Efficient Long-Context LLM Inference on the Edge with Hierarchical KV Eviction using SmartSSD

Anonymous ACL submission

Abstract

Deploying Large Language Models (LLMs) on edge devices such as Personal Computers (PCs) enables low-latency inference with strong privacy guarantees, but long-context inference is fundamentally constrained by limited memory and compute resources. Beyond model parameters, the key–value (KV) cache becomes the dominant bottleneck due to its linear growth with context length. Although prior work exploits contextual sparsity to evict unimportant KV data, these approaches are largely designed for memory-rich platforms and incur prohibitive data transfer overhead when applied to resource-constrained edge devices with external storage. In this paper, we propose HillInfer, an importance-aware long-context LLM inference framework on the edge that leverages SmartSSD-assisted hierarchical KV cache management. HillInfer jointly manages KV cache pools across the CPU and SmartSSD, and performs in-storage importance evaluation to reduce unnecessary data movement. Furthermore, we design an adaptive, prefetch-based pipeline that overlaps computation and KV data transfer across GPU, CPU, and SmartSSD, minimizing end-to-end inference latency without sacrificing accuracy. We implement HillInfer on a PC with a commodity GPU, and experiments across multiple models and benchmarks demonstrate up to 8.56 $\times$ speedup over baselines while preserving model accuracy.

1 Introduction

Large Language Models (LLMs) (Zhao et al., 2023; Chang et al., 2024; Minaee et al., 2024; Naveed et al., 2025; Li et al., 2024; Friha et al., 2024) are transforming artificial intelligence by demonstrating exceptional capability in understanding natural language and supporting a wide range of language-centric downstream tasks (Isik et al., 2024; Wei et al., 2021; Oyelade et al., 2025). In practice, many of these tasks rely on prompts that inherently involve sensitive information, such as personal medical records, proprietary creative content, and confidential business data, etc. This privacy concern has driven a growing trend toward deploying LLMs on edge devices, such as Personal Computers (PCs), giving rise to edge-based LLM inference (Cai et al., 2024; Yu et al., 2024; Lu et al., 2024).

However, the limited memory capacity and computational resources of edge devices pose fundamental challenges to efficient LLM deployment (Tian et al., 2025; Yin et al., 2026). Even worse, the growing complexity and diversity of downstream tasks increasingly require LLMs to process longer input contexts (Liu et al., 2024; Sun et al., 2025; Lin et al., 2024). As a result, beyond the memory occupied by model parameters, the KV cache (Li et al., 2024), whose memory footprint grows linearly with the input context length, becomes the primary memory bottleneck in long-context LLM inference on edge devices (Sun et al., 2025; Zhang et al., 2025; Hooper et al., 2025). As illustrated in Figure 2a, inference with the Qwen-7B (Bai et al., 2023) model under a 4K context and batch size of 8 consumes approximately 30 GB of memory, surpassing the 24 GB GPU memory capacity of the high-end commodity GPU such as the NVIDIA RTX 4090, with the majority of this overhead attributed to the KV cache.

Limitations of Prior Arts. Inspired by the Transformer architecture and linguistic characteristics (Vaswani et al., 2017; Beltagy et al., 2020; Levy and Goldberg, 2014), long-context inputs exhibit a high degree of sparsity, where only a small subset of tokens, which is usually less than 20% (Sun et al., 2025; Lee et al., 2024; Tang et al., 2024), plays a critical role in attention computations (Beltagy et al., 2020). Existing H2O-like (Zhang et al., 2023) approaches leverage this observation by estimating token importance and selectively retaining the KV data of important tokens on the GPU for computation, while evicting less important KV data in each autoregressive decoding step (Lee et al.,

2024; Zhang et al., 2023; Zhao et al., 2024; Gao et al., 2024; Tang et al., 2024; Liu et al., 2023; Juravsky et al.; Ye et al., 2024; Zheng et al., 2024). However, these methods are primarily designed for industrial-grade GPUs with large memory capacity and typically perform KV cache offloading only between the GPU and the host CPU. However, on memory-constrained edge devices, the KV cache generated by long-context inference can easily exceed the combined capacity of GPU and CPU memory, necessitating offloading to external storage. For example, as shown in Figure 2a, inference with the Llama2-13B (Huang et al., 2023) model on a 4K context requires approximately 51 GB of memory, far exceeding the memory limits 44GB of a typical PC equipped with a 12 GB commodity GPU (RTX 3080) and 32 GB CPU memory.

Moreover, several studies have explored leveraging external storage to assist KV cache management (Sun et al., 2025; Sheng et al., 2023; Chen et al., 2025); however, they still suffer from substantial data transfer latency due to the limited bandwidth of conventional storage devices when performing token importance evaluation in each decoding step. For example, as shown in Figure 3, the KV data transfer latency of the SSD increases significantly with the input context length and eventually becomes the dominant bottleneck in inference latency. In summary, existing approaches are fundamentally constrained by the absence of in-storage computation support, forcing KV data to be repeatedly transferred between the SSD and host for token importance evaluation, which introduces significant and unavoidable data transfer latency.

Enhancement Opportunity. Fortunately, recent studies (Tian et al., 2024; Pan et al., 2025; Liang et al., 2022; Kim et al., 2022; Kang et al., 2013; Lee et al., 2020) have demonstrated the effectiveness of the Computational Storage Devices (CSDs, e.g., SmartSSD (Samsung Semiconductor)) in accelerating vector search and data processing by enabling in-storage computation, highlighting their easy installation and strong potential for importance-aware hardware acceleration in LLM inference. Unlike traditional SSD-centric approaches (Sun et al., 2025; Sheng et al., 2023; Chen et al., 2025) that repeatedly move data between host memory and storage devices, SmartSSDs embrace Near Data Processing (NDP) by executing computations locally on on-board DRAM and FPGAs, significantly reducing host–SSD data transfer. With CSDs, KV cache data residing on the SSD can be directly evaluated for importance using on-board FPGAs, eliminating the need to transfer this data back to the host. This in-storage importance evaluation substantially reduces data transfer latency.

