Towards Efficient Large Language Model Serving: A Survey on System-Aware KV Cache Optimization

Jiantong Jiang[1], Peiyu Yang[1], Rui Zhang[2], Feng Liu[1]

[1]The University of Melbourne, [2]Huazhong University of Science and Technology (www.ruizhang.info)
{jiantong.jiang, peiyu.yang}@unimelb.edu.au
rayteam@yeah.net, fengliu.ml@gmail.com

https://github.com/jjiantong/Awesome-KV-Cache-Optimization

Abstract

Despite the rapid advancements of large language models (LLMs), LLM serving systems remain memory-intensive and costly. The key-value (KV) cache, which stores KV tensors during autoregressive decoding, is crucial for enabling low-latency, high-throughput LLM inference serving. In this survey, we focus on system-aware KV infrastructure for serving LLMs (abbreviated as *sKis*). We revisit recent work from a system behavior perspective, organizing existing efforts into three dimensions: execution and scheduling (temporal), placement and migration (spatial), and representation and retention (structural). Furthermore, we analyze cross-behavior co-design affinity and behavior-objective links, highlighting future opportunities. Our work systematizes a rapidly evolving area, providing a foundation for understanding and innovating KV cache designs in modern LLM serving infrastructure.

1 Introduction

Large language models (LLMs) have showcased exceptional abilities across diverse applications (Zhao et al., 2023), with notable examples like GPT (Radford et al., 2018, 2019; Brown et al., 2020; OpenAI, 2023), LLaMA (Touvron et al., 2023a,b), and OPT (Zhang et al., 2022). These models excel at large-scale high-quality language understanding and generation, powered by the Transformer architecture (Vaswani et al., 2017), which efficiently captures long-range dependencies via self-attention.

Despite their success, serving LLMs efficiently remains non-trivial (Li et al., 2024a). Transformer-based LLMs generate tokens autoregressively, with each token conditioned on all previous ones. To avoid redundant compute, serving systems adopt a *key-value (KV) cache* (Pope et al., 2023) to store intermediate KV tensors of the generated tokens. Yet, as prompt and output length grow, the KV cache can reach millions of tokens (Ding et al., 2024), creating memory bottlenecks and highlighting the critical role of KV cache optimization. Thus, a growing body of KV-centric techniques has emerged, yielding memory savings and efficiency gains in throughput and latency (Li et al., 2024b).

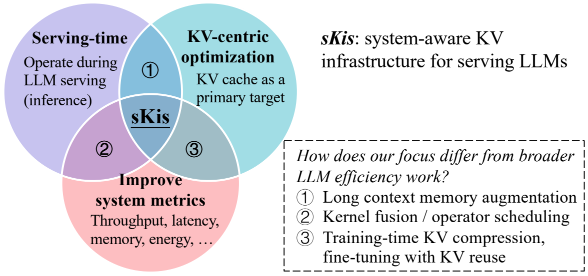

Figure 1: Positioning of the survey scope ("sKis").

To this end, we argue that it deserves a deep investigation of system-aware, serving-time, KV-centric optimization methods, as shown in Fig. 1, which we call this scope *sKis*. We adopt a system-oriented taxonomy to offer a comprehensive understanding of sKis, categorizing methods along three fundamental axes of system behaviors, as shown in Fig. 2: (i) **execution and scheduling** focuses on the *temporal* control of when KV data is accessed, computed, or scheduled (cf. § 3); (ii) **placement and migration** captures the *spatial* decisions of where KV data is placed or moved across memory tiers or devices (cf. § 4); and (iii) **representation and retention** concerns the *structural* treatment of how KV data is compressed or managed (cf. § 5). We further analyze cross-behavior co-design patterns and behavior–objective effects to reveal overlooked regions and open challenges (cf. § 6).

While prior surveys span efficient LLM inference and serving (Zhou et al., 2024; Yuan et al., 2024; Miao et al., 2023; Li et al., 2024a; Zhen et al., 2025), they are general surveys where the KV cache is discussed only as a minor component. KV-specific surveys are closest to our topic (Shi et al., 2024; Li et al., 2024b; Liu et al., 2025c), but they typically organize by lifecycle stages or

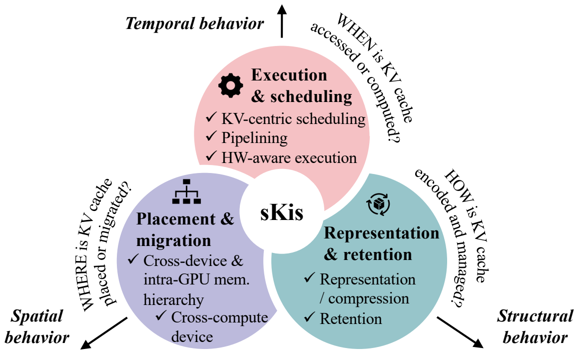

Figure 2: Taxonomy of the survey that covers temporal, spatial, and structural dimensions.

Table 1: Comparison of our work with surveys related to efficient LLM inference or serving.

Survey	KV-centric	Serving only	No retrain	Organizing principle
Miao et al. (2023)		✓		Algorithm-system
Yuan et al. (2024)		✓		Optimization layer
Li et al. (2024a)		✓	✓	System component
Zhou et al. (2024)		✓		Optimization layer
Zhen et al. (2025)		✓	✓	Serving scale
Shi et al. (2024)	✓			Lifecycle stage
Li et al. (2024b)	✓	✓		Optimization layer
Liu et al. (2025c)	✓	✓		Compression types
This survey (sKis)	✓	✓		System behavior

optimization layers. Instead, this survey focuses exclusively on sKis and distinguishes itself by offering a novel behavior-oriented perspective and a deeper understanding. We compare related surveys in Tab. 1 and provide further details in App. C.

To the best of our knowledge, we are the first to frame KV cache optimization as a temporal-spatial-structural behavior space, enabling principled analysis and actionable future directions. Because this design space is decoupled from model and kernel details, it also offers a stable lens for situating new techniques in this rapidly evolving area.

2 Foundations and Taxonomy

LLM Inference and KV Cache. LLMs generate tokens autoregressively, as shown in Fig. 3 (see preliminaries on Transformer-based LLMs in App. A). At each step, the model consumes the input and previously generated tokens to generate the next token. This process has two phases: *prefill* processes the initial input and generates the first output token, and *decode* generates tokens autoregressively. Due to the quadratic cost of self-attention, repeatedly computing attention across tokens is expensive. To this end, *key-value (KV) cache* is used to store the intermediate KV tensors computed previously, allowing the model to efficiently reuse them without

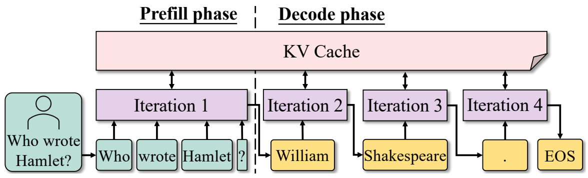

Figure 3: Prefill and decode phases of LLM inference.

recomputing attention over the entire sequence.

Scope and Taxonomy. This survey investigates recent advances in the sKis scope shown in Fig. 1.

> *sKis denotes system-aware KV infrastructure for serving LLMs. A method belongs to sKis if it: (i) operates during serving (inference), (ii) centers on KV caches as the primary optimization target, and (iii) aims to improve system metrics without retraining the base LLM's weights or modifying its Transformer architecture.*

This survey organizes literature on sKis by low-level system behaviors, as shown in Fig. 2. We offer further details in App. B. However, similar to how a modern OS includes components for scheduling, memory, and I/O, LLM serving systems often involve techniques spanning various aspects. Thus, a single paper may naturally touch on several categories. For clarity and focus, we mention 1-2 primary categories per work based on its main contributions. Minor associations are not elaborated, and we refer to App. D for details. We summarize the methods in Fig. 4 and the findings in App. E.

3 KV Execution and Scheduling

This section captures the temporal behaviors of KV cache usage, including how cache entries are scheduled and executed efficiently at runtime.

3.1 KV-centric Scheduling

While scheduling is a long-studied system problem, KV-centric scheduling (*KVS*) methods explicitly integrate KV characteristics into runtime decisions.

At the **request level**, some methods adopt KV usage-aware scheduling to balance resource load and reduce contention (Hu et al., 2024b; Duan et al., 2024; Xiong et al., 2024; Shahout et al., 2024; Wu et al., 2024). For example, TetriInfer (Hu et al., 2024b) prioritizes requests using predicted KV usage to mitigate prefill-decode interference. Another line is reuse-aware, prioritizing high-reuse requests to maximize KV cache hit rate (Zheng et al., 2024) or using KV reuse potential as a key signal in decisions (Srivatsa et al., 2024; Qin et al., 2024).

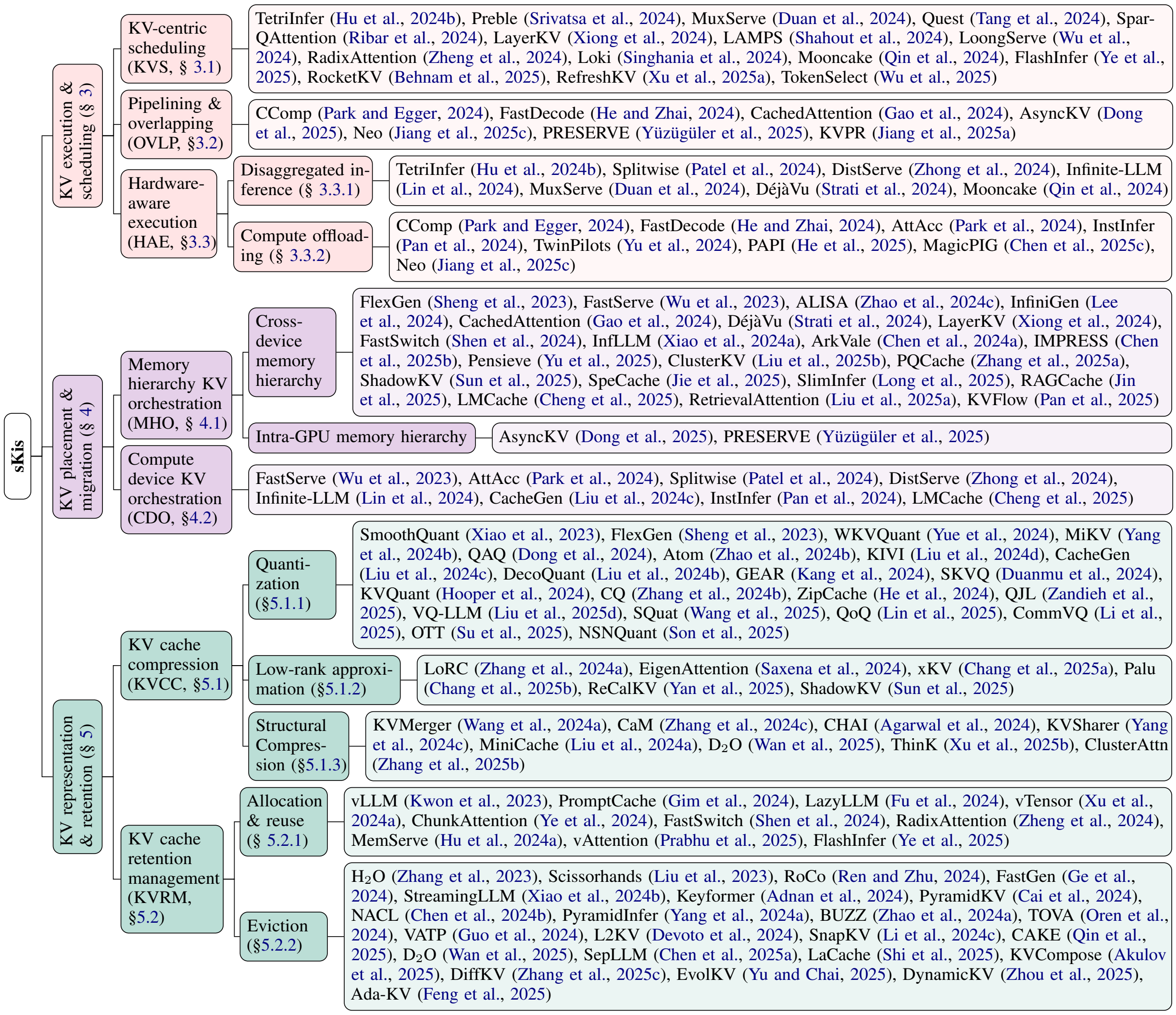

Figure 4: Taxonomy of sKis and associated methods. Each method is annotated with its primary contributions for conciseness. Minor category associations are omitted here and listed in App. D, Tab. 9.

At finer granularity, **token-level** methods decide which KV entries participate in attention based on estimated contributions (Tang et al., 2024; Ribar et al., 2024; Singhania et al., 2024; Behnam et al., 2025; Xu et al., 2025a; Wu et al., 2025), for example via periodic refresh that alternates full-context and subset attention (Xu et al., 2025a). At the **kernel level**, methods like FlashInfer (Ye et al., 2025) schedule attention workloads across CUDA thread blocks based on query and KV lengths.

