StoreLLM: Energy Efficient Large Language Model Inference with Permanently Pre-stored Attention Matrices

Dan Wang
The Hong Kong Polytechnic Univeristy
Kowloon, Hong Kong
dan.wang@polyu.edu.hk

Boan Liu
The Hong Kong Polytechnic Univeristy
Kowloon, Hong Kong
bo-an.liu@connect.polyu.hk

Rui Lu
The Hong Kong Polytechnic University
Kowloon, Hong Kong
csrlu@comp.polyu.edu.hk

Zhaorui Zhang
The Hong Kong Polytechnic Univeristy
Kowloon, Hong Kong
zhaorui.zhang@polyu.edu.hk

Shuntao Zhu
The Hong Kong Polytechnic Univeristy
Kowloon, Hong Kong
shun-tao.zhu@connect.polyu.hk

Abstract

Energy efficiency has become an important design issue in Large Language Model (LLM) inference systems. The main energy consumption goes to computing. There are studies to reduce computing or to conduct computing in regions with green energy. In this paper, we study an orthogonal perspective. We observe that the attention matrices of the tokens remain largely unchanged across different LLM inference. We argue that there can be over-computing of the attention matrices across different LLM inference in LLM inference systems. As the energy of computing is substantially greater than the energy of storage access, we propose StoreLLM, an LLM inference system where the attention matrices of tokens are pre-stored so that the computing of the attention matrices in any LLM inference can be substituted by storage access. Our analysis shows that it is possible to permanently pre-store the attention matrices of all tokens in storage, and we develop mechanisms to effectively maintain the LLM inference performance. Our evaluation shows that StoreLLM can outperform state-of-the-art LLM inference systems LazyLLM by 1.45× in energy consumption with a sacrifice of 5.05% in delays. With further improvements, StoreLLM-MoE and StoreLLM-PTQ can achieve 2.64× and 2.83× energy reduction as compared to state-of-the-art LLM systems.[1]

CCS Concepts

• **Computing methodologies** → *Natural language generation.*

Keywords

Large Language Model, KV Cache, Hierarchy Storage System

ACM Reference Format:
Dan Wang, Boan Liu, Rui Lu, Zhaorui Zhang, and Shuntao Zhu. 2025. StoreLLM: Energy Efficient Large Language Model Inference with Permanently Pre-stored Attention Matrices. In *The 16th ACM International Conference on Future and Sustainable Energy Systems (E-ENERGY '25), June 17–20, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3679240.3734604

[1]The source code can be obtained from https://github.com/StoreLLM/StoreLLM/

1 Introduction

The increasing energy consumption of large language models (LLM) has drawn much attention [8][12]. In particular, LLM inference produces 25× more carbon emissions than training (GPT-3) [3]. In an LLM inference system, an LLM takes prompts as inputs and outputs the inference results for each prompt. Specifically, the LLM inference computing consists of a *Prefill* stage and a *Decode* stage. In the Prefill stage, the LLM processes the input tokens of a prompt and computes the intermediate attention matrices, i.e., the KVQ matrices, for each token. The attention matrices are then fed into feed-forward networks (FFNs) to generate the "first" output token. In the Decode stage, the LLM uses the cached attention matrices to generate output tokens auto-regressively, until a stopping criterion is met. The Prefill stage dominates LLM computing.

There are many studies to reduce LLM inference computing. KV Cache [17] is a widely used technique that reuse previously computed KV in decoding stages. LazyLLM [11] observes that in many scenarios, certain input tokens are unimportant to the LLM inference results and develops a metric to skip the computing of the unimportant tokens. Activation-aware Weight Quantization (AWQ) [18] observes that high-bit weight tensors could be redundant and develops a post-training quantization method to project high-bit weight tensors to low-bits to reduce the model size in the LLM inference. There are also studies focusing on carbon reduction [21], e.g., CarbonMin relocates the workloads to the locations with low carbon power [4]. In this paper, we study an orthogonal perspective. We observe that, given an LLM, the attention matrices, specifically the KVQ matrices, of tokens remain largely unchanged across different LLM inference. For example, for two prompts, e.g., "Dogs are friends of humankind" and "Dogs bark", the KVQ matrices "Dogs" are highly similar in the LLM inference computing of these two prompts. Intrinsically, the KVQ matrices for the token "Dogs" in these two prompts rely only on their embeddings sent into the computing of KVQ matrices, which are identical.[2] We argue that there