Challenges. However, integrating the CSD into importance-aware KV eviction workloads is non-trivial and raises multiple algorithm- and system-level challenges. First, token importance is query-dependent (Sun et al., 2025; Lee et al., 2024; Tang et al., 2024), meaning that it varies across decoding steps. This dynamic behavior leads to frequent KV cache updates and inter-step data transfer. Under such conditions, a straightforward KV management strategy may still incur frequent ping-pong data movement between the CSD and the CPU, offsetting the benefits of in-storage computation. Thus, how to jointly manage KV data and perform importance evaluation across the CSD and CPU becomes a significant challenge. Second, although the CSD introduces in-storage computation capability, the relatively limited SSD bandwidth remains a bottleneck for data transfer. As a result, effectively coordinating computation and data transfer among GPU, CPU, and CSD, and designing an efficient system-level pipeline to minimize end-to-end latency, poses another significant challenge.

In this paper, we propose HillInfer, an efficient long-context LLM inference framework on the edge that presents a hierarchical KV cache eviction strategy and adaptive pipeline using SmartSSD, to tackle these two challenges. Specifically, to tackle the first challenge, we design a hierarchical KV cache manager with SmartSSD, which jointly manages KV cache pools on the CPU and SmartSSD. During bidirectional cache placement and eviction across these two pools, we comprehensively consider temporal locality, token importance, and cache hit rate, thereby minimizing excessive ping-pong data movement between devices. To address the second challenge, we propose a layer-wise, adaptive prefetch-based pipeline. The pipeline adaptively coordinates the proportion of importance evaluation performed on SmartSSD and CPU, enabling effective overlap between computation for layer i and KV data transfer for layer i+1, to hide importance evaluation and data transfer latency. We implement HillInfer with Python code, based on an offloading-based framework (Sheng et al., 2023), and develop HLS-based C++ code on the FPGA unit. The main contributions of this paper are summarized as follows:

(1) We propose HillInfer, to the best of our

knowledge, the first edge-based long-context LLM inference framework that enables hierarchical importance-aware KV cache eviction using SmartSSD, effectively reducing inference latency while preserving model accuracy.

(2) We design a hierarchical KV cache manager assisted by SmartSSD with bidirectional KV cache pools, along with a system-level adaptive prefetch-based pipeline, to minimize ping-pong data movement and further reduce inference latency.

(3) We implement HillInfer based on an offloading-based LLM inference framework and develop an HLS-based FPGA program on SmartSSD to support in-storage importance evaluation.

(4) We conduct extensive experiments across multiple models and datasets, demonstrating that HillInfer achieves up to 8.56 × speedup over prior-art baselines without sacrificing model accuracy.

2 Background and Motivations

2.1 LLM Inference

LLM inference consists of two phases: prefilling and decoding. In the prefilling phase, the tokenized input prompt is processed in a single forward pass to initialize the model states. During decoding, the model generates tokens autoregressively, where each newly generated token is appended to the context and used to predict the next token until an end-of-sequence token or the maximum length is reached. Modern LLMs are typically built on transformer decoders, where each layer comprises (multi-head) self-attention and a feed-forward network. Given an input sequence, hidden states are linearly projected into Query (Q), Key (K), and Value (V) tensors, and attention is applied to capture contextual dependencies. Multi-head attention further improves model capacity by attending to multiple semantic subspaces in parallel.

During autoregressive decoding, recomputing attention over all previous tokens at each step incurs quadratic complexity and leads to prohibitive latency as the context length grows. To mitigate this cost, modern LLMs adopt KV caching, which stores Key and Value tensors from previous steps and reuses them for subsequent attention computation, significantly improving decoding efficiency. However, the KV cache grows linearly with the context length and the number of decoder layers, quickly becoming the dominant memory consumer. On resource-constrained edge devices with limited GPU and host memory, this rapidly expanding KV cache becomes the primary bottleneck for long-context inference.

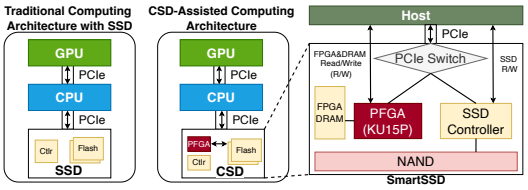


Figure 1: SSD-based Arch. vs. CSD-Assisted Arch.

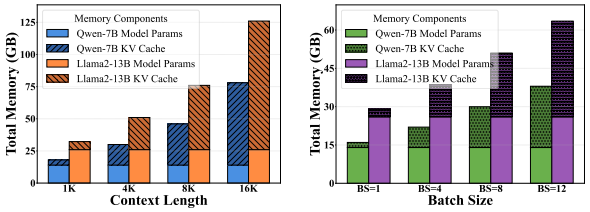
2.2 Computational Storage

A Computational Storage Device (CSD) (Samsung Semiconductor) is based on the principle that processing data close to where it is stored is more efficient than transferring it to the host for computation. By integrating computing units into storage devices, CSDs enable near-data processing (NDP), reducing data movement between storage and the host CPU and alleviating PCIe bandwidth contention. In traditional architectures, all devices share the host PCIe bus, making system performance constrained by PCIe bandwidth. In contrast, CSD-enabled systems process data locally within each storage device via internal PCIe switches, allowing better scalability as more CSDs are added.

Commercial CSDs, such as Samsung SmartSSD (Samsung Semiconductor), integrate on-board DRAM and FPGA units to support in-storage computation and data access, as illustrated in Figure 1. The on-board DRAM acts as a cache for NAND flash and can be accessed by both the FPGA and the host CPU. While the host interacts with the SmartSSD via standard I/O interfaces and can offload computation to the device, data movement between NAND flash and the FPGA is handled internally through P2P transfers over the internal PCIe switch, further reducing host involvement.