3.2 Pipelining and Overlapping

Pipelining and overlapping (*OVLP*) methods hide KV-related latency by overlapping compute, I/O, and communication. Though often embedded in broader systems, Tab. 2 highlights methods where OVLP forms the core technical contribution. We summarize them by mode and list the corresponding overlapped operations and granularity. OVLP is key to reducing idle time and improving efficiency.

3.3 Hardware-aware Execution

This section focuses on hardware-aware execution (*HAE*) methods that adapt KV-related operations to the underlying heterogeneous hardware.

3.3.1 Disaggregated Inference

Disaggregated inference decouples inference compute onto distinct hardware resources to reduce contention and improve utilization. Infinite-LLM (Lin et al., 2024) adopts this idea at the operator level by splitting attention across distributed instances. Several systems instead apply prefill-decode (PD) disaggregation, assigning compute-bound prefill and memory-bound decode to different compute pools (Hu et al., 2024b; Patel et al., 2024; Zhong et al., 2024; Strati et al., 2024). Mooncake (Qin et al., 2024) further couples PD disaggregation with a KV-centric scheduler and distributed cache pool, while MuxServe (Duan et al., 2024) colocates PD jobs within each GPU through SM partitioning.

Table 2: Summary of OVLP methods. "Comp" denotes compute, "I/O" denotes KV data movement (host–device transfer or on-device memory movement), and "comm" denotes collective communication.

Mode	Method	Overlapped operations (with transfer path)	Granularity
Comp–Comp	FastDecode (He and Zhai, 2024)	CPU R-part comp ↔ GPU S-part comp	Token-wise
	Neo (Jiang et al., 2025c)	CPU attention comp ↔ GPU linear ops	Sub-batch-wise
Comp–I/O	CComp (Park and Egger, 2024)	CPU MHSA comp ↔ FFN data transfer (CPU→GPU)	Split point
	CachedAttention (Gao et al., 2024)	GPU comp ↔ KV load/store (CPU↔GPU)	Layer-wise
	AsyncKV (Dong et al., 2025)	GPU attention comp ↔ GPU KV prefetch (HBM→L2)	KV block-wise
	KVPR (Jiang et al., 2025a)	GPU KV re-comp ↔ KV transfer (CPU↔GPU)	Split point
I/O–Comm	PRESERVE (Yüzügüler et al., 2025)	GPU KV prefetch (HBM→L2) ↔ GPU collective comm	Operator-wise

3.3.2 Compute Offloading

Compute offloading relocates partial compute to auxiliary devices to reduce GPU bottlenecks, utilizing hardware heterogeneity and workload features.

A practical instantiation is **CPU offloading**, which leverages host CPUs for memory-intensive compute (He and Zhai, 2024; Park and Egger, 2024; Chen et al., 2025c; Jiang et al., 2025c). They often follow a compute-near-cache principle for better locality. For example, FastDecode (He and Zhai, 2024) and Neo (Jiang et al., 2025c) offload both attention and KV caches, using cost-aware hardware selection and a load-aware scheduler, respectively.

Beyond CPUs, several methods offload compute to **alternative devices**, such as computational storage drive (CSD) (Pan et al., 2024) and processing-in-memory (PIM) (He et al., 2025; Park et al., 2024). These methods expand the compute offloading space to broader device heterogeneity.

Takeaways & Limitations – Spatial Behavior

- KVS and OVLP directly target KV reuse and stall hiding. Lightweight cost models or predictors often enhance them. However, they are typically evaluated on controlled workloads, with limited analysis of robustness under bursty traffic or multi-tenant settings.
- HAE improves throughput and hardware utilization by decoupling compute and specializing kernels, but its reliance on low-level primitives can make portability non-trivial for practitioners in some cases.

More analysis is provided in Apps. E.1 and E.2.

4 KV Placement and Migration

This section focuses on the spatial behaviors of how KV caches are placed and migrated across memory hierarchies and between compute devices. Figure 5 visualizes the architecture and transfer paths.

4.1 Memory Hierarchy KV Orchestration

To scale under memory limits, we survey memory hierarchy KV orchestration (*MHO*) methods that distribute KV caches across memory hierarchies.

Cross-device Memory Hierarchy. A broad range of methods migrate KV entries across faster but

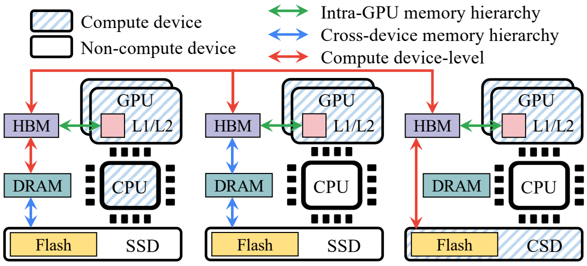

Figure 5: Illustration of KV cache placement and migration across memory hierarchies and compute devices.

limited GPU HBM, and larger but slower alternatives like CPU DRAM or SSD. Most works are **importance-aware**, designing importance scoring policies that maintain only critical KV entries on GPU (Zhao et al., 2024c; Lee et al., 2024; Xiao et al., 2024a; Chen et al., 2024a, 2025b; Yu et al., 2025; Liu et al., 2025b; Zhang et al., 2025a; Sun et al., 2025; Jie et al., 2025; Long et al., 2025; Liu et al., 2025a). For instance, ArkVale (Chen et al., 2024a), ClusterKV (Liu et al., 2025b), PQ-Cache (Zhang et al., 2025a), and SpeCache (Jie et al., 2025) offload full KV cache to CPU and keep only a compressed or summarized proxy on GPU. They then estimate importance via proxies to guide the next prefetch. Another line of cross-device methods optimizes KV placement and migration from a **system cost** view. FlexGen (Sheng et al., 2023) places KV caches across GPU, CPU, and disk via a cost model that maximizes throughput under bandwidth and latency constraints. At runtime, many systems make online decisions about KV offloading or reloading based on system-level signals, such as queueing state, memory pressure, compute and I/O costs, and future reuse signals (Wu et al., 2023; Gao et al., 2024; Strati et al., 2024; Xiong et al., 2024; Shen et al., 2024; Jin et al., 2025; Cheng et al., 2025; Pan et al., 2025).

Intra-GPU Memory Hierarchy. Another line of MHO methods migrates KV entries between on-chip L1/L2 caches and off-chip HBM. Dong

Table 3: Summary of KV cache quantization (q.) methods. "Avg. bits" shows the average bitwidth per KV element based on the reported main results. This metric indicates memory savings and is comparable across methods.

Method	Granularity Keys	Granularity Values	Prec. mode	Important region	Outlier handling	Avg. bits
SmoothQuant (Xiao et al., 2023)	Channel-wise		Fixed	–	Smoothing via scaling	8
FlexGen (Sheng et al., 2023)	Group-wise		Fixed	–	–	4
WKVQuant (Yue et al., 2024)	2D (channel & token)		Mixed	Current token	Dynamic token-wise q.	~4
MiKV (Yang et al., 2024b)	Token-wise		Mixed	Existing policy	Outlier balancing	~4
QAQ (Dong et al., 2024)	Token-wise		Mixed	Attention-aware	Sparse matrix (FP16)	1.8-2.7
Atom (Zhao et al., 2024b)	Group-wise		Mixed	Outlier channels	Selective high-bits	4.25
KIVI (Liu et al., 2024d)	Channel-wise	Token-wise	Mixed	Recent tokens	Channel-wise confining	~2
CacheGen (Liu et al., 2024c)	Layer-wise		Mixed	Shallow layers	–	1.9-2.9
DecoQuant (Liu et al., 2024b)	Decomposed-tensor-wise		Mixed	Small tensors	Tensor decomposition	4
GEAR (Kang et al., 2024)	Channel-wise	Token-wise	Fixed	–	Sparse matrix (FP16)	4.4/5.0
SKVQ (Duanmu et al., 2024)	Group-wise		Mixed	Init. & recent tokens	Clipped dynamic q.	2.25
KVQuant (Hooper et al., 2024)	Channel-wise	Token-wise	Mixed	First token	Sparse matrix (FP16)	4.3
CQ (Zhang et al., 2024b)	Token-wise channel-group		Fixed	–	–	1.3
ZipCache (He et al., 2024)	Channel-wise	Chan.-sep. token-wise	Mixed	Norm. attention	Channel-wise norm.	3.2
QJL (Zandieh et al., 2025)	Token-wise		Fixed	–	Selective high-bits	3/5
VQ-LLM (Liu et al., 2025d)	Group-wise (configurable)		Fixed	–	–	2/4
SQuat (Wang et al., 2025)	Block-wise	Token-wise	Fixed	–	–	3.1
QoQ (Lin et al., 2025)	Channel-wise		Fixed	–	Smooth attention	4
CommVQ (Li et al., 2025)	Token-wise vector		Fixed	–	–	2
OTT (Su et al., 2025)	Channel-wise	Token-wise	Mixed	Outlier & recent tokens	Full precision	2.5
NSNQuant (Son et al., 2025)	Token-wise vector		Fixed	–	Token-wise norm.	1.2/2.2

et al. (2025) asynchronously prefetched upcoming KV blocks from HBM into L2 so that subsequent attention steps mostly hit in L2. Similarly, PRESERVE (Yüzügüler et al., 2025) fetches KV caches and inserts such operations selectively via graph-level optimization to avoid cache pollution.

4.2 Compute Device KV Orchestration

Unlike hierarchical memory, compute device KV orchestration (*CDO*) places and moves KV across compute-capable devices to enable distributed or heterogeneous serving. A common line performs intra-cluster orchestration, typically coupled with PD disaggregation (Wu et al., 2023; Patel et al., 2024; Zhong et al., 2024; Lin et al., 2024; Cheng et al., 2025). For instance, DistServe (Zhong et al., 2024) proposes placement schemes for prefill and decode across high and low node-affinity GPU clusters, and uses a pull-based scheme where decode GPUs fetch KV as needed from prefill GPUs.

Beyond tightly coupled clusters, CacheGen (Liu et al., 2024c) targets remote KV transfer in network setups. It reduces network delay by encoding KV tensors into bitstreams and adaptively streaming them based on runtime bandwidth. Finally, CDO also extends to heterogeneous accelerators, offloading attention to devices such as PIMs and CSDs (Park et al., 2024; Pan et al., 2024).

> **Takeaways & Limitations – Spatial Behavior**
> MHO and CDO directly target interconnect bottlenecks through tiered KV management, and most systems overlap KV transfers with compute to hide latency, which is a central factor in their effectiveness. However, bandwidth is typically handled without explicit modeling contention among concurrent KV transfers, making tail behavior hard to analyze. Another gap is the explicit joint optimization of offload and prefetch under shared memory and interconnect budgets. More takeaways are provided in App. E.3.

5 KV Representation and Retention

This section focuses on structural system behaviors of KV cache representation and retention.

5.1 KV Cache Compression

KV cache compression (*KVCC*) is a central research thrust as it directly reduces memory usage.

5.1.1 KV Cache Quantization

Quantization compresses floating-point tensors into lower-precision formats. Early works enable 8- and 4-bit KV (Xiao et al., 2023; Sheng et al., 2023). Later schemes adopt mixed precision, assigning high precision to critical KV entries. We compare methods along core algorithmic axes and effective bitwidths in Tab. 3 and present key insights here.

One recurring pattern is **asymmetric KV quantization** (cf. "granularity" in Tab. 3), as keys and values exhibit distinct outlier patterns and quantization sensitivities. For example, a common practice is to quantize keys per-channel and values per-token. A second insight is that **outliers** play a crucial role in low-bit quantization, so many methods store them in higher bitwidths or design dedicated outlier handling techniques (cf. "outlier handling" in Tab. 3).

Recent advances use **vector quantization (VQ)** to compress groups with codebooks and capture

Table 4: Summary of low-rank approximation methods.

Target	Method	Granularity	Rank
Cached KV tensors	xKV (Chang et al., 2025a)	LG	F
	ReCalKV (Yan et al., 2025)	K: HG; V: L	B
	ShadowKV (Sun et al., 2025)	L (K-only)	F
W^K,W^V	LoRC (Zhang et al., 2024a)	L	R
	Palu (Chang et al., 2025b)	HG	S
QKV	EigenAttention (Saxena et al., 2024)	L	B

Granularity: L = layer-wise; LG = layer-group-wise; HG = head-group-wise.
Rank policy: F fixed; S searched; B budget-driven; R rule-based.

Table 5: Summary of structural compression methods.