[2]Intuitively, the KVQ matrices are the "dictionary" of words/tokens. For the same words/tokens, their "meaning" remains largely unchanged across sentences/prompts.

is an over-computing of the attention matrices, and we propose a system that can pre-store the attention matrices across the LLM inference. With pre-stored attention matrices, the computing of the attention matrices during LLM inference can be substituted by storage access. This can reduce the energy of LLM inference since the energy of computing (per calculation) is substantially greater than that of storage access (per read/write) by 20× to 100× [16].

We observe three challenges in developing LLM inference systems with attention pre-storage: (1) the effectiveness depends on the amount of tokens where the attention matrices are pre-computed and stored. Storing the attention matrices of *all* tokens will allow the attention matrices *never* to be recomputed. Yet, this may request massive storage; (2) storage access can be much slower than computing. As such, it is a challenge to maintain the LLM inference performance, and (3) LLM inference has many computing components besides attention matrices; and the potential improvement of the attention pre-storage should be clarified.

First, we argue that storing the attention matrices of all tokens is acceptable. For example, the prompt trace WikiText-103 [22] has 100 million tokens, yet the distinct tokens are 267,735. The storage requirement is 257GB for the attention matrices of these distinct tokens. For BookCorpus [37], there are over 1.3 million distinct tokens, requiring storage of 1.2TB. Pre-storing all tokens can enjoy the reuse and maximally avoid recomputing. This can largely amortize the pre-computation energy. For example, the average recurrence of the tokens of WikiText-103 is 380.53.

Second, storing attention matrices requires disk space. To improve the computing performance, we develop two mechanisms: (1) a *Hierarchical Cache Model:* to cache the *high-frequency tokens* in DRAM. We select the top tokens from the Corpus of Contemporary American English (COCA) [6]. These tokens can cover the majority of *token access* regardless of whether the trace is Wikitext-103 or BookCorpus and (2) a *Computing-storage scheduler:* given delay requirements, to re-compute some *low-frequency tokens* stored in the disk to reduce the completion delay of the prompts. We develop a fully functional StoreLLM. It has an offline profiling phase and a runtime inference phase. In offline, we profile computing components of LLM inference, e.g., KVQ-Linear, FFN, etc., in terms of their computing time, storage access time, and the respective energy consumed on specific pieces of computer hardware. We use the AccelWattch model [15] to profile the energy in computing and the model in [23] to profile the energy in storage access. We use Powenetics v2 [5], a Power Measurement Device, to calibrate the energy profiles. In runtime, the KVQ matrices are accessed from storage or recomputed following our scheduler.

Third, our evaluation shows that computing the attention matrices and FFN accounts for 35% and 50% of the energy consumption of the LLM computing. FFN parses the relation of a word/token within a sentence/prompt and is prompt-dependent. Thus, FFN has to be computed for each prompt. FFN computing has redundancy, and many mechanisms have been developed to compress FFN. To improve the potential energy reduction, we employ two mechanisms: (1) StoreLLM-MoE, where we employ the Mixture of Experts (MoE) [7] for FFN and MoEfication[35] to activate a part of FFN and (2) StoreLLM-PTQ, where we apply quantization [18] to FFN.

Our evaluation shows that StoreLLM achieves an energy saving (Joule per token) of 1.45× as compared to state-of-the-art LLM inference systems, LazyLLM [11], with a slight delay increase of 2.87% while maintaining 96.87% accuracy performance. With FFN compression mechanisms, StoreLLM-MoE and StoreLLM-PTQ show energy savings of 2.64×, 2.83× with over 86.27% accuracy. We plan to open source StoreLLM once it goes public.

