) = \frac{1}{|S|}\sum_{t\in S} k_t, \quad \mu_V(S) = \frac{1}{|S|}\sum_{t\in S} v_t.$$

Null은 정보가 없는 zero surrogate를 사용한다. Local은 각 victim chunk의 KV 평균으로 chunk별 surrogate를 생성한다. Global은 모든 victim chunk에 대해 동일한 공통 surrogate pair를 사용한다.

표 1: Experimental server configuration.

CPU	Intel Xeon Silver 4410Y 24 Cores, 2 Sockets
Memory	128 GB DRAM
GPU	NVIDIA RTX 4000 Ada Generation 20 GB VRAM
OS	Ubuntu 22.04.4 LTS
Kernel	Linux 5.15.0-102-generic

표 2: Workload configuration.

Workload	Task Type	#Cases	Avg. Prompt Length
Qasper	Long-document QA	30	4.79k
GovReport	Long-form Summarization	30	6.81k
Needle-in-a-Haystack	Synthetic Retrieval	30	4.84k

4. 실험 및 평가

4.1 실험 환경

모든 실험은 `Meta-Llama-3-8B-Instruct`를 대상으로 수행하였으며, baseline은 공개 구현 및 NVIDIA KVPress [9]를 이용했다. SurrogateKV의 구현은 KVCache-Factory [10]를 기반으로, victim selection, surrogate construction, compressed cache assembly를 구현하였다. 실험 환경은 (표 1)과 같다.

평가에는 장문 문서 질의응답 *Qasper*, 장문 요약 *GovReport*, synthetic long-context retrieval 벤치마크 *Needle-in-a-Haystack*을 사용하였다 (표 2). 워크로드별 score 스케일이 다르므로, 각 workload에서 *FullKV* 성능을 100으로 정규화한 상대 점수 $100 \times \text{Score}(m)/\text{Score}(\textit{FullKV})$를 사용하였다.

Remaining KV cache는 *FullKV* 대비 압축 후 실제 attention 대상에 남는 KV entry 비율이며, SurrogateKV의 경우 보존된 KV와 surrogate KV를 모두 포함한다. 하이퍼파라미터로는 recent suffix 크기 8, chunk 크기 32를 사용하였다.

4.2 실험 결과 및 분석

먼저 surrogate method에 따른 차이를 비교한다. 그림 2에서 *SurKV-Global*과 *SurKV-Local*은 대부분의 KV budget 구간에서 유사한 점수 분포를 보인다. 반면 *SurKV-Null*은 전반적으로 낮은 점수를 보여, victim chunk를 정보 없는 zero surrogate로 두는 것보다, 의미 정보를 포함한 surrogate representation으로 남기는 전략이 효과적임을 확인할 수 있다.

TTFT는 *SurKV-Global*이 0.463–0.473 s 수준으로 유지되며, *FullKV* 대비 최대 증가 폭도 22.9 ms에 그쳤다. 이는 surrogate construction의 추가 비용이 있으나, chunk 단위 압축 이후 실제 attention 대상이 줄어 전체 TTFT 증가가 제한적임을 보여준다. *SurKV-Local*은 chunk별 surrogate를 생성하지만, 연산 비용이 크지 않아 *SurKV-Global*과 유사한 TTFT 범위에 머문다. 따라서 이후 비교에서는 정확도와 runtime 단순성의 균형이 우수한 *SurKV-Global*을 대표 method로 사용한다.

다음으로 *SurKV-Global*과 기존 pruning 기반 접근법들을 비교한다. 그림 3에서 *SurKV-Global*은 대부분의 KV budget 구간에서 기존 접근법보다 높은 정규화 점수 분포를 유지한다. 특히 25% remaining cache에서 *SurKV-Global*이 96.06점을 기록하여, PyramidKV의 86.33점보다 9.73점 높았다.

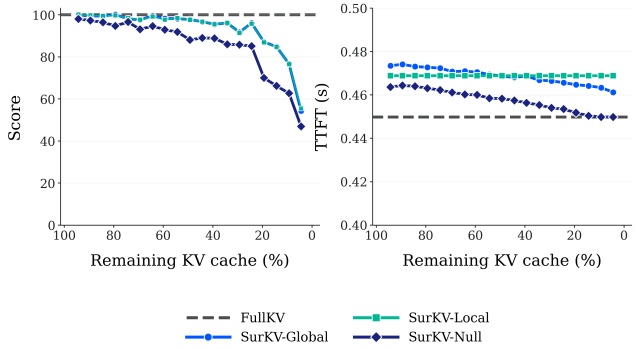

그림 2: Score and TTFT comparison across surrogate methods.

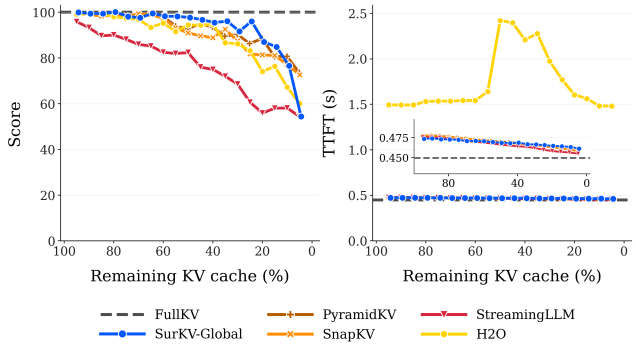

그림 3: Score and TTFT comparison between SurKV-Global and baselines.