Family	Method	Unit	Signal
Merging	KVMerger (Wang et al., 2024a)	Token	A S
	CaM (Zhang et al., 2024c)	Token	A
	KVSharer (Yang et al., 2024c)	Layer	S
	MiniCache (Liu et al., 2024a)	Layer	S
	D2O (Wan et al., 2025)	Token	S
Pruning	CHAI (Agarwal et al., 2024)	Head	S
	ThinK (Xu et al., 2025b)	Channel	Q
	ClusterAttn (Zhang et al., 2025b)	Token	A

Signal: A attention score; S similarity/dissimilarity; Q query norm.

inter-element correlation. CQ (Zhang et al., 2024b) couples channels and learns centroids for 1-bit KV. VQ-LLM (Liu et al., 2025d) and CommVQ (Li et al., 2025) reduce overhead via fused VQ kernels and RoPE-commutative codebooks. Son et al. (2025) further improved calibration robustness.

5.1.2 KV Cache Low-rank Approximation

Low-rank methods constrain KV-related tensors to a low-dimensional subspace, as summarized in Tab. 4 by target: (i) cached KV, (ii) KV projection weights (W^K, W^V), or (iii) QKV attention subspace. For instance, xKV (Chang et al., 2025a) applies layer-group singular value decomposition to cached KV, while Palu (Chang et al., 2025b) factorizes (W^K, W^V) with searched rank allocation. KV tensor methods are most plug-and-play but add projection cost, whereas weight and QKV ones increase kernel coupling and engineering overhead. Some low-rank methods learn extra parameters, requiring training and thus out of scope under sKis.

5.1.3 KV Cache Structural Compression

Unlike value-level methods, structural compression reduces KV memory by modifying cache organization (e.g., layer, head, channel, token). We compare existing methods in Tab. 5, including (i) **pruning**, which drops a subset of structural units, and (ii) **merging**, which fuses units into shared forms. The decisions are often guided by attention or similarity measures (cf. "signal" in Tab. 5), with clustering sometimes used to form groups (Wang et al., 2024a; Agarwal et al., 2024; Zhang et al., 2025b).

5.2 KV Cache Retention Management

Going beyond representations, this section focuses on mechanisms that efficiently manage the retention of the KV cache (*KVRM*) during serving.

5.2.1 KV Cache Allocation and Reuse

Structure-aware methods redesign KV cache layouts for flexible allocation and reuse. One line targets virtualized allocation (Kwon et al., 2023; Xu et al., 2024a; Shen et al., 2024; Prabhu et al., 2025). A famous example is PagedAttention (Kwon et al., 2023), which uses fixed-size pages with logical-to-physical mapping to reduce fragmentation and support memory reuse. Another line builds structured indices for prompt sharing (Gim et al., 2024; Ye et al., 2024; Zheng et al., 2024), exemplified by radix tree (Zheng et al., 2024). A third line standardizes KV layouts for kernels; Ye et al. (2025) introduced a block-sparse and composable format.

Orthogonally, **semantics-guided** methods further reduce materialization by computing KV only for critical tokens (Fu et al., 2024) and extend reuse to disaggregated LLM serving via an elastic MemPool system (Hu et al., 2024a).

5.2.2 KV Cache Eviction

KV cache eviction reduces memory by discarding less critical token KV states under a budget. We compare algorithmic details of existing methods in Tab. 6 and highlight three key insights.

First, methods differ in when eviction is applied (cf. "mode" in Tab. 6). Static methods evict once during or after prefill and keep the retained set fixed in decoding, while dynamic ones update online during decoding to track importance shifts. Second, eviction policies often retain a recent window or attention sink tokens, and select extra tokens by lightweight signals such as attention-derived scores, heuristics, or robust variants that mitigate bias in local attention statistics (cf. "eviction policy" in Tab. 6). Third, recent works move beyond uniform budgets and instead assign budgets across layers and even heads via preset and adaptive allocation (cf. "budget policy" in Tab. 6). Some methods treat budget policy as a plug-in to existing eviction rules.

Takeaways & Limitations – Structural Behavior

- KVCC delivers the most direct memory relief, but its real bottleneck is reliable compression. Memory savings may not translate into system gains without system co-design, due to tail (e.g., outlier) behavior, compression overhead, and kernel or runtime constraints.
- KVRM improves effective capacity by deciding which KV states exist at runtime. The key challenge is fast

Table 6: Summary of representative KV cache eviction methods in chronological order.

Method	Mode	Eviction policy	Budget policy
H_2O (Zhang et al., 2023)	Dynamic	R + H_2 (via accumulated attention)	Uniform
Scissorhands (Liu et al., 2023)	Dynamic	R + Attention scores	Uniform
RoCo (Ren and Zhu, 2024)	Dynamic	Mean & std. dev. of attention scores	Uniform
FastGen (Ge et al., 2024)	Static	Hybrid (special/punctuation/locality/H_2)	Uniform
StreamingLLM (Xiao et al., 2024b)	Dynamic	R S	Uniform
Keyformer (Adnan et al., 2024)	Dynamic	R + Key (via Gumbel-softmax scores)	Uniform
PyramidKV (Cai et al., 2024)	Static	Observation window-based identification	Preset (L, pyramid)
NACL (Chen et al., 2024b)	Dynamic	Attention w.r.t. proxy tokens & randomness	Uniform
PyramidInfer (Yang et al., 2024a)	Dynamic	R + PvC (via ensemble attention)	Preset (L, pyramid)
BUZZ (Zhao et al., 2024a)	Dynamic	R S + Segmented local H_2	Uniform
TOVA (Oren et al., 2024)	Dynamic	Drop lowest attention score token at each step	Uniform
VATP (Guo et al., 2024)	Dynamic	S + Attention & value L_1-norm	Uniform
L2KV (Devoto et al., 2024)	Dynamic	Key L_2-norm	Uniform
SnapKV (Li et al., 2024c)	Static	Observation window-based identification	Uniform
CAKE (Qin et al., 2025)	Dynamic	R + Mean & var. of attention scores	Adaptive (L, layer preference)
D_2O (Wan et al., 2025)	Dynamic	R S + H_2 & recall via merging (§ 5.1.3)	Adaptive (L, attention density)
SepLLM (Chen et al., 2025a)	Dynamic	R S + Separator tokens	Uniform
LaCache (Shi et al., 2025)	Dynamic	Ladder pattern based	Preset (L, ladder)
KVCompose (Akulov et al., 2025)	Dynamic	Aggregated attention & form composite token	Adaptive (L, composite importance)
DiffKV (Zhang et al., 2025c)	Dynamic	R + Relative significance of attention scores	Adaptive (H, sparsity pattern)
EvolKV (Yu and Chai, 2025)	Dynamic	Plug-in (adopt existing eviction methods)	Adaptive (L, evolutionary search)
DynamicKV (Zhou et al., 2025)	Static	R + Attention w.r.t. instruction tokens	Adaptive (L, task-aware)
Ada-KV (Feng et al., 2025)	Dynamic	Plug-in (adopt existing eviction methods)	Adaptive (H, attention sparsity)

Eviction policy: R recent tokens; S attention sink tokens (Xiao et al., 2024b), which means initial tokens. *Budget policy:* L = layer-wise, H = head-wise.

> and stable utility estimation. In practice, policies are often workload-sensitive, and robustness under complex serving environments remains under-studied.
> More analysis is provided in Apps. E.4 and E.5.

6 Observations and Open Challenges

Here, we identify observations from two complementary lenses: (i) a behavior×objective matrix and (ii) a behavior-behavior co-design affinity network, which naturally motivate open challenges. We show the links of observations and challenges and present key directions in Fig. 7 in App. G.1.

Figure 6 (behavior-behavior co-design affinity network) visualizes cross-behavior co-occurrence in the literature, with edge thickness proportional to normalized weights (low-score edges omitted; computation details in App. F). This affinity reflects observed co-design patterns rather than validated performance gains. Table 7 (behavior × objective matrix) marks each behavior's impact on serving objectives as direct (●) or indirect (○); stars ($\star$) on direct cells statistically flag $\geq 70\%$ of papers reporting such gains. Side bars show research density. Objectives cover latency, throughput, GPU memory, interconnect I/O, and energy. We also include quality impact ↓ to capture degradation as a trade-off. Key observations are as follows.

O1. Structural works are most studied and dominate memory savings, while others yield savings indirectly (e.g., via migration or reuse), indicating a community bias toward memory efficiency.

O2. Temporal behaviors act most directly on latency and throughput, since KVS, OVLP, and HAE map cleanly to reductions in scheduling stalls, pipeline bubbles, and device under-utilization. However, tail latency reporting is sparse.

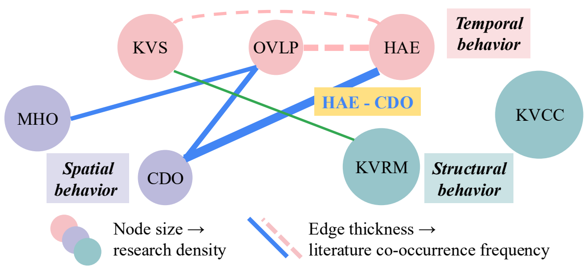

Figure 6: Behavior-behavior co-design affinity network.

O3. Spatial methods primarily target interconnect I/O, often paired with OVLP. Their core focus is KV transfer, and by overlapping it with compute, they effectively hide transfer latency.

O4. Energy is under-explored, although many methods reduce memory or compute intensity that should translate to energy benefits.

O5. Quality loss is universal. Temporal methods risk inconsistent request handling; spatial methods risk missed KV data; and structural methods directly reduce KV precision. The practical question is to ensure such degradation is controllable.

O6. HAE–CDO is the strongest co-design pattern. Compute layouts that exploit device heterogeneity often co-design with KV colocating or transfer, yielding joint gains in utilization and I/O.

O7. KVCC remains isolated despite its popularity, which suggests a missed opportunity for co-design.

Table 7: Behavior × objective matrix of sKis methods. Side bars encode research density (rows/columns). Cells mark relevance levels (●= direct, ○= indirect) and high-prevalence flags (★: ≥ 70% of papers report gains).

Behaviors	Mean latency	Tail latency	Through-put	GPU memory	Inter-connect I/O	Energy /power	Quality impact ↓	Row density
KV-centric scheduling	●★	●	●	○	○	○	○	
Pipelining and overlapping	●★	○	●★		○	○	○	
Hardware-aware execution	●★	●	●★		●	●	○	
Memory hierarchy KV orchestration	○	○	○	○	●★	●	●	
Compute device KV orchestration	○	○	●★	○	●	●	●	
KV cache compression	○	○	○	●★	●	●	●	
KV cache retention management	○	○	○	●★	●	●	●	
Column density								

Behavior dimension: temporal, spatial, structural. *Row density* 0–10 11–20 21–30 31+ *Column density* 0–20 21–40 41–60 61+

The above observations reveal both progress and gaps of current sKis research, which motivate the next set of system-level open challenges.

C1. SLO-driven tail control ← O2. Service-level objectives (SLOs) are critical to LLM serving, with tail latency dominating user experience (Dean and Barroso, 2013; Wang et al., 2024b), yet most systems omit tail metrics. Under long contexts and bursts, KV generation, migration, and compression may interfere and trigger SLO violations. The challenge is to attribute SLO violations to concrete KV behaviors and paths, motivating studies on standardized preemption and degradation semantics to make tail outcomes controllable.

C2. Energy-aware sKis ← O4. With surging data center demand, sKis should be energy-aware, but energy is rarely reported or optimized. Future research could integrate power profiling into runtime decisions, establish serving-time energy models, and jointly optimize energy-latency-quality under power constraints. Another possible direction is to study energy-friendly KV granularities and layouts.

C3. Trustworthy and efficient sKis ← O5. LLM serving must ensure not only quality but also trustworthiness (Han et al., 2025), yet trust risks are rarely considered, leaving a gap between efficiency gains and trust failures. For example, structural methods can harm robustness in ways standard metrics miss, as policies may evict or compress low-salience but crucial context, causing severe errors under workload shifts despite stable mean accuracy. Such trust concerns also span reliability, privacy, and safety across diverse sKis behaviors. Notably, sKis techniques can be dual-use: Jiang et al. (2025b) turned KV eviction into a defense against jailbreak attacks, suggesting that sKis techniques may become trust mechanisms. Future work could make trustworthiness behavior-attributed and SLO-aware. We give further discussion in App. G.2.

C4. Generalizable HAE–CDO ← O6. While HAE and CDO form the strongest co-design pattern, policies are often tailored to specific fabric or single-tenant settings. Future directions include making such pattern portable across heterogeneous topologies (e.g., NVLink, NVSwitch, PCIe, CXL) and adaptive to multi-tenant settings.

C5. Co-optimization and intermediate semantics ← O7. Most sKis optimize behaviors in isolation, despite their interactions under bandwidth and latency constraints. Future studies could explore co-optimization under shared budgets. For instance, to co-decide eviction, offload, and prefetch given predicted reuse, success probability, and I/O contention. Another promising direction is to exploit fine-grained intermediate semantics for behaviors and view co-optimization as state transitions over them. We give concrete examples in App. G.3 illustrating how intermediate semantics enable co-optimizing eviction, compression, and migration.

C6. Unified benchmarks. We review LLM inference benchmarking practices in App. G.4.1. We find inconsistent metric definitions and measures across tools, preventing reliable apples-to-apples comparisons across papers. We therefore advocate unified sKis benchmarks and offer a concise checklist of metrics (e.g., trust metrics and KV-centric resource metrics), representative stress workloads, and reporting standards, detailed in App. G.4.2.