2.3 Exploring Existing Techniques

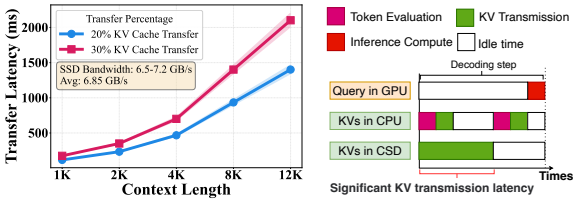
For long-context LLM inference, a series of common approaches leverage contextual sparsity to estimate token importance and selectively evict KV cache entries associated with less important tokens (Zhang et al., 2023; Lee et al., 2024; Zhao et al., 2024; Gao et al., 2024; Tang et al., 2024). However, these approaches largely rely on two-level GPU–CPU KV offloading and are evaluated on testbeds with industrial-grade GPUs and large memory capacity. On resource-constrained edge devices, as shown in Figure 2, increasing the context length or batch size can cause excessive



(a) Memory usage with batch size = 8, varying context lengths.

(b) Memory usage with context length = 4K, varying batch sizes.

Figure 2: Memory usage analysis for Qwen-7B and Llama2-13B models with full KV cache. (a) shows how memory increases with context length at a fixed batch size of 8. (b) shows how memory scales with batch size at a fixed context length of 4K.



(a) KV Cache Transfer Latency.

(b) Latency Analysis.

Figure 3: KV Cache Transfer Latency Analysis: (a) It shows the transfer time from SSD to GPU/CPU for 20% (blue) and 30% (red) of full KV cache across different context lengths for token evaluations at a batch size of 8 using Qwen-7B; (b) The illustration of the overhead of KV Cache transfer latency.

KV cache growth, leading to host out-of-memory (OOM). Even with memory pooling, recomputing evicted KV entries introduces additional computation latency and prevents effective KV eviction.

In addition, other approaches leverage external storage (e.g., SSD) to assist KV cache management (Sun et al., 2025; Sheng et al., 2023; Chen et al., 2025), but they suffer from significant data transfer latency during token importance evaluation due to the limited bandwidth of the SSD. Figure 3a shows that transfer latency increases with the context length, and becomes more pronounced as a larger fraction of the KV cache is stored on the SSD. As illustrated in Figure 3b, during each decoding step, transferring KV data between the SSD and host for token importance evaluation dominates latency and results in significant GPU idle periods.

In summary, the above existing approaches cannot be directly applied to resource-constrained edge devices for long-context LLM inference. To address this, we propose a CSD-assisted inference acceleration framework, HillInfer, which overcomes the key challenges associated with leveraging computational storage devices.

3 System Design of HillInfer

In this section, we detail the designs of HillInfer. We first describe the system architecture and workflow (Sec. 3.1), followed by the hierarchical KV cache manager with SmartSSD (Sec. 3.2). Finally, we present an adaptive prefetch-based pipeline coordinating CPU, GPU, and SmartSSD (Sec. 3.3).

3.1 System Overview

We present HillInfer, an efficient long-context LLM inference framework on the edge that leverages SmartSSD for hierarchical KV eviction and adaptive system-level pipeline.

System Framework. Figure 4 illustrates the system framework of HillInfer, which consists of a GPU, a CPU, and a SmartSSD. The GPU stores model weights, activations, and important KV data, and performs LLM inference, including prefilling and decoding stages. The CPU maintains a large portion of the KV cache and partial model weights (larger model), and is responsible for token importance evaluation, KV cache management, and pipeline coordination. The SmartSSD stores the left of KV data and leverages its on-board FPGA to perform near-data token importance evaluation.

System Workflow. In detail, as show in Figure 4, the GPU first executes the prefilling stage and caches the generated KV data on the CPU. During decoding, the KV data of newly generated tokens, together with the current query, are sent to both the CPU and the SmartSSD. The CPU and SmartSSD then independently evaluate token importance based on the current query. The CPU aggregates the evaluation results, selects important KV data, and prefetches them to the GPU for subsequent computation. Finally, the CPU and SmartSSD perform bidirectional KV cache update to maintain the KV cache pools.

3.2 Hierarchical KV Cache Manager with SmartSSD.

Hierarchical Importance Evaluation (HIE). In existing KV eviction schemes, importance evaluation is typically performed on the CPU, which requires transferring KV data from other devices to the CPU, incurring significant data movement latency. HillInfer instead leverages the FPGA on the SmartSSD to perform near-data importance evaluation, thereby reducing unnecessary KV transfers. However, such hierarchical importance evaluation needs to coordinate computation and data move-

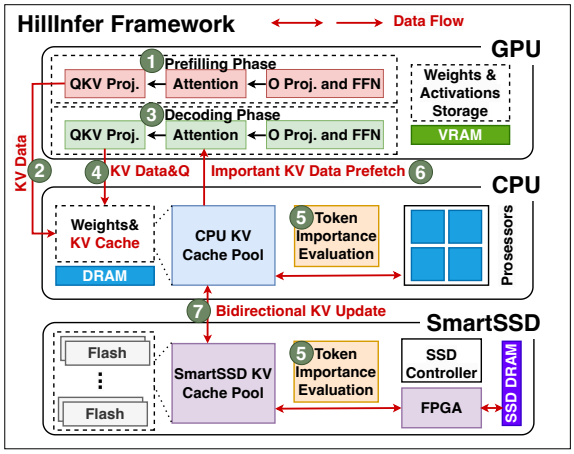


Figure 4: The Framework Overview of HillInfer.

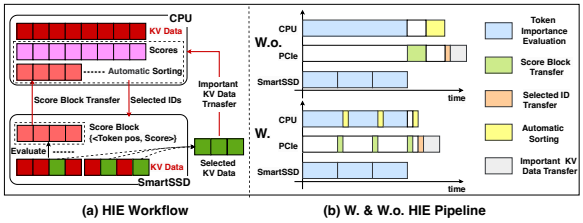


Figure 5: The Workload and Pipeline of HIE.

ment between the CPU and the SmartSSD.