2 Related Work

Sustainable computing has attracted many studies, e.g., energy accounting [2, 9] and energy-aware computing [25, 31, 32] for various computer systems and applications.

Our work falls into energy-aware computing for a key energy-consuming application, LLM inference. We present a new observation on energy overrun by over-computing the attention matrices in LLM inference and a new solution using storage access to substitute computing. We comment that using cache to accelerate computing performance is not new. Specifically in LLM inference, the KVQ matrices are cached in the Decode phase, and CacheGen [19], Q-Hitter [36] also cache KVQ matrices to accelerate LLM. Yet all these studies only cache the KVQ matrices for the *current* prompt, and the KVQs have to be recomputed for a new prompt. There are only a few hundred tokens in a prompt with limited recurrence. As a comparison, we store the KVQ matrices of all tokens. Our storage works across prompts that can reach hundreds of millions of tokens.

We comment that there are also studies showing that the energy of storage systems is also significant, and proposed solutions [26][13]. We are orthogonal since StoreLLM uses storage to substitute computing, which has orders more energy consumption.

3 StoreLLM Design

3.1 Overview

The overall **problem** of StoreLLM is that given a well-trained LLM and a computing server with GPU/CPU and storage DRAM/SSD; and given a prompt and a delay requirement for this prompt, StoreLLM computes or reads attention matrices of the tokens and executes LLM inference, to minimize the energy consumption. The energy consumption is measured by *energy per token*, and the delay is measured as the time-to-first-token (TTFT) [29].

StoreLLM pre-computes and stores the attention matrices of all tokens in disk storage (SSD). To support the delay requirements, StoreLLM employs two mechanisms: (1) StoreLLM selects a small number of *high-frequency tokens* and employs a *hierarchical cache model* (§3.2) to store their attention matrices in DRAM for fast access and (2) for the majority *low-frequency tokens*, StoreLLM has a *compute-storage scheduler* to recompute the attention matrices when necessary to ensure the LLM inference delay requirements.

StoreLLM consists of an offline profiling phase and a runtime inference phase. The offline Profilers (§3.3) profile the LLM inference components, i.e., the KVQ-Linears, FFNs, etc., in terms of their computing time, storage access time, and the respective energy. In runtime (§3.4), StoreLLM first accesses DRAM for the KVQ matrices of a token. If there is a miss, StoreLLM applies a *compute-storage scheduler* to determine whether to re-compute the KVQ matrices or to access the KVQ matrices from disk storage.

3.2 The Hierarchical Cache Model

StoreLLM employs a two-level hierarchical cache. Specifically, we allocate a portion of the DRAM to store the KVQ matrices for high-frequency tokens. Linguistics shows that word frequency adheres to Zipf's Law [24]. Following COCA [6], this corpus represents American English and is prompt-agnostic. We select the top-K_{DRAM} words/tokens from this corpus. Let S_{DRAM} represent the capacity of the DRAM, and P_{DRAM} denote the proportion of DRAM allocated for storing high-frequency tokens. $S_K(\mathtt{M})$, $S_V(\mathtt{M})$, $S_Q(\mathtt{M})$ are the profiled sizes of the KVQ matrices of each token, we have

$$K_{DRAM} = S_{DRAM} \cdot P_{DRAM} / (S_K(\mathtt{M}) + S_V(\mathtt{M}) + S_Q(\mathtt{M})) \quad (1)$$

Again, K_{DRAM} is influenced by the allocated DRAM size and the word database (COCA) but not by prompt traces. It is possible to develop context-aware K_{DRAM}, and we leave it to future work.

3.3 Offline Profilers

We present the details of the physics of computing and storage (SSD and DRAM) in Appendix A. To schedule the disk access (DRAM is negligible) and the computing of the KVQ matrices of a token, we profile the energy and the delay of each computing component in the LLM inference, as well as the energy and the delay of the KVQ disk access. We first present the energy models for LLM inference. We then present the StoreLLM offline profilers.