7 Conclusion

This survey presents a systematic overview of sKis, offering a system behavior-oriented taxonomy covering temporal, spatial, and structural dimensions. By cross-analyzing behavior-objective impacts and behavior-behavior co-design patterns, we reveal overlooked regions and open challenges. We hope this survey inspires continued exploration toward efficient and trustworthy LLM serving.

Limitations

This paper offers a comprehensive review and summary of current methods in the area of system-aware KV cache optimization. However, given the extensive body of related work and the rapidly evolving nature of this research area, we may have overlooked some equally valuable contributions. We tried to include all relevant studies and references wherever feasible.

Additionally, this survey conducts no new experiments. Our claims synthesize results reported in public papers and open-source implementations, primarily under mainstream platforms and common configurations, which may constrain the generality of our conclusions. We avoid aggregating raw speedup or memory numbers across papers, because the reported gains are tightly coupled with model, hardware, workload, or baseline choices.

Finally, we outline several KV-centric research directions to improve the efficiency in LLM serving, including SLO-first tail-latency control, energy-aware sKis, trustworthy sKis, generalizable HAE-CDO, co-optimization and intermediate semantics, and unified benchmarks. We plan to leave these aspects for future work.

References

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J Nair, Ilya Soloveychik, and Purushotham Kamath. 2024. Keyformer: KV cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6:114–127.

Saurabh Agarwal, Bilge Acun, Basil Hosmer, Mostafa Elhoushi, Yejin Lee, Shivaram Venkataraman, Dimitris Papailiopoulos, and Carole-Jean Wu. 2024. CHAI: Clustered head attention for efficient LLM inference. In *International Conference on Machine Learning*, pages 291–312. PMLR.

Dmitry Akulov, Mohamed Sana, Antonio De Domenico, Tareq Si Salem, Nicola Piovesan, and Fadhel Ayed. 2025. KVCompose: Efficient structured KV cache compression with composite tokens. *arXiv preprint arXiv:2509.05165*.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and Yuxiong He. 2022. DeepSpeed-Inference: enabling efficient inference of transformer models at unprecedented scale. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE.

Artem Babenko and Victor Lempitsky. 2014. Additive quantization for extreme vector compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 931–938.

Guangji Bai, Zheng Chai, Chen Ling, Shiyu Wang, Jiaying Lu, Nan Zhang, Tingwei Shi, Ziyang Yu, Mengdan Zhu, Yifei Zhang, Carl Yang, Yue Cheng, and Liang Zhao. 2024. Beyond efficiency: A systematic survey of resource-efficient large language models. *arXiv preprint arXiv:2401.00625*.

Payman Behnam, Yaosheng Fu, Ritchie Zhao, Po-An Tsai, Zhiding Yu, and Alexey Tumanov. 2025. RocketKV: Accelerating long-context LLM inference via two-stage KV cache compression. In *International Conference on Machine Learning*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. 2024. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Chi-Chih Chang, Chien-Yu Lin, Yash Akhauri, Wei-Cheng Lin, Kai-Chiang Wu, Luis Ceze, and Mohamed S Abdelfattah. 2025a. xKV: Cross-layer SVD for KV-cache compression. *arXiv preprint arXiv:2503.18893*.

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. 2025b. Palu: KV-cache compression with low-rank projection. In *International Conference on Learning Representations*.

Guoxuan Chen, Han Shi, Jiawei Li, Yihang Gao, Xiaozhe Ren, Yimeng Chen, Xin Jiang, Zhenguo Li, Weiyang Liu, and Chao Huang. 2025a. SepLLM: Accelerate large language models by compressing one segment into one separator. In *International Conference on Machine Learning*.

Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. 2024a. ArkVale: Efficient generative LLM inference with recallable key-value eviction. *Advances in Neural Information Processing Systems*, 37:113134–113155.

Weijian Chen, Shuibing He, Haoyang Qu, Ruidong Zhang, Siling Yang, Ping Chen, Yi Zheng, Baoxing Huai, and Gang Chen. 2025b. IMPRESS: An importance-informed multi-tier prefix KV storage system for large language model inference. In *USENIX Conference on File and Storage Technologies*, pages 187–201.

Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. 2024b. NACL: A general and effective KV cache eviction framework for LLM at inference time. In *Annual Meeting of the Association for Computational Linguistics*, pages 7913–7926.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Léon Bottou, Zhihao Jia, and Beidi Chen. 2025c. MagicPIG: LSH sampling for efficient LLM generation. In *International Conference on Learning Representations*.

Yihua Cheng, Yuhan Liu, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Kuntai Du, and Junchen Jiang. 2025. LMCache: An efficient KV cache layer for enterprise-scale LLM inference. *arXiv preprint arXiv:2510.09665*.

Krishna Teja Chitty-Venkata, Siddhisanket Raskar, Bharat Kale, Farah Ferdaus, Aditya Tanikanti, Ken Raffenetti, Valerie Taylor, Murali Emani, and Venkatram Vishwanath. 2024. LLM-Inference-Bench: Inference benchmarking of large language models on AI accelerators. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1362–1379. IEEE.

Jeffrey Dean and Luiz André Barroso. 2013. The tail at scale. *Communications of the ACM*, 56(2):74–80.

Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. 2024. A simple and effective L_2 norm-based strategy for KV cache compression. In *Conference on Empirical Methods in Natural Language Processing*, pages 18476–18499.

Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. 2024. LongRoPE: Extending LLM context window beyond 2 million tokens. *arXiv preprint arXiv:2402.13753*.

Shichen Dong, Wen Cheng, Jiayu Qin, and Wei Wang. 2024. QAQ: Quality adaptive quantization for LLM KV cache. *arXiv preprint arXiv:2403.04643*.

Yanhao Dong, Yubo Miao, Weinan Li, Xiao Zheng, Chao Wang, and Feng Lyu. 2025. Accelerating LLM inference throughput via asynchronous KV cache prefetching. *arXiv preprint arXiv:2504.06319*.

Jiangfei Duan, Runyu Lu, Haojie Duanmu, Xiuhong Li, Xingcheng Zhang, Dahua Lin, Ion Stoica, and Hao Zhang. 2024. MuxServe: Flexible spatial-temporal multiplexing for multiple LLM serving. In *International Conference on Machine Learning*, pages 11905–11917. PMLR.

Haojie Duanmu, Zhihang Yuan, Xiuhong Li, Jiangfei Duan, Xingcheng Zhang, and Dahua Lin. 2024. SKVQ: Sliding-window key and value cache quantization for large language models. *arXiv preprint arXiv:2405.06219*.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. 2025. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. *Advances in Neural Information Processing Systems*.

Qichen Fu, Minsik Cho, Thomas Merth, Sachin Mehta, Mohammad Rastegari, and Mahyar Najibi. 2024. LazyLLM: Dynamic token pruning for efficient long context LLM inference. *arXiv preprint arXiv:2407.14057*.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. 2024. Cost-efficient large language model serving for multi-turn conversations with CachedAttention. In *USENIX Annual Technical Conference*, pages 111–126.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *International Conference on Learning Representations*.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. 2024. Prompt Cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338.

Robert Gray. 1984. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29.

Xiangming Gu, Tianyu Pang, Chao Du, Qian Liu, Fengzhuo Zhang, Cunxiao Du, Ye Wang, and Min Lin. 2025. When attention sink emerges in language models: An empirical view. In *International Conference on Learning Representations*.

Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. 2024. Attention score is not all you need for token importance indicator in KV cache reduction: Value also matters. In *Conference on Empirical Methods in Natural Language Processing*, pages 21158–21166.

Bo Han, Jiangchao Yao, Tongliang Liu, Bo Li, Sanmi Koyejo, and Feng Liu. 2025. Trustworthy machine learning: From data to models. *Foundations and Trends® in Privacy and Security*, 7(2-3):74–246.

Jiaao He and Jidong Zhai. 2024. FastDecode: High-throughput GPU-efficient LLM serving using heterogeneous pipelines. *arXiv preprint arXiv:2403.11421*.

Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. 2024. ZipCache: Accurate and efficient KV cache quantization with salient token identification. In *Advances in Neural Information Processing Systems*, volume 37, pages 68287–68307.

Yintao He, Haiyu Mao, Christina Giannoula, Mohammad Sadrosadati, Juan Gómez-Luna, Huawei Li, Xiaowei Li, Ying Wang, and Onur Mutlu. 2025. PAPI: Exploiting dynamic parallelism in large language model decoding with a processing-in-memory-enabled computing system. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 766–782.

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

Cunchen Hu, HeYang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, and Yizhou Shan. 2024a. MemServe: Context caching for disaggregated LLM serving with elastic memory pool. *arXiv preprint arXiv:2406.17565*.

Cunchen Hu, HeYang Huang, Liangliang Xu, Xusheng Chen, Jiang Xu, Shuang Chen, Hao Feng, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, and Yizhou Shan. 2024b. Inference without interference: Disaggregate LLM inference for mixed downstream workloads. *arXiv preprint arXiv:2401.11181*.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128.

Chaoyi Jiang, Lei Gao, Hossein Entezari Zarch, and Murali Annavaram. 2025a. KVPR: Efficient LLM inference with i/o-aware KV cache partial recomputation. In *Annual Meeting of the Association for Computational Linguistics*.

Jiantong Jiang, Zeyi Wen, Atif Mansoor, and Ajmal Mian. 2024. Fast inference for probabilistic graphical models. In *USENIX Annual Technical Conference*, pages 95–110.

Jiantong Jiang, Zeyi Wen, and Ajmal Mian. 2022. Fast parallel Bayesian network structure learning. In *IEEE International Parallel and Distributed Processing Symposium*, pages 617–627. IEEE.

Tanqiu Jiang, Zian Wang, Jiacheng Liang, Changjiang Li, Yuhui Wang, and Ting Wang. 2025b. Robustkv: Defending large language models against jailbreak attacks via kv eviction. In *International Conference on Learning Representations*.

Xuanlin Jiang, Yang Zhou, Shiyi Cao, Ion Stoica, and Minlan Yu. 2025c. Neo: Saving GPU memory crisis with CPU offloading for online LLM inference. *Proceedings of Machine Learning and Systems*.

Shibo Jie, Yehui Tang, Kai Han, Zhi-Hong Deng, and Jing Han. 2025. SpeCache: Speculative key-value caching for efficient generation of LLMs. In *International Conference on Machine Learning*.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. 2025. RAGCache: Efficient knowledge caching for retrieval-augmented generation. *ACM Transactions on Computer Systems*. Just Accepted.

Lena Jurkschat, Preetam Gattogi, Sahar Vahdati, and Jens Lehmann. 2025. BALI-a benchmark for accelerated language model inference. *IEEE Access*.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. 2024. GEAR: An efficient KV cache compression recipe for near-lossless generative inference of LLM. *arXiv preprint arXiv:2403.05527*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 155–172.

Baolin Li, Yankai Jiang, Vijay Gadepally, and Devesh Tiwari. 2024a. LLM inference serving: Survey of recent advances and opportunities. *arXiv preprint arXiv:2407.12391*.

Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. 2024b. A survey on large language model acceleration based on KV cache management. *arXiv preprint arXiv:2412.19442*.

Junyan Li, Yang Zhang, Muhammad Yusuf Hassan, Talha Chafekar, Tianle Cai, Zhile Ren, Pengsheng Guo, Foroozan Karimzadeh, Chong Wang, and Chuang Gan. 2025. CommVQ: Commutative vector quantization for KV cache compression. In *International Conference on Machine Learning*.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024c. SnapKV: LLM knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Bin Lin, Tao Peng, Chen Zhang, Minmin Sun, Lanbo Li, Hanyu Zhao, Wencong Xiao, Qi Xu, Xiafei Qiu, Shen Li, Zhigang Ji, Yong Li, and Wei Lin. 2024. Infinite-LLM: Efficient LLM service for long context with distattention and distributed KVCache. *arXiv preprint arXiv:2401.02669*.

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

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. 2024a. MiniCache: KV cache compression in depth dimension for large language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 139997–140031.

Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. 2025a. RetrievalAttention: Accelerating long-context LLM inference via vector retrieval. *Advances in Neural Information Processing Systems*.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. 2025b. ClusterKV: Manipulating LLM KV cache in semantic space for recallable compression. In *ACM/IEEE Design Automation Conference*, pages 1–7. IEEE.

Peiyu Liu, Ze-Feng Gao, Xin Zhao, Yipeng Ma, Tao Wang, and Ji-Rong Wen. 2024b. Unlocking data-free low-bit quantization with matrix decomposition for KV cache compression. In *Annual Meeting of the Association for Computational Linguistics*, pages 2430–2440.

Yanyu Liu, Jingying Fu, Sixiang Liu, Yitian Zou, You Fu, Jiehan Zhou, and Shouhua Zhang. 2025c. KV cache compression for inference efficiency in LLMs: A review. *arXiv preprint arXiv:2508.06297*.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. 2024c. CacheGen: KV cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 38–56.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364.