As illustrated in Figure 5 (a), the SmartSSD and the CPU perform token importance evaluation in parallel. After evaluating a batch of n tokens, the SmartSSD aggregates their importance scores into a Score Block, represented as $<Token\ pos,\ Score>$, and immediately transfers the block to the CPU. The CPU incrementally merges and sorts the received scores with its local results. Once all tokens are evaluated, the CPU selects important token IDs and requests the corresponding KV data from the SmartSSD for merging. As illustrated in Figure 5 (b), without HIE, merging the token importance scores computed by the SmartSSD and the CPU, as well as transferring the selected important KV data, would introduce substantial additional latency. HIE hides the latency of Score Blocks transfer and merging by overlapping them with parallel computation on both devices, thereby significantly reducing the overall token importance evaluation overhead.

Bidirectional KV Cache Pools (BKP). Since token importance is query-dependent at each decoding step (Sun et al., 2025; Lee et al., 2024; Tang et al., 2024), naive eviction strategies often trigger excessive ping-pong data movement. To address this, HillInfer maintains two hierarchical KV cache pools on the CPU and the SmartSSD (Figure 4), and performs bidirectional KV pool update after each decoding step to maintain stable KV placement across devices. Our goal is to minimize the fraction of important KV data residing on the SmartSSD, thereby avoiding frequent ping-pong data movement between the CPU and SmartSSD for KV data that is repeatedly identified as important across decoding steps.

Specifically, besides token importance, we also consider temporal locality and KV cache hit rate. First, within a decoding step, tokens closer to the current step tend to be more important (Zhao et al., 2024). We therefore keep the KV cache of the most recent $\alpha \cdot N$ steps in the CPU, avoiding transferring it to the SmartSSD, where N denotes the context length, and α is an importance rate hyperparameter set by the user (typically $10\% \sim 20\%$ in our experiment). Second, some tokens may not exhibit strong importance but have a high cache hit rate. To prevent frequent ping-pong data movement, we maintain a KV cache hit rate table and keep the top-α fraction of KV data in the CPU, avoiding their transfer to the SmartSSD. Finally, during the maintenance of the cache hit rate table, if any of the top-α fraction of KV data reside on the SmartSSD, they are transferred back to the CPU. To maintain balance, an equal number of the lowest-ranked KV data in the table are simultaneously moved from the CPU to the SmartSSD.

3.3 Adaptive Prefetch-based Pipeline

Inspired by layer-wise KV data prefetching techniques (Lee et al., 2024; Sun et al., 2025), HillInfer overlaps importance evaluation and KV prefetching for layer $i{+}1$ while the GPU computes layer i. Despite its benefits, when traditional SSD-based KV eviction schemes are deployed on resource-constrained devices, KV data transfers between the SSD and CPU incur significant transmission latency, leading to substantial GPU idle time, as shown in Figure 6 (a). Although SmartSSD can reduce KV transfer latency by enabling in-storage computation (Figure 6 (b)), naively determining the capacity ratio of KV cache pools on the CPU and SmartSSD still increases the importance evaluation latency and GPU idle time.

To address the above issues, we propose an Adaptive Prefetch-based Pipeline (APP), as illustrated in Figure 6 (c). APP adaptively determines the capacity ratio of the two KV cache pools to effectively hide KV prefetch latency and reduce GPU idle time. Assume that the total amount of KV data

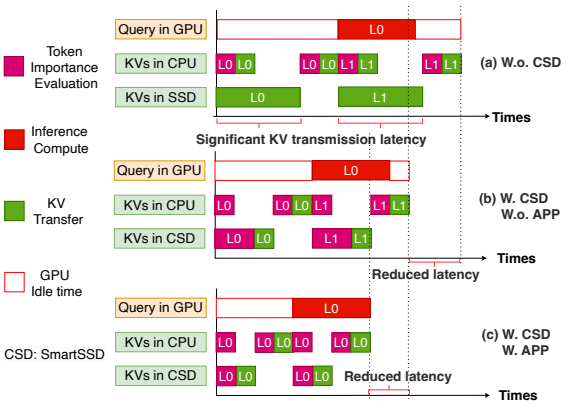


Figure 6: The Latency Comparison without CSD and with CSD and APP Technique.

is M. Let f_c and f_s denote the KV data processing average throughput (often limited by memory bandwidth) of the CPU and the SmartSSD, respectively, and let M_c, M_0, and M_s be the amounts of KV data stored in the CPU, the available CPU memory threshold, and SmartSSD cache pools, where f_c and f_s are obtained through offline profiling. To minimize the GPU idle time, it suffices to configure the cache capacity ratio β such that

$$\frac{M_c}{f_c} \approx \frac{M_s}{f_s},\ M_c < M_0 \tag{1}$$

which equivalently yields $\beta \approx \frac{f_c}{f_s}$. This condition ensures that token importance evaluation on the CPU and SmartSSD can be completed in a nearly synchronized manner, allowing GPU computation to effectively overlap both token importance evaluation and KV data prefetching, thereby minimizing GPU idle time.

4 Implementation and Experiments

4.1 Implementation

We implement HillInfer by extending the offloading-based LLM inference framework Flex (Sheng et al., 2023) with 500+ lines of additional Python code, while incorporating some codes about data movement from the prefetch-based framework InfiniGen (Lee et al., 2024). Furthermore, we developed HLS-based C++ code to implement token-level KV cache importance evaluation on the FPGA units of the SmartSSD. Host–SmartSSD communication is implemented using the Xilinx Runtime (XRT) interface (Xilinx, Inc.). Our preliminary experiments are built upon the Hugging Face Transformers framework (Wolf et al., 2019), and HillInfer can be readily applied to accelerate most open-source LLMs, such as LLaMA (Huang et al., 2023), Qwen (Bai et al., 2023), and OPT, etc., without requiring modifications to model architectures.