The Energy Model for Computing. We present the energy model for GPU. We follow the AccelWattch power model [15]. Let $P_i(f)$ be the power of a GPU when executing LLM layer i,

$$P_i(f) = \beta_i C f^3 + \tau_i f + P_0 \quad (2)$$

where f is the core frequency of the GPU, and C is the gate capacitance decided by the GPU type. Parameters β_i and τ_i depend on the specific GPU when executing LLM layer i. P_0 is the constant power caused by peripheral components such as onboard fans. Modern GPUs usually employ Dynamic Voltage and Frequency Scaling (DVFS) [27], which adjusts the frequency according to the workload demands, which is our future work. According to [9], the execution time TC_i for an LLM layer i is

$$TC_i = \frac{FLOP(i, \mathbf{p})}{FLOPS_{peak} \cdot eff_i} \quad (3)$$

where $FLOP(i)$ is the number of the Floating point operations (FLOPs) of layer i, $\mathbf{p}$ is the input prompt, $FLOPS_{peak}$ is the GPU peak throughput count by Floating point operations per seconds (FLOPS), eff_i is the hardware efficiency representing the actual computing throughput divided by the peak throughput.

The energy consumption for computing an LLM layer i is

$$EC_i = P_i(f) \cdot TC_i \quad (4)$$

Given an LLM $\mathtt{M}$ with N layer and the specification of GPU, we can profile the $FLOP(\cdot)$, $\mathbf{G} = \{P_0, FLOPS_{peak}, \{\beta_i\}, \{\tau_i\}, \{eff_i\}, f, C\}$.

The Energy Model for Disk Access. Given a storage reading operation, the energy depends on the amount of reading data. To formulate the energy consumption of reading demanding data of an LLM layer i, we use the model in [23]:

$$ES_i = E_{MB} \cdot S_i \quad (5)$$

Here, S_i is the data amount of layer i, and E_{MB} is the energy of the reading operations per MByte data, counted in $Joule/MByte$. The storage model can be profiled as $\mathbf{S} = \{E_{MB}, \{S_i\}\}$.

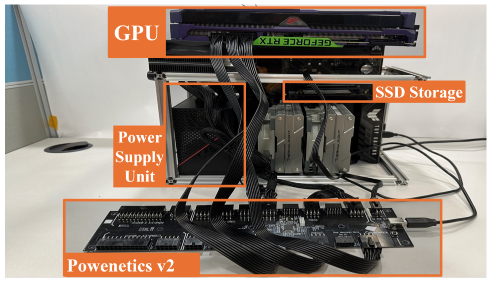

Figure 1: Energy profiling by Powenetics v2. It connects the GPU through an 8-pin plug and it connects the SSD through a PCIe expansion card.

StoreLLM offline profiler: We profile the execution of each LLM inference component. Given a physical computing environment, we need to profile $\mathbf{G}, \mathbf{S}, FLOPS(\cdot)$. Intrinsically, we employ a uniform sampling method with a delay cutoff to filter infeasible configurations during profiling. This allows us to quantify the hardware efficiency and calculate the delay and energy for LLM layers, enhancing profiling efficiency while ensuring accuracy. The profiling detail and results are shown in Appendix C.

3.4 Runtime Inference

The low-frequency tokens are only stored in SSD. Although the overall number of access to these tokens is small, they will increase the delay (TTFT).[3] We develop *a scheduler* to either disk access or re-compute the tokens in a prompt to reduce delays.

The problem for this scheduler is that, given a delay constraint of a prompt, for each token, schedules either (re-)compute the KVQ matrices of this token or disk access the KVQ matrices of this token with the objective of minimizing the energy per token. This problem exhibits a bounded Knapsack structure. Intuitively, each token has a "value" on energy saving (disk access instead of computing) and a "weight" of execution time. The value/energy saving of a token differs for different prompts. Due to space limitations, we omit further problem analysis. The details of our algorithm is in Appendix D. Intuitively, our scheduler takes the offline profiles and the time deduction of the pipeline parallel scheme as inputs and employs a dynamic programming search algorithm.