Zihan Liu, Xinhao Luo, Junxian Guo, Wentao Ni, Yangjie Zhou, Yue Guan, Cong Guo, Weihao Cui, Yu Feng, Minyi Guo, Yuhao Zhu, Minjia Zhang, Jingwen Leng, and Chen Jin. 2025d. VQ-LLM: High-performance code generation for vector quantization augmented LLM inference. In *IEEE International Symposium on High Performance Computer Architecture*, pages 1496–1509. IEEE.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024d. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *International Conference on Machine Learning*, pages 32332–32344. PMLR.

Lingkun Long, Rubing Yang, Yushi Huang, Desheng Hui, Ao Zhou, and Jianlei Yang. 2025. SlimInfer: Accelerating long-context LLM inference via dynamic token pruning. *arXiv preprint arXiv:2508.06447*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Hongyi Jin, Tianqi Chen, and Zhihao Jia. 2023. Towards efficient generative large language model serving: A survey from algorithms to systems. *arXiv preprint arXiv:2312.15234*.

NVIDIA. 2023. TensorRT-LLM. `https://github.com/NVIDIA/TensorRT-LLM`.

NVIDIA. 2025a. GenAI-Perf. *NVIDIA Docs*.

NVIDIA. 2025b. NVIDIA NIM LLMs benchmarking. *NVIDIA Docs*.

OpenAI. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. 2024. Transformers are multi-state RNNs. In *Conference on Empirical Methods in Natural Language Processing*, pages 18724–18741.

Xiurui Pan, Endian Li, Qiao Li, Shengwen Liang, Yizhou Shan, Ke Zhou, Yingwei Luo, Xiaolin Wang, and Jie Zhang. 2024. InstInfer: In-storage attention offloading for cost-effective long-context LLM inference. *arXiv preprint arXiv:2409.04992*.

Zaifeng Pan, Ajjkumar Patel, Zhengding Hu, Yipeng Shen, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. 2025. KVFlow: Efficient prefix caching for accelerating LLM-based multi-agent workflows. *Advances in Neural Information Processing Systems*.

Daon Park and Bernhard Egger. 2024. Improving throughput-oriented LLM inference with CPU computations. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 233–245.

Jaehyun Park, Jaewan Choi, Kwanhee Kyung, Michael Jaemin Kim, Yongsuk Kwon, Nam Sung Kim, and Jung Ho Ahn. 2024. AttAcc! unleashing the power of PIM for batched transformer-based generative model inference. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 103–119.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient generative LLM inference using phase splitting. In *ACM/IEEE Annual International Symposium on Computer Architecture*, pages 118–132. IEEE.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624.

Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. 2025. vAttention: Dynamic memory management for serving LLMs without PagedAttention. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1133–1150.

Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2024. Mooncake: A KVCache-centric disaggregated architecture for LLM serving. *arXiv preprint arXiv:2407.00079*.

Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. 2025. CAKE: Cascading and adaptive KV cache eviction with layer preferences. In *International Conference on Learning Representations*.

Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew Kalbarczyk, Tamer Başar, and Ravishankar K Iyer. 2024. Power-aware deep learning model serving with μ-serve. In *USENIX Annual Technical Conference*, pages 75–93.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: Memory optimizations toward training trillion parameter models. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE.

Ray. 2024. LLMPerf. https://github.com/ray-project/llmperf.

Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, and 1 others. 2020. MLPerf inference benchmark. In *ACM/IEEE Annual International Symposium on Computer Architecture*, pages 446–459. IEEE.

Siyu Ren and Kenny Q Zhu. 2024. On the efficacy of eviction policy for key-value constrained generative language model inference. *arXiv preprint arXiv:2402.06262*.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. 2024. SparQ Attention: Bandwidth-efficient LLM inference. In *International Conference on Machine Learning*, pages 42558–42583. PMLR.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. 2024. Eigen Attention: Attention in low-rank space for KV cache compression. In *Findings of Empirical Methods in Natural Language Processing*, pages 15332–15344.

Rana Shahout, Cong Liang, Shiji Xin, Qianru Lao, Yong Cui, Minlan Yu, and Michael Mitzenmacher. 2024. Fast inference for augmented large language models. *arXiv preprint arXiv:2410.18248*.

Ao Shen, Zhiyao Li, and Mingyu Gao. 2024. Fastswitch: Optimizing context switching efficiency in fairness-aware large language model serving. *arXiv preprint arXiv:2411.18424*.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. FlexGen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR.

Dachuan Shi, Yonggan Fu, Xiangchi Yuan, Zhongzhi Yu, Haoran You, Sixu Li, Xin Dong, Jan Kautz, Pavlo Molchanov, and Yingyan Lin. 2025. LaCache: Ladder-shaped KV caching for efficient long-context modeling of large language models. In *International Conference on Machine Learning*.

Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. 2024. Keep the cost down: A review on methods to optimize LLM's KV-cache consumption. *arXiv preprint arXiv:2407.18003*.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. 2024. Loki: Low-rank keys for efficient sparse attention. *Advances in Neural Information Processing Systems*, 37:16692–16723.

Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. 2017. SmoothGrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*.

Donghyun Son, Euntae Choi, and Sungjoo Yoo. 2025. NSNQuant: A double normalization approach for calibration-free low-bit vector quantization of KV cache. *Advances in Neural Information Processing Systems*.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. 2024. Preble: Efficient distributed prompt scheduling for llm serving. In *International Conference on Learning Representations*.

Foteini Strati, Sara McAllister, Amar Phanishayee, Jakub Tarnawski, and Ana Klimovic. 2024. Déjàvu: KV-cache streaming for fast, fault-tolerant generative LLM serving. In *International Conference on Machine Learning*, pages 46745–46771.

Yi Su, Yuechi Zhou, Quantong Qiu, Juntao Li, Qingrong Xia, Ping Li, Xinyu Duan, Zhefeng Wang, and Min Zhang. 2025. Accurate KV cache quantization with outlier tokens tracing. In *Annual Meeting of the Association for Computational Linguistics*.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. 2025. ShadowKV: KV cache in shadows for high-throughput long-context LLM inference. In *International Conference on Machine Learning*.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. Quest: query-aware sparsity for efficient long-context LLM inference. In *International Conference on Machine Learning*, pages 47901–47911.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aur'elien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Niko lay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Daniel M. Bikel, Lukas Blecher, Cris tian Cantón Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023b. LLaMA 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, and Mi Zhang. 2025. D2O: Dynamic discriminative operations for efficient generative inference of large language models. In *International Conference on Learning Representations*.

Guanhua Wang, Zhuang Liu, Brandon Hsieh, Siyuan Zhuang, Joseph Gonzalez, Trevor Darrell, and Ion Stoica. 2021. sensAI: Convnets decomposition via class parallelism for fast inference on live data. *Proceedings of Machine Learning and Systems*, 3:664–679.

Hao Wang, Ligong Han, Kai Xu, and Akash Srivastava. 2025. SQuat: Subspace-orthogonal KV cache quantization. *arXiv preprint arXiv:2503.24358*.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. 2024a. Model tells you where to merge: Adaptive KV cache merging for LLMs on long-context tasks. *arXiv preprint arXiv:2407.08454*.

Zhibin Wang, Shipeng Li, Yuhang Zhou, Xue Li, Rong Gu, Nguyen Cam-Tu, Chen Tian, and Sheng Zhong. 2024b. Revisiting SLO and goodput metrics in LLM serving. *arXiv preprint arXiv:2410.14257*.

Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. 2024. LoongServe: Efficiently serving long-context large language models with elastic sequence parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 640–654.

Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. 2023. Fast distributed inference serving for large language models. *Advances in Neural Information Processing Systems*.

Wei Wu, Zhuoshi Pan, Chao Wang, Liyi Chen, Yunchu Bai, Tianfu Wang, Kun Fu, Zheng Wang, and Hui Xiong. 2025. TokenSelect: Efficient long-context inference and length extrapolation for LLMs via dynamic token-level KV cache selection. In *Conference on Empirical Methods in Natural Language Processing*, pages 21275–21292.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2024a. InfLLM: Training-free long-context extrapolation for LLMs with an efficient context memory. *Advances in Neural Information Processing Systems*, 37:119638–119661.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR.

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

Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. 2018. Gandiva: Introspective cluster scheduling for deep learning. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 595–610.

Yi Xiong, Hao Wu, Changxu Shao, Ziqing Wang, Rui Zhang, Yuhong Guo, Junping Zhao, Ke Zhang, and Zhenxuan Pan. 2024. LayerKV: Optimizing large language model serving with layer-wise KV cache management. *arXiv preprint arXiv:2410.00428*.

Fangyuan Xu, Tanya Goyal, and Eunsol Choi. 2025a. RefreshKV: Updating small KV cache during long-form generation. In *Annual Meeting of the Association for Computational Linguistics*, pages 24878–24893.

Jiale Xu, Rui Zhang, Cong Guo, Weiming Hu, Zihan Liu, Feiyang Wu, Yu Feng, Shixuan Sun, Changxu Shao, Yuhong Guo, Junping Zhao, Ke Zhang, Minyi Guo, and Jingwen Leng. 2024a. vTensor: Flexible virtual tensor management for efficient LLM serving. *arXiv preprint arXiv:2407.15309*.

Mengwei Xu, Wangsong Yin, Dongqi Cai, Rongjie Yi, Daliang Xu, Qipeng Wang, Bingyang Wu, Yihao Zhao, Chen Yang, Shihe Wang, Qiyang Zhang, Zhenyan Lu, Li Zhang, Shangguang Wang, Yuanchun Li, Yunxin Liu, Xin Jin, and Xuanzhe Liu. 2024b. A survey of resource-efficient LLM and multimodal foundation models. *arXiv preprint arXiv:2401.08092*.

Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. 2025b. ThinK: Thinner key cache by query-driven pruning. In *International Conference on Learning Representations*.

Xianglong Yan, Zhiteng Li, Tianao Zhang, Linghe Kong, Yulun Zhang, and Xiaokang Yang. 2025. ReCalKV: Low-rank KV cache compression via head reordering and offline calibration. *arXiv preprint arXiv:2505.24357*.

Dongjie Yang, Xiaodong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. 2024a. PyramidInfer: Pyramid KV cache compression for high-throughput LLM inference. In *Findings of the Association for Computational Linguistics*, pages 3258–3270.

June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. 2024b. No token left behind: Reliable KV cache compression via importance-aware mixed precision quantization. *arXiv preprint arXiv:2402.18096*.

Peiyu Yang, Naveed Akhtar, Zeyi Wen, and Ajmal Mian. 2023a. Local path integration for attribution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3173–3180.

Peiyu Yang, Naveed Akhtar, Zeyi Wen, Mubarak Shah, and Ajmal Saeed Mian. 2023b. Re-calibrating feature attributions for model interpretation. In *International Conference on Learning Representations*.

Yifei Yang, Zouying Cao, Qiguang Chen, Libo Qin, Dongjie Yang, Hai Zhao, and Zhi Chen. 2024c. KVSharer: Efficient inference via layer-wise dissimilar KV cache sharing. *arXiv preprint arXiv:2410.18517*.

Lu Ye, Ze Tao, Yong Huang, and Yang Li. 2024. ChunkAttention: Efficient self-attention with prefix-aware KV cache and two-phase partition. In *Annual Meeting of the Association for Computational Linguistics*, pages 11608–11620.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. 2025. FlashInfer: Efficient and customizable attention engine for LLM inference serving. *Proceedings of Machine Learning and Systems*.

Bohan Yu and Yekun Chai. 2025. EvolKV: Evolutionary KV cache compression for LLM inference. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 1673–1689.

Chengye Yu, Tianyu Wang, Zili Shao, Linjie Zhu, Xu Zhou, and Song Jiang. 2024. TwinPilots: A new computing paradigm for GPU-CPU parallel LLM inference. In *ACM International Systems and Storage Conference*, pages 91–103.

Lingfan Yu, Jinkun Lin, and Jinyang Li. 2025. Stateful large language model serving with pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 144–158.

Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, Yan Yan, Beidi Chen, Guangyu Sun, and Kurt Keutzer. 2024. LLM inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*.

Yuxuan Yue, Zhihang Yuan, Haojie Duanmu, Sifan Zhou, Jianlong Wu, and Liqiang Nie. 2024. WKVQuant: Quantizing weight and key/value cache for large language models gains more. *arXiv preprint arXiv:2402.12065*.

Ahmet Caner Yüzügüler, Jiawei Zhuang, and Lukas Cavigelli. 2025. PRESERVE: Prefetching model weights and KV-cache in distributed LLM serving. *arXiv preprint arXiv:2501.08192*.

Amir Zandieh, Majid Daliri, and Insu Han. 2025. QJL: 1-bit quantized JL transform for KV cache quantization with zero overhead. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 25805–25813.

Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. 2025a. PQCache: Product quantization-based KVCache for long context LLM inference. *Proceedings of the ACM on Management of Data*, 3(3):1–30.