4.2 Experiments Setting

Hardware. To closely reflect realistic PC deployment scenarios, we conduct our experiments on a Ubuntu 22.04 desktop system equipped with an NVIDIA GeForce RTX 4090 GPU, an Intel(R) Xeon(R) Platinum 8352V CPU @ 2.10GHz, 24 GB of GPU memory, 64 GB of host memory, and a 300 GB Intel SSD connected via a PCIe 4.0 interface. We further employ a Samsung SmartSSD as the computational storage device with Xilinx's UltraScale+ FPGA, 4 GB DDR4, and 4 TB NAND Flash, providing a peak bandwidth of approximately 4 GB/s, which is connected to the motherboard through PCIe, as illustrated in Figure 7.

Figure 7: Hardware Architecture of HillInfer.

Models. To evaluate the adaptability of HillInfer across different model architectures and scales, we conduct experiments using models from the LLaMA family (Huang et al., 2023) (LongChat 7B and LLaMA 13B), Qwen-7B (Bai et al., 2023), and OPT-6.7B. We set the output length as 128 when inferring these models without special statement.

Datasets. For evaluating model inference accuracy, we use several few-shot tasks from the LM-evaluation-harness benchmark (Gao et al., 2021), including OpenBookQA, RTE, PIQA, COPA, and PG-19 (Rae et al., 2019). To assess long-context inference latency, we conduct experiments using LongBench (Bai et al., 2024).

Baselines. We compare HillInfer against four baseline approaches. (1) Full Cache: This baseline retains the entire KV cache without performing any token importance evaluation and is commonly treated as the accuracy-test baseline. (2) H2O-like: This approach adopts a token importance evaluation strategy similar to H2O (Zhang et al., 2023) and employs an offloading-based mechanism for KV cache transfer. (3) Prefetch-based (Lee et al., 2024): This baseline follows an InfiniGen-style design, performing importance evaluation and

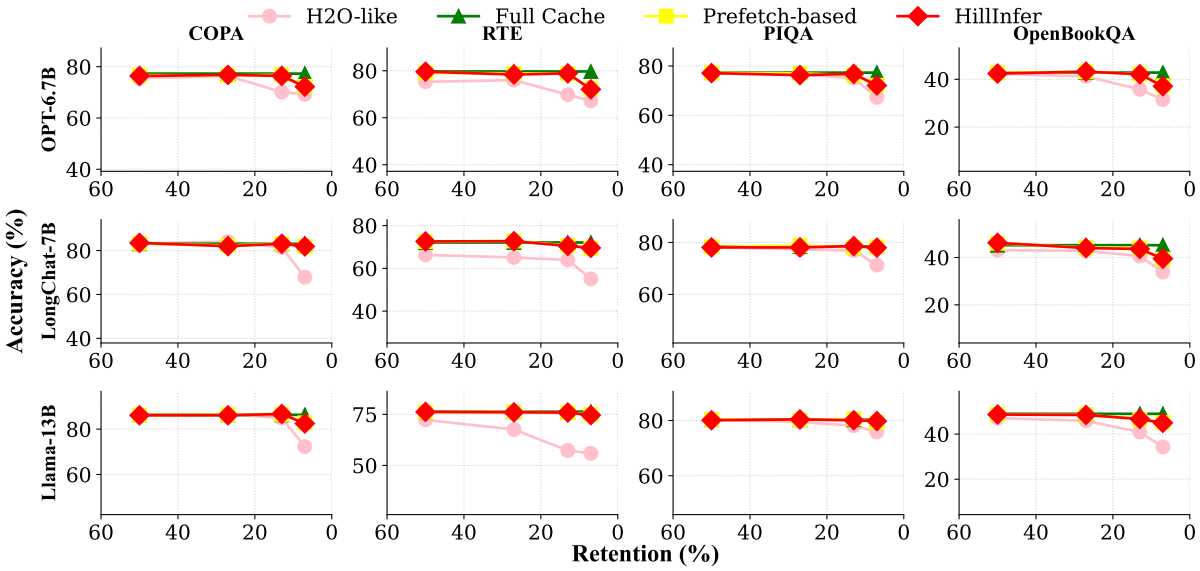


Figure 8: The accuracy comparison of different frameworks across four datasets and three models under different relative KV Cache sizes.

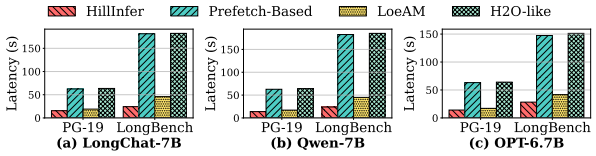


Figure 9: The Inference Latency Comparison with Different Models on PG-19 and LongBench.

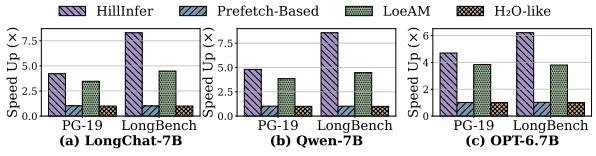


Figure 10: The Speed Up Comparison with Different Models on PG-19 and LongBench.

prefetching layer-wise KV data from the external storage and host memory. (4) LeoAM-like: This approach applies the LeoAM method (Sun et al., 2025) for token importance evaluation and leverages the KV-Abstract to optimize KV data transfer.

Comparison metics. For model accuracy evaluation, we use Accuracy (%) as the metric. End-to-end inference performance is measured using Latency (ms). To assess throughput and acceleration effectiveness, we report Speedup (×), respectively. All evaluation metrics follow those adopted in prior work (Lee et al., 2024; Chen et al., 2025; Sun et al., 2025). We set the importance rate α as 0.2 and batch size = 8 in our experiment.