Pruning 기반 접근법은 cache budget 밖의 KV를 제거하기 때문에, pruning 범위가 커질수록 문서 전역의 요약적 맥락이나 후속 attention 경로를 잃기 쉽다. 반면 SurrogateKV는 victim chunk를 surrogate token으로 대체하여, 제거 구간의 압축된 문맥 정보를 attention 대상에 남긴다. 이 결과는 대부분의 reduction 구간에서 SurrogateKV가 성능 저하를 효과적으로 완화함을 보여준다. TTFT 역시 0.463–0.473 s 수준으로 유지되어, PyramidKV, SnapKV, StreamingLLM과 유사한 latency band에서 더 높은 score–budget trade-off를 보였다.

다만 특정 evidence token을 찾아야 하는 retrieval-heavy 설정에서는 극단적인 pruning 비율에서 성능 이득이 제한된다. 그림 3에서 5–10%의 적은 KV budget 구간에서는 Needle workload의 점수 하락으로 인해 일부 baseline보다 낮은 점수를 보인다. 이는 surrogate token이 제거 구간의 평균적 표현은 제공하지만, 정답을 직접 담고 있는 원본 retrieval evidence 자체를 보존하기는 어렵기 때문이다. 따라서 SurrogateKV의 장점은 극단적 budget 상황에서의 최적 성능보다, 넓은 budget 범위에서 성능 저하를 완만하게 유지하는 데 있음을 확인할 수 있다.

마지막으로, surrogate representation 간 cosine similarity를 통해 Global과 Local 변형의 표현적 특성을 분석한다. 그림 4에서 *SurKV-Global*은 프롬프트가 달라져도 높은 유사도를 유지했으며(K 0.981, V 0.867), 특히 key에서 거의 동일한 방향성을 보였다. 반면 *SurKV-Local*은 더 낮은 same-layer 유사도(K 0.787, V 0.692)를 보여, 지역별로 세분화된 표현을 형성함을 나타낸다. 그럼에도 Global-Local 간 direct alignment는 비교적 높았다(K

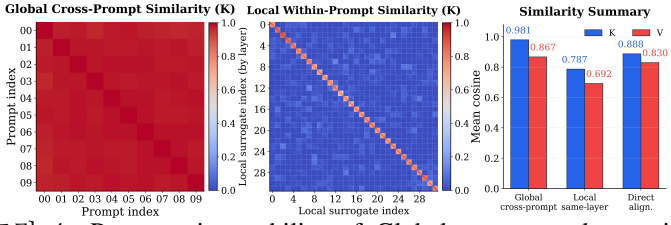

그림 4: Prompt-wise stability of Global surrogate, layer-wise structure of Local surrogate, and their direct alignment.

0.888, V 0.830). 이는 두 변형이 서로 다른 구성 단위에도, attention routing에 중요한 key-space 구조를 공유하며, *SurKV-Global*이 유사한 정확도를 유지하면서도 더 낮고 안정적인 TTFT를 보이는 결과를 설명한다. 또한 key가 value보다 일관되게 높은 유사도를 보여, 향후 K/V의 비대칭적 설계 가능성을 시사한다.

5. 결론

본 연구는 낮은 중요도의 KV chunk를 단순 제거하지 않고 surrogate token으로 대체하는 *SurrogateKV*를 제안하였다. *SurrogateKV*는 중요한 chunk와 recent suffix는 원본 KV로 유지하고, victim chunk는 attention이 참조 가능한 surrogate token으로 대체해 pruning으로 인한 attention redistribution과 정보 손실을 완화한다. 실험 결과, *SurKV-Global*은 대부분 KV budget 구간에서 기존 pruning 기반 접근법보다 높은 score–budget trade-off를 보였으며, 25% remaining cache에서 PyramidKV 대비 9.73점 높은 점수를 달성하였다. TTFT는 0.463–0.473 s 범위를 유지하며, FullKV 대비 최대 증가 폭은 22.9 ms에 그쳤다. 이는 KV cache compression을 단순한 삭제 문제가 아니라, 원본 KV와 surrogate 사이의 표현 자원 배분 문제로 다룰 수 있음을 보여준다.

참고 문헌

[1] Y. Zhong *et al.*, "DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.

[2] Y. Liu *et al.*, "LMCache: An efficient KV cache layer for enterprise-scale LLM inference," *arXiv preprint arXiv:2510.09665*, 2025.

[3] Z. Zhang *et al.*, "H_2O: Heavy-hitter oracle for efficient generative inference of large language models," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[4] G. Xiao *et al.*, "Efficient streaming language models with attention sinks," in *International Conference on Learning Representations (ICLR)*, 2024.

[5] Y. Li *et al.*, "SnapKV: LLM knows what you are looking for before generation," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[6] Z. Cai *et al.*, "PyramidKV: Dynamic KV cache compression based on pyramidal information funneling," in *Conference on Language Modeling (COLM)*, 2025.

[7] Y. Zhang *et al.*, "CaM: Cache merging for memory-efficient LLMs inference," in *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[8] C. Hooper *et al.*, "KVQuant: Towards 10 million context length LLM inference with KV cache quantization," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[9] NVIDIA, "KVPress: LLM KV cache compression made easy." https://github.com/NVIDIA/kvpress, 2025.

[10] Z. Cai, "KVCache-Factory: Unified KV cache compression methods for auto-regressive models." https://github.com/Zefan-Cai/KVCache-Factory, 2024.