Minwei Zhang, Haifeng Sun, Jingyu Wang, Shaolong Li, Wanyi Ning, Qi Qi, Zirui Zhuang, and Jianxin Liao. 2025b. ClusterAttn: KV cache compression under intrinsic attention clustering. In *Annual Meeting of the Association for Computational Linguistics*, pages 14451–14473.

Rongzhi Zhang, Kuang Wang, Liyuan Liu, Shuohang Wang, Hao Cheng, Chao Zhang, and Yelong Shen. 2024a. LoRC: Low-rank compression for LLMs KV cache with a progressive compression strategy. *arXiv preprint arXiv:2410.03111*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. 2024b. KV cache is 1 bit per channel: Efficient large language model inference with coupled quantization. *Advances in Neural Information Processing Systems*, 37:3304–3331.

Yanqi Zhang, Yuwei Hu, Runyuan Zhao, John CS Lui, and Haibo Chen. 2025c. DiffKV: Differentiated memory management for large language models with parallel KV compaction. In *ACM SIGOPS Symposium on Operating Systems Principles*, pages 431–445.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. 2024c. CaM: Cache merging for memory-efficient LLMs inference. In *International Conference on Machine Learning*, pages 58840–58850. PMLR.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2O: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

Junqi Zhao, Zhijin Fang, Shu Li, Shaohui Yang, and Shichao He. 2024a. BUZZ: Beehive-structured sparse KV cache with segmented heavy hitters for efficient LLM inference. *arXiv preprint arXiv:2410.23079*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Z. Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, and 3 others. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2024b. Atom: Low-bit quantization for efficient and accurate LLM serving. *Proceedings of Machine Learning and Systems*, 6:196–209.

Youpeng Zhao, Di Wu, and Jun Wang. 2024c. ALISA: Accelerating large language model inference via sparsity-aware KV caching. In *ACM/IEEE Annual International Symposium on Computer Architecture*, pages 1005–1017. IEEE.

Ranran Zhen, Juntao Li, Yixin Ji, Zhenlin Yang, Tong Liu, Qingrong Xia, Xinyu Duan, Zhefeng Wang, Baoxing Huai, and Min Zhang. 2025. Taming the titans: A survey of efficient LLM inference serving. In *International Natural Language Generation Conference*, pages 522–541.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph Gonzalez, Clark W. Barrett, and Ying Sheng. 2024. SGLang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems*, 37:62557–62583.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 193–210.

Xiabin Zhou, Wenbin Wang, Minyan Zeng, Jiaxian Guo, Xuebo Liu, Li Shen, Min Zhang, and Liang Ding. 2025. DynamicKV: Task-aware adaptive KV cache compression for long context LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 8042–8057.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhan Dong, and Yu Wang. 2024. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294*.

A Preliminaries on LLMs

LLMs are built from stacked Transformer blocks, each with multi-head self-attention (MHSA) and feed-forward network (FFN). These blocks are sequential, where the output of one block serves as the input to the next.

For the i-th attention head, MHSA applies learned projections W^{Q_i}, W^{K_i}, and W^{V_i} to the input features X to get queries, keys, and values:

$$Q_i = XW^{Q_i}, K_i = XW^{K_i}, V_i = XW^{V_i}.$$

Then the self-attention operation is applied to each tuple (Q_i, K_i, V_i) and get the output of Z_i:

$$Z_i = \mathrm{attention}(Q_i, K_i, V_i) = \mathrm{softmax}(\frac{Q_i K_i^\top}{\sqrt{d_k}})V_i,$$

where d_k is the dimension of the keys. Finally, outputs of all the attention heads are concatenated:

$$Z = \mathrm{concat}(Z_1, Z_2, ..., Z_h)W^O,$$

where W^O is the trainable parameters. Following this, the output of MHSA is fed into the FFN module, which applies two linear transformations with a nonlinear activation (e.g., ReLU):

$$\mathrm{FFN}(x) = \mathrm{ReLU}(xW_1 + b_1)W_2 + b_2,$$

where W_1, W_2, b_1, and b_2 are learnable parameters of the FFN. These modules together enable contextualized autoregressive modeling in LLMs.

B Design of Our Taxonomy

Our taxonomy follows a system behavior-oriented view of sKis introduced in § 2 and respects the domain boundary. Specifically, we classify techniques by their operational impact along three dimensions: temporal, spatial, and structural. This behavior-oriented perspective follows established practice in machine learning systems research (Xiao et al., 2018; Rajbhandari et al., 2020; Wang et al., 2021; Jiang et al., 2022; Qiu et al., 2024; Jiang et al., 2024) and aligns closely with how serving systems are actually built and optimized in practice, allowing diverse methods to be interpreted under a unified framework.

For example, many methods perform KV cache *selection* by identifying tokens (i.e., KV entries) that are more or less important for future computation. In our taxonomy, we do not treat selection itself as a category. In contrast, we classify methods based on the system action taken after selection:

- If unimportant KV entries are permanently discarded to free GPU memory, the method is categorized as KV cache eviction (cf. § 5.2.2) under KV representation and retention (structural dimension).
- If unimportant KV entries are offloaded to secondary storage (e.g., CPU RAM) for possible future retrieval and reload, the method falls under memory hierarchy KV orchestration (cf. § 4.1) in KV placement and migration (spatial dimension).
- If the tokens are retained in GPU memory but excluded from computation, the method is considered token-level scheduling, which is categorized as KV-centric scheduling (cf. § 3.1) under KV execution and scheduling (temporal dimension).

In short, selection is treated as a preparatory step, not a classification criterion itself. This helps prevent ambiguity and ensures that each category in our taxonomy corresponds to a distinct system-level optimization behavior.

C Related Surveys

To supplement the discussion in § 1, we here present existing related surveys and compare them with our work.

Several recent surveys have covered the areas of efficient LLM inference and serving. Miao et al. (2023) explored both algorithmic innovations and system architectures for efficient LLM serving, Yuan et al. (2024) analyzed LLM inference techniques through a Roofline-based framework, Zhou et al. (2024) organized efficient LLM inference methods across data-, model-, and system-level optimizations, Li et al. (2024a) examined system-level enhancements for LLM inference serving, Zhen et al. (2025) reviewed recent advances across different LLM serving scenarios, while Bai et al. (2024) and (Xu et al., 2024b) focused on resource-efficient LLMs. However, these **general surveys** typically treat KV cache optimization as a minor component within the broader pipelines.

In contrast, dedicated surveys that focus specifically on the KV cache remain rare. Shi et al. (2024) adopted a lifecycle-based taxonomy spanning training-stage, deploy-stage, and post-training optimizations. Li et al. (2024b) categorized KV cache management strategies into token-level, model-level, and system-level optimizations. Liu et al. (2025c) focused on compression strategies of the KV cache, such as selective token strategies, quantization, and attention compression. These **KV-specific surveys** are closest to our topic. However, they mostly organize methods by lifecycle stages or by abstraction levels, leaving the serving-time system behavior of the KV cache largely unexamined.

Different from the above surveys, we concentrate exclusively on the sKis scope (i.e., serving-time, KV-centric, system metrics, no retraining or architecture change) and aim to provide a deeper understanding within this scope. By classifying methods according to their impact along temporal, spatial, and structural dimensions, our survey enables cross-behavior and behavior×objective analysis, which complements prior surveys and clarifies actionable research gaps for KV-centric serving. Table 8 shows a comparative summary.

D Supplementary Paper Categorization

Table 9 provides a supplementary mapping of all surveyed methods across the full taxonomy of 7 subcategories under 3 major optimization di-

Table 8: Comparison of scope and taxonomy with existing surveys related to efficient LLM inference or serving.

Survey	KV-centric	Serving only	No retrain	System metrics	Organizing principle
Miao et al. (2023)		✓		✓	Algorithm-, system-level
Yuan et al. (2024)		✓		✓	Optimization layer (parameter-, algorithm-, system-, hardware-level)
Li et al. (2024a)		✓	✓	✓	System component (KV cache and memory, computation, cloud deployment, emerging research fields)
Zhou et al. (2024)		✓		✓	Optimization layer (data-, model-, system-level)
Zhen et al. (2025)		✓	✓	✓	Serving scale (instance-, cluster-level, emerging scenarios)
Bai et al. (2024)				✓	Lifecycle (architecture design, pre-training, fine-tuning, inference, system design)
Xu et al. (2024b)				✓	Optimization layered (architecture, algorithm, systems)
Shi et al. (2024)	✓			✓	Lifecycle (training, deploy, post-training)
Li et al. (2024b)	✓	✓		✓	Optimization layer (token-, model-, system-level)
Liu et al. (2025c)	✓	✓		✓	KV compression types (selective token, quantization, attention compression, hybrid)
This survey (sKis)	✓	✓	✓	✓	System behaviors (temporal, spatial, structural dimensions)

mensions. The finer-grained categories in this table include (i) KV-centric scheduling (cf. § 3.1), (ii) pipelining and overlapping (cf. § 3.2), (iii) hardware-aware execution (cf. § 3.3), (iv) memory hierarchy KV orchestration (cf. § 4.1), (v) compute device KV orchestration (cf. § 4.2), (vi) KV cache compression (cf. § 5.1), and (vii) KV cache retention management (cf. § 5.2).

As discussed in § 2, to maintain structural clarity and prevent overly diffuse categorization, each method is primarily discussed under one or two key optimization categories that reflect its main contributions. These categories are denoted as primary category (●) in Tab. 9. However, some methods also touch upon additional optimization aspects that are not covered or elaborated in the main sections. For example, to support its "hardware-aware execution" design of decoupling prefill and decode phases across heterogeneous devices, Splitwise (Patel et al., 2024) incorporates a fine-grained layer-wise transmission strategy that transmits the KV cache from the prefill node to the decode node and overlaps such KV cache transmission with the computation in the prefill phase. They serve as enabling mechanisms that make the decoupled strategy feasible and link Splitwise to the "device-level KV transfer" and "pipelining and overlapping" categories. We summarize these omitted associations in Tab. 9, denoted by ◐, to provide a more complete mapping for readers interested in cross-cutting techniques.

Venue Diversity. We can observe from the "Venue" column of Tab. 9 that the methods span a broad range of research communities. The publication venues include top-tier machine learning and artificial intelligence conferences (e.g., ICLR, ICML, NeurIPS, AAAI), natural language processing venues (e.g., ACL, EMNLP, COLM), systems and architecture conferences (e.g., ASPLOS, ISCA, HPCA, FAST, ATC, EuroSys, OSDI, SOSP, SC, SIGCOMM, DAC), and interdisciplinary forums such as MLSys and SIGMOD. We also include some impactful arXiv submissions. This diversity underscores the inherently cross-cutting nature of KV cache optimization, which lies at the intersection of model serving and system efficiency. It also highlights the growing recognition of this topic across various research communities.

E Takeaways

Through a comprehensive literature review of sKis, we have discovered takeaways across several domains. These include scheduling & overlapping, hardware-aware execution, placement & migration, compression, and eviction.

E.1 Scheduling and Overlapping

KVS and OVLP directly target runtime stalls. KVS prioritizes limited resources for the most reusable and latency-sensitive work; OVLP is also a type of scheduling, aligning compute with data transfer to fill pipeline bubbles.

Takeaway:

✓ KVS is a multi-objective optimization problem. Modern schedulers often prioritize KV usage over time rather than FLOPS, and KV reuse-driven scheduling is the default paradigm.

✓ KVS is enhanced by prediction. Lightweight

Table 9: Full mapping of representative methods reviewed in this paper to their corresponding sKis categories. Methods are chronologically ordered with publication venues.