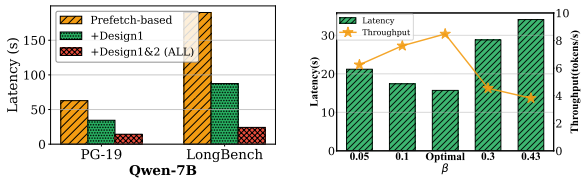
4.3 Main Results

Model Accuracy. Since LeoAM and prefetch-based methods achieve comparable accuracy (Sun et al., 2025), for simplicity, we evaluate the accuracy of HillInfer across different models and datasets against Full Cache, H2O-like, and prefetch-based methods. Our results show that HillInfer maintains comparable accuracy to these baselines, as shown in Figure 8. This demonstrates that, while leveraging SmartSSD for inference acceleration, HillInfer carefully designs BKP and APP to manage KV data offloading rather than incorrectly evicting KV data, thereby avoiding any degradation in model accuracy. In addition, we observe that setting the importance rate between 10% and 20% generally yields a favorable trade-off between performance and accuracy.

End-to-end Latency and Throughput. We evaluate the end-to-end latency and throughput of different models and batch sizes using LongBench. Compared to the baselines, we find that HillInfer reduces end-to-end latency by $76.25\%\tilde{}88.32\%$, as shown in Figure 9. We also observe that HillInfer achieves a speedup of $4.21 \sim 8.56\times$, as shown in Figure 10, with more pronounced speed-up gains on datasets with longer context lengths.

4.4 System Analysis

Ablation Study. Figure 11a illustrates the individual contributions of Design 1 (HIE&BKP, Sec. 3.2) and Design 2 (Sec. 3.3) to HillInfer. Compared to the prefetch-based method, +Design1 and Design1&2 achieve varying degrees of inference latency reduction, indicating that both designs contribute to lowering inference latency.



(a) Ablation Study with LongBench & PG-19 on Qwen-7B.

(b) Sensitive Analysis with PG-19 on Qwen-7B.

Figure 11: Ablation Study and Sensitive Analysis.

Sensitive Analysis. Figure 11b shows the latency variation under different values of β. We observe that the latency is minimized when $\beta \approx \frac{f_c}{f_s}$ in this experiment setting, which is consistent with the design rationale of APP.

Overhead Analysis. In HIE, HillInfer transfers Score Blocks from the SmartSSD to the CPU. Each score block only occupies $2n$ half-precision floating-point values (i.e., $4n$ bytes), which is negligible compared to the memory footprint of the KV cache. As a result, its transfer latency can be fully hidden by computation. In BKP, HillInfer maintains a cache hit table with a size of $2N$ bytes, which is negligible compared to the memory footprint of the KV cache.

5 Related Work

Long-context LLM Inference Framework. A large number of recent works study long-context LLM serving in data center networks, with an emphasis on resource scheduling and KV cache management to improve system throughput and inference latency (Agrawal et al., 2024; Qin et al., 2025; Hooper et al., 2024; Lim et al., 2024; Lin et al., 2024; Wu et al., 2024). Many existing works also focus on long-context LLM inference on single-node GPU with large memory capacity (e.g., A100), demonstrating that inference latency can be significantly reduced through algorithm–system co-design (Liu et al., 2024; Pan et al., 2025; Yao et al., 2025; Zhao et al., 2024; Zhang et al., 2025). However, for edge devices with limited resources, long-context LLM inference is fundamentally constrained by memory capacity. Some prior works explore leveraging external memory, such as SSDs, to accelerate inference (Sheng et al., 2023; Sun et al., 2025; Chen et al., 2025). Nevertheless, the relatively low bandwidth of the SSD means that frequent data transfers between SSD and host memory become the bottleneck in end-to-end latency.

KV data eviction for LLM Inference. To address the memory and computation constraints in long-context inference, many prior works exploit the inherent sparsity of long-context inputs. During the decoding phase, these methods evaluate the importance of each token and selectively retain the KV data of important tokens on the GPU for subsequent inference, while evicting less important KV data to the CPU or dropping it entirely (Zhang et al., 2023; Zhao et al., 2024; Tang et al., 2024; Lin et al., 2024; Sun et al., 2025; Gao et al., 2024; Liu et al., 2023; Juravsky et al.; Ye et al., 2024; Zheng et al., 2024). However, directly applying these methods on edge devices often faces fundamental challenges: the limited memory capacity may lead to out-of-memory (OOM) errors, while the low bandwidth of external storage makes frequent KV data transfers the dominant bottleneck to end-to-end inference latency.

LLM optimization using CSD. Recently, several studies have focused on employing CSDs, e.g., SmartSSDs, to accelerate LLM-related workloads such as vector search (Tian et al., 2024; Liang et al., 2022; Kim et al., 2022; Niu et al., 2024) and data processing (Kang et al., 2013; Lee et al., 2020; Soltaniyeh et al., 2022; Salamat et al., 2021), and inference (Pan et al., 2025; Deng et al., 2025; Duan et al., 2025). Although prior work has demonstrated the benefits of CSDs, supporting long-context LLM inference on edge devices remains challenging due to excessive ping-pong data movement and pronounced latency bottlenecks.

In summary, HillInfer addresses the limitations of prior approaches. To the best of our knowledge, HillInfer is the first framework to enable efficient long-context LLM inference on edge devices by integrating CSDs into importance-aware KV eviction, and achieves substantial inference speedup without sacrificing model accuracy

6 Conclusion

In this paper, we presented HillInfer, an importance-aware long-context LLM inference framework for edge devices that leverages SmartSSD-assisted hierarchical KV cache management. By combining in-storage importance evaluation with an adaptive prefetch-based pipeline, HillInfer effectively reduces data movement and inference latency under tight memory constraints. Experiments on a PC with a consumer-grade GPU show that HillInfer achieves up to 8.56 × speedup over state-of-the-art baselines while maintaining model accuracy.