Methods	Venue	KV-centric scheduling	Pipelining and overlapping	Hardware-aware execution	Memory hierarchy KV orchestration	Compute device KV orchestration	KV cache compression	KV cache retention management
		Taxonomy of sKis						
SmoothQuant (Xiao et al., 2023)	ICML						●	
FlexGen (Sheng et al., 2023)	ICML		◐	◐	●		●	
vLLM (Kwon et al., 2023)	SOSP				◐			●
FastServe (Wu et al., 2023)	NeurIPS		◐		●	●		
H_2O (Zhang et al., 2023)	NeurIPS							●
Scissorhands (Liu et al., 2023)	NeurIPS							●
TetriInfer (Hu et al., 2024b)	arXiv	●		●		◐		
RoCo (Ren and Zhu, 2024)	arXiv							●
WKVQuant (Yue et al., 2024)	arXiv						●	
MiKV (Yang et al., 2024b)	arXiv						●	
FastDecode (He and Zhai, 2024)	arXiv		●	●		◐	◐	
QAQ (Dong et al., 2024)	arXiv						●	
AttAcc (Park et al., 2024)	ASPLOS			●		●		
FastGen (Ge et al., 2024)	ICLR							●
StreamingLLM (Xiao et al., 2024b)	ICLR							●
Preble (Srivatsa et al., 2024)	ICLR	●		◐				◐
Keyformer (Adnan et al., 2024)	MLSys							●
Atom (Zhao et al., 2024b)	MLSys						●	
PromptCache (Gim et al., 2024)	MLSys							●
PyramidKV (Cai et al., 2024)	arXiv							●
Splitwise (Patel et al., 2024)	ISCA		◐	●		●		
ALISA (Zhao et al., 2024c)	ISCA				●		◐	◐
DistServe (Zhong et al., 2024)	OSDI		◐	●		●		
Infinite-LLM (Lin et al., 2024)	arXiv		◐	●		●		
InfiniGen (Lee et al., 2024)	OSDI				●			
CachedAttention (Gao et al., 2024)	ATC	◐	●		●	◐		◐
LazyLLM (Fu et al., 2024)	arXiv							●
KVMerger (Wang et al., 2024a)	arXiv						●	
vTensor (Xu et al., 2024a)	arXiv							●
KIVI (Liu et al., 2024d)	ICML						●	
CHAI (Agarwal et al., 2024)	ICML						●	
CaM (Zhang et al., 2024c)	ICML						●	
MuxServe (Duan et al., 2024)	ICML	●		●		◐		◐
Quest (Tang et al., 2024)	ICML	●						
SparQAttention (Ribar et al., 2024)	ICML	●						
DéjàVu (Strati et al., 2024)	ICML			●	●	◐		◐
CacheGen (Liu et al., 2024c)	SIGCOMM		◐			●	●	
DecoQuant (Liu et al., 2024b)	ACL						●	
NACL (Chen et al., 2024b)	ACL							●
PyramidInfer (Yang et al., 2024a)	ACL							●
ChunkAttention (Ye et al., 2024)	ACL							●
InstInfer (Pan et al., 2024)	arXiv		◐	●		●		◐
TwinPilots (Yu et al., 2024)	SYSTOR		◐	●		◐		
GEAR (Kang et al., 2024)	arXiv						●	
LoRC (Zhang et al., 2024a)	arXiv						●	
SKVQ (Duanmu et al., 2024)	COLM						●	
LayerKV (Xiong et al., 2024)	arXiv	●	◐		●			◐
CComp (Park and Egger, 2024)	PACT		●	●		◐		
KVSharer (Yang et al., 2024c)	arXiv						●	
LAMPS (Shahout et al., 2024)	arXiv	●			◐			◐
BUZZ (Zhao et al., 2024a)	arXiv							●
LoongServe (Wu et al., 2024)	SOSP	●	◐	◐				◐
EigenAttention (Saxena et al., 2024)	EMNLP						●	
TOVA (Oren et al., 2024)	EMNLP							●
VATP (Guo et al., 2024)	EMNLP							●
L2KV (Devoto et al., 2024)	EMNLP							●
FastSwitch (Shen et al., 2024)	arXiv	◐		◐	●			●

Continued on next page

● = primary category with main analysis; ◐ = secondary category omitted or only briefly mentioned in our paper to maintain focused classification.

Continued from previous page

Methods	Venue	Taxonomy of sKis						
		KV-centric scheduling	Pipelining and overlapping	Hardware-aware execution	Memory hierarchy KV orchestration	Compute device KV orchestration	KV cache compression	KV cache retention management
KVQuant (Hooper et al., 2024)	NeurIPS						●	
CQ (Zhang et al., 2024b)	NeurIPS						●	
ZipCache (He et al., 2024)	NeurIPS						●	
SnapKV (Li et al., 2024c)	NeurIPS							●
MiniCache (Liu et al., 2024a)	NeurIPS						●	
InfLLM (Xiao et al., 2024a)	NeurIPS				●			◐
RadixAttention (Zheng et al., 2024)	NeurIPS	●						●
Loki (Singhania et al., 2024)	NeurIPS	●						
ArkVale (Chen et al., 2024a)	NeurIPS				●		◐	
MemServe (Hu et al., 2024a)	arXiv							●
Mooncake (Qin et al., 2024)	FAST	●	◐	●		◐		
IMPRESS (Chen et al., 2025b)	FAST				●			◐
QJL (Zandieh et al., 2025)	AAAI						●	
VQ-LLM (Liu et al., 2025d)	HPCA						●	
xKV (Chang et al., 2025a)	arXiv						●	
SQuat (Wang et al., 2025)	arXiv						●	
vAttention (Prabhu et al., 2025)	ASPLOS		◐					●
PAPI (He et al., 2025)	ASPLOS			●				
Pensieve (Yu et al., 2025)	EuroSys	◐	◐		●			◐
AsyncKV (Dong et al., 2025)	arXiv		●	◐	●			
Palu (Chang et al., 2025b)	ICLR						●	
CAKE (Qin et al., 2025)	ICLR							●
D_2O (Wan et al., 2025)	ICLR						●	●
ThinK (Xu et al., 2025b)	ICLR						●	
MagicPIG (Chen et al., 2025c)	ICLR			●				
QoQ (Lin et al., 2025)	MLSys						●	
FlashInfer (Ye et al., 2025)	MLSys	●		◐	◐			●
Neo (Jiang et al., 2025c)	MLSys		●	●		◐		
PRESERVE (Yüzügüler et al., 2025)	arXiv		●	◐	●			
ReCalKV (Yan et al., 2025)	arXiv						●	
ClusterKV (Liu et al., 2025b)	DAC				●			
PQCache (Zhang et al., 2025a)	SIGMOD		◐		●		◐	
ShadowKV (Sun et al., 2025)	ICML		◐		●		●	
SepLLM (Chen et al., 2025a)	ICML							●
CommVQ (Li et al., 2025)	ICML						●	
LaCache (Shi et al., 2025)	ICML							●
SpeCache (Jie et al., 2025)	ICML		◐		●			
RocketKV (Behnam et al., 2025)	ICML	●						◐
ClusterAttn (Zhang et al., 2025b)	ACL						●	
RefreshKV (Xu et al., 2025a)	ACL	●						
OTT (Su et al., 2025)	ACL						●	
KVPR (Jiang et al., 2025a)	ACL		●	◐			◐	
SlimInfer (Long et al., 2025)	arXiv		◐		●			
RAGCache (Jin et al., 2025)	TOCS		◐		●			
KVCompose (Akulov et al., 2025)	arXiv							●
LMCache (Cheng et al., 2025)	arXiv		◐	◐	●	●		◐
DiffKV (Zhang et al., 2025c)	SOSP						◐	●
TokenSelect (Wu et al., 2025)	EMNLP	●						◐
EvolKV (Yu and Chai, 2025)	EMNLP							●
DynamicKV (Zhou et al., 2025)	EMNLP							●
RetrievalAttention (Liu et al., 2025a)	NeurIPS				●			
Ada-KV (Feng et al., 2025)	NeurIPS							●
NSNQuant (Son et al., 2025)	NeurIPS						●	
KVFlow (Pan et al., 2025)	NeurIPS	◐	◐		●		●	

● = primary category with main analysis; ◐ = secondary category omitted or only briefly mentioned in our paper to maintain focused classification.

predictors plus a robust policy outperform traditional FCFS or SJF schemes (Hu et al., 2024b; Qin et al., 2024; Shahout et al., 2024).

- ✓ The key to OVLP is to perform at the true bottleneck with asymmetric pipelines. For example, keep compute-bound prefill on GPU, and overlap memory-bound decode attention and KV with I/O or collective communication.
- ✓ Preferring recompute to transfer, e.g., partially recomputing KV while streaming the rest (Jiang et al., 2025a), or prefetching KV caches into L2 during collectives (Dong et al., 2025; Yüzügüler et al., 2025), can substantially reduce pipeline bubbles, especially when bandwidth is the bottleneck.

E.2 Hardware-aware Execution

HAE improves throughput, reduces mean/tail latency, and extends servable context without retraining, by decoupling phases and mapping execution to hardware capabilities.

Takeaway:

- ✓ Compute should follow hardware capabilities. When executing on a given device, it is critical to specialize kernels, tiling, and memory layouts to that device.
- ✓ Create KV locality within the device rather than moving KV across devices. It is effective to keep hot KV caches close to the compute.
- ✓ Compute-intensive prefill and memory-bound decode benefit from phase-specific execution mappings (cf. § 3.3.2).
- ✓ HAE should adapt to the access granularity and parallelism of the target device.

E.3 Placement and Migration

MHO and CDO govern where KV caches reside across the memory hierarchy and how they transfer during serving. They act directly on interconnect bandwidth bottlenecks, with GPU memory relief emerging as a by-product of tiering and offloading.

Takeaway:

- ✓ It is a common MHO pattern to keep only future-useful KV caches on the GPU, demote the rest to CPU or SSD, and reload guided by attention cues. Cost models can be effectively used to choose CPU, GPU, SSD paths (Sheng et al., 2023; Jin et al., 2025).
- ✓ Most MHO and CDO solutions overlap I/O transfers with compute or collectives to hide latency, although they often serve OVLP as a secondary category.
- ✓ Under interconnect bottlenecks, co-adaptation of transfer paths, precisions, or decoding strategies can reduce TTFT and SLO violations compared with static schemes (Zhong et al., 2024; Liu et al., 2024c; Shen et al., 2024).
- ✓ Migration granularity and path should align with attention access patterns and device access units.
- ✓ Prefetch-evict co-optimization remains rare. The field would benefit from a unified objective that jointly accounts for prefetch deadlines and eviction risk.

E.4 KV Cache Compression

KV caches can quickly overwhelm the memory capacity of GPUs and pose bandwidth pressure as context length or batch size increases, since the size of the KV cache scales linearly with these two factors. Consequently, prior works have proposed various approaches to directly compress the KV cache, such as quantization, low-rank approximation, and structural compression.

Takeaway:

- ✓ Outlier handling dominates performance at low bitwidths or ranks (Su et al., 2025). Isolating outliers (e.g., higher bitwidths) for value-level compression methods prevents worst-case error explosions.
- ✓ Recent advances trend toward applying vector quantization (VQ) for KV cache quantization, and they often reach very low-bit (i.e., 1-2 bits) quantization with modest quality loss (Zhang et al., 2024b; Liu et al., 2025d; Son et al., 2025; Li et al., 2025). VQ (Gray, 1984) is a popular technique to represent high-dimensional data using a smaller set of representative vectors, known as codebooks. Variants of VQ like product quantization (Jegou et al., 2010) and additive quantization (Babenko and Lempitsky, 2014) have been proposed to applied to KVCC.
- ✓ KVCC has been developed mostly at the algorithm level, while system-level integration is thin (cf. Fig. 6). Thus, memory reductions often fail to translate into lower mean/tail latency or higher throughput unless KVCC is co-designed with execution, migration, and runtime control.

We further discuss the co-design of KVCC with execution, migration, and runtime control (the last akeaway) as follows: (i) Co-design with execution: quantization/de-quantization and low-rank updates can be fused into attention kernels or overlapped with compute, so compression overhead does not re-introduce stalls in the decode pipeline; (ii) Co-design with migration: aligning compressed packing units with device access units ensures that memory footprint reductions translate into fewer, fully utilized transfer chunks that fit overlap windows. (iii) Co-design with runtime control: exposing tunable parameters (e.g., bitwidth, rank, sparsity) to the runtime and adjusting them under SLOs remains an opportunity beyond static configurations.

E.5 KV Cache Eviction

KV cache eviction decides which past tokens remain resident under tight memory and bandwidth budgets, so that long contexts can be served. It operates in both phases and trades memory and transfer cost against utility to the attention compute.

Takeaway:

- ✓ KV cache eviction is important in both prefill and decode. The former focuses on the KV cache to be computed, while the latter focuses on the KV cache that has been computed.
- ✓ Most systems retain a small recent window, a tiny set of "attention sink" anchor tokens (Xiao et al., 2024b; Gu et al., 2025), and a few "heavy hitters" (Zhang et al., 2023) identified by cumulative attention.
- ✓ Token importance should not be judged by attention scores alone. KV norms provide strong and low-overhead signals (Guo et al., 2024; Devoto et al., 2024). We also recommend calibrating token importance scores before using them for eviction (Sundararajan et al., 2017; Smilkov et al., 2017; Yang et al., 2023a,b).
- ✓ It is effective to use heterogeneous budgets across layers or heads, rather than a uniform upper bound (Cai et al., 2024; Yang et al., 2024a; Wan et al., 2025; Qin et al., 2025; Shi et al., 2025; Akulov et al., 2025; Zhang et al., 2025c; Yu and Chai, 2025; Zhou et al., 2025; Feng et al., 2025). For example, shallow layers often deserve larger retention, while deeper layers emphasize global semantics and tolerate more sparsity.
- ✓ Pairing KV cache eviction with similarity-based recall or merge is stronger than hard deletion, preserving salient context under tight budgets and improving long-context consistency (Wan et al., 2025).

	KVS	OVLP	HAE	MHO	CDO	KVCC	KVRM
KVS	–	2.5	4.75	4	1.75	0	6.5
OVLP	2.5	–	9.25	8.5	6.5	2.75	2.25
HAE	4.75	9.25	–	3.75	10	1.25	3.25
MHO	4	8.5	3.75	–	3	3.5	5.75
CDO	1.75	6.5	10	3	–	1.25	2.75
KVCC	0	2.75	1.25	3.5	1.25	–	1.75
KVRM	6.5	2.25	3.25	5.75	2.75	1.75	–

Table 11: Raw (pre-normalization) co-occurrence matrix that encodes the weighted co-occurrence strength between system behaviors across papers.

F Behavior-behavior Co-design Affinity Computation

As discussed in § 6, Fig. 6 presents a behavior-behavior co-design affinity network that summarizes how often behaviors co-occur within the same paper across seven behaviors, including KVS, OVLP, HAE, MHO, CDO, KVCC, and KVRM. Below we detail the procedure for calculating the normalized co-occurrence strengths for behavior pairs, which are reflected by the edge thicknesses in Fig. 6.

Let $\mathcal{B}$ ={KVS, OVLP, HAE, MHO, CDO, KVCC, KVRM} denote the set of system behaviors and $\mathcal{P}$ the set of papers. For paper $p \in \mathcal{P}$ and behavior $i \in \mathcal{B}$, let the categorical label be $\ell_{p,i} \in \{\mathsf{P}, \mathsf{S}, \mathsf{NA}\}$, which means primary category (●), secondary category (◐), or no category. Each $\ell_{p,i}$ can be observed from Tab. 9. We map labels to numeric weights by $\omega(\ell_{p,i}) = \mathbb{1}_{[\ell_{p,i}=\mathsf{P}]} + \alpha \mathbb{1}_{[\ell_{p,i}=\mathsf{S}]}$ with $\alpha = 0.5$, where $\mathbb{1}_{[\cdot]}$ is the indicator function that equals 1 when the stated condition holds and 0 otherwise.

Constructing raw co-occurrence. The raw co-occurrence matrix $C \in \mathbb{R}^{|\mathcal{B}|\times|\mathcal{B}|}$ aggregates pairwise co-appearance strength, as shown in Tab. 11. Each cell C_{ij} of the behavior pair i, j is defined by summing the per-paper products of their weights:

$$C_{ij} = \sum_{p\in\mathcal{P}} \omega(\ell_{p,i})\omega(\ell_{p,j}).$$

Equivalently, each paper p contributes 1 if both behaviors are primary, α if one is primary and the other secondary, α^2 if both are secondary, and 0 otherwise. The matrix C is symmetric.

Constructing normalized co-design affinity. While the raw co-occurrence matrix C captures absolute overlap, it is biased by marginal popularity, because the behaviors with larger research density tend to have larger C_{ij} even without specific affinity. We therefore normalize C using the Tanimoto

	KVS	OVLP	HAE	MHO	CDO	KVCC	KVRM
KVS	–	0.09	0.16	0.11	0.07	0	0.14
OVLP	0.09	–	0.42	0.30	0.38	0.06	0.05
HAE	0.16	0.42	–	0.10	0.53	0.02	0.06
MHO	0.11	0.30	0.10	–	0.10	0.06	0.10
CDO	0.07	0.38	0.53	0.10	–	0.03	0.06
KVCC	0	0.06	0.02	0.06	0.03	–	0.02
KVRM	0.14	0.05	0.06	0.10	0.06	0.02	–

Table 12: Normalized co-design affinity matrix that encodes relative co-occurrence strength between behaviors across papers. Scores greater than the threshold $\theta = 0.14$ are highlighted and visualized in Fig. 6 accordingly.

coefficient. We define the per-behavior squared weight Q_i:

$$Q_i = \sum_{p \in \mathcal{P}} w_{p,i}^2.$$

Then the Tanimoto-normalized co-design affinity matrix $S \in \mathbb{R}^{|\mathcal{B}| \times |\mathcal{B}|}$ reflects relative co-occurrence strength on a $[0, 1]$ scale, as shown in Tab. 12. Each cell in S_{ij} of the behavior pair i, j is defined as the ratio of their shared weighted presence to their squared union:

$$S_{ij} = \frac{C_{ij}}{Q_i + Q_j - C_{ij}}.$$

Compared to C_{ij}, this score controls marginal sizes and is visualized in Fig. 6. We draw an undirected edge between behaviors i and j iff $S_{ij} > \theta$, where we set the threshold $\theta = 0.14$; edges below the threshold are omitted to reduce clutter. Edge thickness is proportional to S_{ij}.

G Extended Discussion on Observations and Open Challenges

Due to space constraints, this section complements § 6 with further discussion of observations and open challenges, including an overview of observations and open challenges, trustworthy sKis, intermediate semantics, and sKis benchmarking.

G.1 Overview of Observations and Open Challenges

To improve navigability, we here provide a compact summary table in Fig. 7, which links each open challenge (C1-C6 in § 6) to its motivating observation and highlights the key future research directions at a glance.

G.2 Trustworthy sKis

Trustworthiness is an important topic for LLM serving. As discussed in C3 in § 6, efficiency optimizations typically account for average quality loss, but trustworthiness is rarely measured or attributed. The challenge lies in identifying concrete KV behaviors that degrade trust.

One representative example (also related to our discussion in C3) is that KV cache eviction and compression can compromise *quality robustness*. They may drop rare but critical tokens with low accumulated attention (e.g., an exception clause in a contract, or a high value in a financial limit), which can lead to catastrophic errors on a small subset of inputs while the system still appears efficient and accurate on average. This failure mode can be amplified by distribution shift in workloads, such as in autonomous agent workloads, where statistically sparse tokens become logically important. Optimizations tuned to the original distribution may prune these sparse critical dependencies, causing agents to hallucinate success.

Trustworthiness risks extend beyond robustness to reliability, privacy, and safety, and can arise from diverse sKis behaviors. For instance, temporal asynchrony may expose stale KV and introduce nondeterminism, harming reliability; cross-tier migration can leave residual KV state or transfer KV in plaintext, harming privacy.

A key gap is that many methods only measure average metrics on relatively easy workloads, but rarely consider quality lower bound, recall SLO, or semantic violation metrics, so such worst-case failures remain invisible. A promising direction is to consider trustworthy metrics and integrate runtime mechanisms such as violation detectors and recovery policies to provide a quality lower bound under stress.

G.3 Intermediate Semantics for Behaviors

We here provide additional discussion of intermediate semantics as a supplement to C5 in § 6.

Intuitively, intermediate semantics for sKis behaviors aim to bridge the gap between binary decisions (e.g., "retain" vs. "evict"). Future research could explore intermediate states between the binary states, such as "reclaimable on GPU", "compressed on GPU", "compressed on CPU", "summarized on CPU/SSD", and so on. In this way, a co-optimization strategy can be formalized as a transition between these states. For example, the

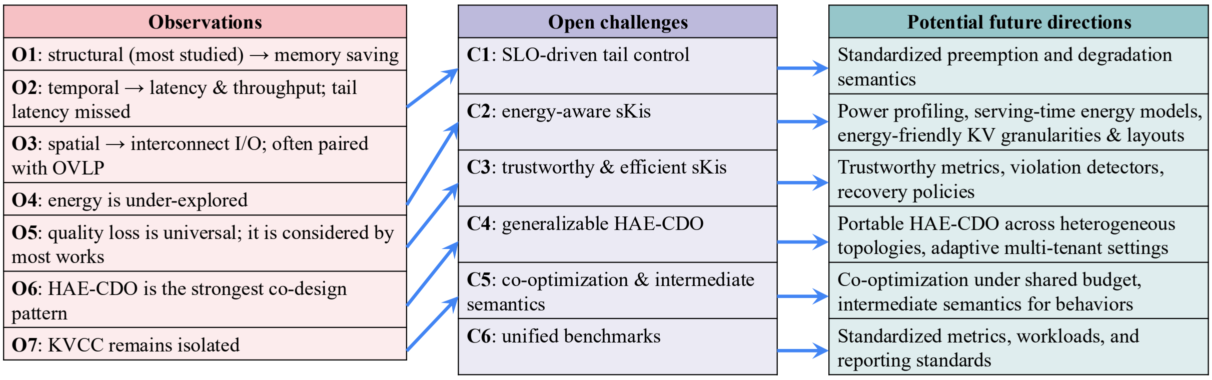

Figure 7: A roadmap linking open challenges to motivating observations and potential research directions.

compress-then-offload strategy first transitions a KV unit from a “keep” state to a “compressed on GPU” state, then to a “compressed on CPU” state. This creates a low-fidelity resident state that trades precision for I/O bandwidth. Similarly, lazy eviction transitions a KV unit to a “reclaimable on GPU” state with a grace period to allow cheap recovery before the final transition to permanent eviction (i.e., state “evict”). These concrete examples show how future work can co-optimize eviction, compression, and migration by exploiting intermediate semantics.

G.4 Benchmarking for sKis

In this section, we focus on system-performance benchmarking during serving, which measures actual performance metrics like latency, throughput, service-level objectives (SLOs), KV cache memory and bandwidth, and energy. In contrast, task or quality benchmarks focus on datasets and accuracy metrics. In our survey they serve only as quality gates and are not primary evaluation objectives. We refer interested readers to another survey (Li et al., 2024b) for details.

In what follows, we first review recent efforts on sKis benchmarking, then provide actionable benchmarking guidelines, including recommended metrics, workloads, and reporting standards.

G.4.1 Review of sKis Benchmarking Practices

Many popular inference frameworks or systems, such as vLLM (Kwon et al., 2023), TensorRT-LLM (NVIDIA, 2023), and DeepSpeed-Inference (Aminabadi et al., 2022), provide benchmark scripts that measure system metrics for their local checks but remain framework-specific. We therefore view them as systems under test rather than the benchmark itself. In this section, we survey benchmarking efforts that provide a platform- and framework-agnostic way to obtain system measurements. They primarily fall into two categories: client-side tools and benchmark suites.

Client-side tools define and enforce metric semantics. Using one tool across systems yields directly comparable numbers. LLMPerf (Ray, 2024) targets API benchmarking and provides system metric measurement on service endpoints. NVIDIA NIM benchmarking guide (NVIDIA, 2025b) defines the common metrics of time to first token (TTFT), end-to-end request latency, inter-token latency (ITL), tokens per second (TPS), and requests per second (RPS). The companion tool GenAI-Perf (NVIDIA, 2025a) emits the defined metrics and implements the stable-window analysis across OpenAI-API-compatible backends. However, although client-side tools offer specific metrics for LLM-based applications, we find inconsistent metric definitions and measurements across different tools.

Benchmark suites mean standardized packages of workloads, procedures, and reporting rules that specify what to run, how to run it, and what to report. Such suites typically cover multiple systems and hardware and enable reproducible comparisons. MLPerf Inference (Reddi et al., 2020) emphasizes inference system comparison, and its v5.0 includes LLM scenarios with accuracy validation. LLM-Inference-Bench (Chitty-Venkata et al., 2024) evaluates the inference performance of the LLaMA model family across a variety of hardware platforms. BALI (Jurkschat et al., 2025) measures LLM inference across six frameworks or acceleration approaches. It divides inference into three measured stages: setup, tokenize, and generate, and supports two settings: a technical setting with a fixed number of tokens, and a prompt-to-answer setting that includes tokenization.

G.4.2 Actionable Benchmarking Guidelines

Building on the above review, we distill actionable benchmarking guidelines for sKis to improve comparability across systems. We summarize recommended metrics, representative workloads, and reporting standards.

- **Metrics**: Besides standard metrics, we recommend sKis benchmarks report (i) trustworthy metrics that reflect the reliability of the serving system in satisfying SLOs, such as tail latency (P90, P95, P99 latency), SLO violation rate (% of requests > P99 target), goodput (throughput meeting SLOs), recall SLO (success rate of certain semantic segments), and semantic violation rate; and (ii) KV-related resource metrics that measure the utilization of resources, such as KV cache memory footprint (as % of total GPU memory), average effective KV bitwidth (for compression methods), KV-related interconnect I/O (the volume of KV transferred across memory tiers), KV hit rate in memory tiers, KV-related stalls (% of time spent waiting for KV transfers), effective bandwidth utilization (useful KV transfer ratio), and energy efficiency (Joules per token/request).
- **Workloads**: We suggest that sKis benchmarks should at least cover the following three workload types to stress temporal, spatial, and structural KV behaviors: (i) multi-tenant or bursty online serving workloads to test the stability of temporal scheduling under high concurrency, (ii) long-context task workloads to test KV cache placement and migration when memory and I/O become bottlenecks, (iii) heterogenous workloads (e.g., RAG or agent workloads) to test the robustness of structural KV cache optimizations against distribution shift.
- **Reporting standards**: In addition to the basic information like model, hardware, and configuration, we will recommend the following reporting standards for sKis benchmarks: (i) performance under graduated context lengths to validate scalability; (ii) accuracy vs. memory curves for structural methods to reveal the trade-offs; (iii) detailed hardware and topology setups, especially for temporal and spatial methods.

H The Use of AI assistants

We used ChatGPT minimally for wording and grammar suggestions. No technical claims, taxonomy decisions, or analyses were produced by the assistant. All content was authored, verified, and edited by the authors.