Limitations

While HillInfer effectively accelerates long-context LLM inference on edge devices without sacrificing accuracy, several limitations remain and open directions for future work. First, the current implementation targets a single CSD platform, namely Samsung SmartSSD. Future work could extend HillInfer to support a broader range of CSD products, such as those from ScaleFlux, Eideticom, and NVXL, to improve portability and generality. Second, although HillInfer demonstrates significant speedups on PCs equipped with a commodity GPU, extending the framework to Arm-based edge platforms, such as NVIDIA Jetson devices, is an important next step. Finally, as modern LLMs increasingly emphasize complex reasoning capabilities, future work may explore leveraging CSDs to accelerate reasoning-intensive LLM workloads. Supporting a wider range of devices and workloads would further broaden the applicability of CSD-assisted long-context LLM inference in resource-constrained devices.

References

Amey Agrawal, Haoran Qiu, Junda Chen, Íñigo Goiri, Chaojie Zhang, Rayyan Shahid, Ramachandran Ramjee, Alexey Tumanov, and Esha Choukse. 2024. Medha: Efficiently serving multi-million context length llm inference requests without approximations. *arXiv preprint arXiv:2409.17264*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, and 1 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, and 1 others. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 3119–3137.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Fenglong Cai, Dong Yuan, Zhe Yang, and Lizhen Cui. 2024. Edge-llm: A collaborative framework for large language model serving in edge computing. In *2024 IEEE International Conference on Web Services (ICWS)*, pages 799–809. IEEE.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, and 1 others. 2024. A survey on evaluation of large language models. *ACM transactions on intelligent systems and technology*, 15(3):1–45.

Weijian Chen, Shuibing He, Haoyang Qu, Ruidong Zhang, Siling Yang, Ping Chen, Yi Zheng, Baoxing Huai, and Gang Chen. 2025. {IMPRESS}: An {Importance-Informed}{Multi-Tier} prefix {KV} storage system for large language model inference. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pages 187–201.

Lishuo Deng, Shaojie Xu, Jinwu Chen, Changwei Yan, Jiajie Wang, Zhe Jiang, and Weiwei Shan. 2025. Kvnand: Efficient on-device large language model inference using dram-free in-flash computing. *arXiv preprint arXiv:2512.03608*.

Zhuohui Duan, Hao Feng, Haikun Liu, Xiaofei Liao, Hai Jin, and Bangyu Li. 2025. {AegonKV}: A high bandwidth, low tail latency, and low storage cost {KV-Separated}{LSM} store with {SmartSSD-based}{GC} offloading. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pages 321–335.

Othmane Friha, Mohamed Amine Ferrag, Burak Kantarci, Burak Cakmak, Arda Ozgun, and Nassira Ghoualmi-Zine. 2024. Llm-based edge intelligence: A comprehensive survey on architectures, applications, security and trustworthiness. *IEEE Open Journal of the Communications Society*.

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 *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 111–126.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, and 1 others. 2021. A framework for few-shot language model evaluation. *Zenodo*.

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.

Coleman Richard Charles Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, Sebastian Zhao, June Paik, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. 2025. Squeezed attention: Accelerating long context length llm inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32631–32652.

Quzhe Huang, Mingxu Tao, Chen Zhang, Zhenwei An, Cong Jiang, Zhibin Chen, Zirui Wu, and Yansong

Feng. 2023. Lawyer llama technical report. *arXiv preprint arXiv:2305.15062*.

Berivan Isik, Natalia Ponomareva, Hussein Hazimeh, Dimitris Paparas, Sergei Vassilvitskii, and Sanmi Koyejo. 2024. Scaling laws for downstream task performance of large language models. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*.

Jordan Juravsky, Bradley Brown, Ryan Saul Ehrlich, Daniel Y Fu, Christopher Re, and Azalia Mirhoseini. Hydragen: High-throughput llm inference with shared prefixes. In *Workshop on Efficient Systems for Foundation Models II@ ICML2024*.

Yangwook Kang, Yang-suk Kee, Ethan L Miller, and Chanik Park. 2013. Enabling cost-effective data processing with smart ssd. In *2013 IEEE 29th symposium on mass storage systems and technologies (MSST)*, pages 1–12. IEEE.

Ji-Hoon Kim, Yeo-Reum Park, Jaeyoung Do, Soo-Young Ji, and Joo-Young Kim. 2022. Accelerating large-scale graph-based nearest neighbor search on a computational storage platform. *IEEE Transactions on Computers*, 72(1):278–290.

Joo Hwan Lee, Hui Zhang, Veronica Lagrange, Praveen Krishnamoorthy, Xiaodong Zhao, and Yang Seok Ki. 2020. Smartssd: Fpga accelerated near-storage data analytics on ssd. *IEEE Computer architecture letters*, 19(2):110–113.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. {InfiniGen}: Efficient generative inference of large language models with dynamic {KV} cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 155–172.

Omer Levy and Yoav Goldberg. 2014. Linguistic regularities in sparse and explicit word representations. In *Proceedings of the eighteenth conference on computational natural language learning*, pages 171–180.

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

Shengwen Liang, Ying Wang, Ziming Yuan, Cheng Liu, Huawei Li, and Xiaowei Li. 2022. Vstore: in-storage graph based vector search accelerator. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 997–1002.

Hwijoon Lim, Juncheol Ye, Sangeetha Abdu Jyothi, and Dongsu Han. 2024. Accelerating model training in multi-cluster environments with consumer-grade gpus. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 707–720.

Bin Lin, Chen Zhang, Tao Peng, Hanyu Zhao, Wencong Xiao, Minmin Sun, Anmin Liu, Zhipeng Zhang, Lanbo Li, Xiafei Qiu, and 1 others. 2024. Infinite-llm: Efficient llm service for long context with distattention and distributed kvcache. *arXiv preprint arXiv:2401.02669*.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, and 1 others. 2024. 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.

Zhenyan Lu, Xiang Li, Dongqi Cai, Rongjie Yi, Fangming Liu, Xiwen Zhang, Nicholas D Lane, and Mengwei Xu. 2024. Small language models: Survey, measurements, and insights. *arXiv preprint arXiv:2409.15790*.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196*.

Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. 2025. A comprehensive overview of large language models. *ACM Transactions on Intelligent Systems and Technology*, 16(5):1–72.

Fuping Niu, Jianhui Yue, Jiangqiu Shen, Xiaofei Liao, and Hai Jin. 2024. Flashgnn: An in-ssd accelerator for gnn training. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 361–378. IEEE.

Olaide N Oyelade, Hui Wang, and Karen Rafferty. 2025. A survey of adaptation of large language models to idea and hypothesis generation: Downstream task adaptation, knowledge distillation approaches and challenges. *ACM Computing Surveys*.

Xiurui Pan, Endian Li, Qiao Li, Shengwen Liang, Yizhou Shan, Ke Zhou, Yingwei Luo, Xiaolin Wang, and Jie Zhang. 2025. Instattention: In-storage attention offloading for cost-effective long-context llm inference. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2025. Mooncake: Trading more storage for less computation—a {KVCache-centric} architecture for serving {LLM} chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pages 155–170.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*.

Sahand Salamat, Armin Haj Aboutalebi, Behnam Khaleghi, Joo Hwan Lee, Yang Seok Ki, and Tajana Rosing. 2021. Nascent: Near-storage acceleration of database sort on smartssd. In *The 2021 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 262–272.

Samsung Semiconductor. Smartssd-samsung semiconductor. https://semiconductor.samsung.com/ssd/smart-ssd/.

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.

Mohammadreza Soltaniyeh, Veronica Lagrange Moutinho Dos Reis, Matt Bryson, Xuebin Yao, Richard P Martin, and Santosh Nagarakatte. 2022. Near-storage processing for solid state drive based recommendation inference with smartssds®. In *Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering*, pages 177–186.

He Sun, Li Li, Mingjun Xiao, and Chengzhong Xu. 2025. Breaking the boundaries of long-context llm inference: Adaptive kv management on a single commodity gpu. *arXiv preprint arXiv:2506.20187*.

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 *Proceedings of the 41st International Conference on Machine Learning*, pages 47901–47911.

Bing Tian, Haikun Liu, Zhuohui Duan, Xiaofei Liao, Hai Jin, and Yu Zhang. 2024. Scalable billion-point approximate nearest neighbor search using {SmartSSDs}. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 1135–1150.

Chunlin Tian, Xinpeng Qin, Kahou Tam, Li Li, Zijian Wang, Yuanzhe Zhao, Minglei Zhang, and Chengzhong Xu. 2025. Clone: Customizing llms for efficient latency-aware inference at the edge. *USENIX ATC*.

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.

Colin Wei, Sang Michael Xie, and Tengyu Ma. 2021. Why do pretrained language models help in downstream tasks? an analysis of head and prompt tuning. *Advances in Neural Information Processing Systems*, 34.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and 1 others. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

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.

Xilinx, Inc. XRT - Xilinx Runtime Library. https://www.xilinx.com/products/design-tools/vitis/xrt.html.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. 2025. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 94–109.

Lu Ye, Ze Tao, Yong Huang, and Yang Li. 2024. Chunkattention: Efficient self-attention with prefix-aware kv cache and two-phase partition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11608–11620.

Wangsong Yin, Mengwei Xu, Yuanchun Li, and Xuanzhe Liu. 2026. Llm as a system service on mobile devices. *SenSys*.

Zhongzhi Yu, Zheng Wang, Yuhan Li, Ruijie Gao, Xiaoya Zhou, Sreenidhi Reddy Bommu, Yang Zhao, and Yingyan Lin. 2024. Edge-llm: Enabling efficient large language model adaptation on edge devices via unified compression and adaptive layer voting. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6.

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

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, and 1 others. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2).

Youpeng Zhao, Di Wu, and Jun Wang. 2024. Alisa: Accelerating large language model inference

via sparsity-aware kv caching. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 1005–1017. IEEE.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, and 1 others. 2024. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583.

A Appendix

In the appendix, we display the Notation Description (TABLE 1) and the details of the HIE algorithm (Algorithm 1), where $\mathcal{S}_{CPU}$ is the score list, $\mathcal{B}_i$ is the i-th token block, S_i is the i-th score block, $\mathcal{T}_{imp}$ are the important token IDs.

Table 1: Notation Summary

Symbol	Description
n	Size of a score block (number of tokens per block)
N	Context length (sum of input and generated tokens)
α	Importance rate controlling the fraction of important KV data
β	Size ratio of KV cache pools between CPU and SmartSSD
M_0	Available memory capacity on the CPU
M_c	KV cache pool size allocated on the CPU
M_s	KV cache pool size allocated on the SmartSSD
f_c	Average KV data processing throughput of the CPU
f_s	Average KV data processing throughput of the SmartSSD

Algorithm 1 Hierarchical Importance Evaluation (HIE)

Require: Context length N; score block size n; importance rate α
Ensure: Set of important token IDs $\mathcal{T}_{imp}$

Initialize empty score list $\mathcal{S}_{CPU} \leftarrow \emptyset$
Partition tokens into blocks $\{\mathcal{B}_1, \mathcal{B}_2, \dots\}$, each of size n
for all score blocks $\mathcal{B}_i$ **in parallel do**
    SmartSSD:
        Compute importance scores for tokens in $\mathcal{B}_i$
        Aggregate scores into Score Block $\mathcal{S}_i \leftarrow \{\langle \text{token\_id}, \text{score} \rangle\}$
        Transfer $\mathcal{S}_i$ to CPU asynchronously
    CPU:
        Compute local importance scores for assigned tokens
        Incrementally merge received $\mathcal{S}_i$ into $\mathcal{S}_{CPU}$
        Maintain partial sorted order of $\mathcal{S}_{CPU}$
end for
Sort $\mathcal{S}_{CPU}$ by importance score in descending order
Select top $\alpha \cdot N$ token IDs as $\mathcal{T}_{imp}$
Request KV data of $\mathcal{T}_{imp}$ from SmartSSD
Merge received KV data into CPU KV cache pool
return $\mathcal{T}_{imp}$