3.5 StoreLLM with FFN Compression

There are a number of components in the LLM computing pipeline. The two dominant components are KVQ matrices and FFN. There are mechanisms to compress the redundancy of FFN. We incorporate StoreLLM with FFN compression mechanisms.

StoreLLM-MoE: In LLM inference, the majority of the neurons will not be activated. Mixture-of-Experts (MoE) is developed. MoE is a model architecture of Transformer that uses multiple sub-models named *expert* to replace the FFNs, allowing only some of them to be selectively active during inference. The activated neurons are determined by the input token and controlled by an expert router. While existing MoE can expand the knowledge of LLMs for complex tasks, MoEfication reorganizes and divides FFN parameters into

[3] Prompt: "Why, thou owest God a death" (1 Henry IV, William Shakespeare). Here, thou and owest are rare, but they will delay the TTFT of this prompt.

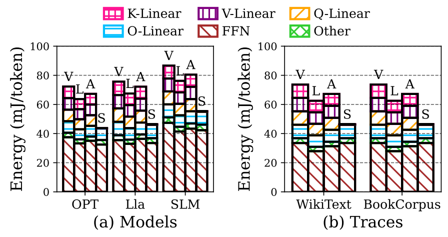

Figure 2: Energy comparison of StoreLLM and Baselines under different models and traces. (V: Vanilla; L: LazyLLM; A: AWQ; S: StoreLLM)

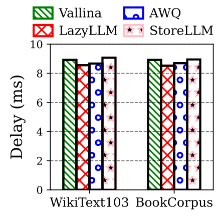

Figure 3: Delay comparison of StoreLLM and Baselines.

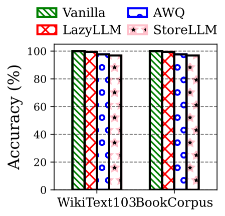

Figure 4: Accuracy comparison of StoreLLM and Baselines.

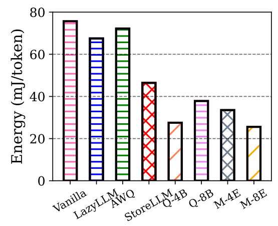

Figure 5: Energy comparison of StoreLLM-MoE and StoreLLM-PTQ.

small MoEs to compress FFNs for computational cost reduction. StoreLLM-MoE employs the MoEfication scheme in [35].

StoreLLM-PTQ: The LLM parameters have high precision and are redundant in many scenarios. Quantization reduces the LLM size by transforming the floating-point values used in model weights, activations, and gradients into integers or other discrete forms. The number of bits decides the memory storage, the energy consumption, and the delay of the inference. By carefully deciding the precision of data, the accuracy of the quantized model can be maintained. Post-Training Quantization (PTQ) is one quantization technique to quantize the model that has already been trained in high precision (e.g., FP32 or FP16). PTQ avoids retraining the model from scratch. StoreLLM-PTQ employs PTQ in [18].

4 Implementation and Evaluation

4.1 Implementation

We implement StoreLLM on a server with an Nvidia RTX3090 GPU under maximum frequency 2130MHz, 32GB DRAM, and a Samsung 980 Pro PCIe 4.0 2TB SSD. We implement three typical LLMs, OPT-6.7B [34], Llama 7B [30], and StableLM-7B [1]. Our default testing model is OPT-6.7B. Other details are shown in Appendix B.

4.2 Evaluation

In this section, we evaluate the overall performance of StoreLLM. Other comparisons under various input prompts, DRAM, and delay requirements are listed in Appendix E.

4.2.1 *Evaluation Methodology.*

Benchmarks. We evaluate using two datasets [20, 28]: Wikitext-103 [22], and BookCorpus [37]. The default one is Wikitext-103, with 100 prompt length. 10% of 32GB DRAM for KVQ matrices, i.e., $P_{DRAM} = 10\%$. We can store the top 3,495 tokens from COCA [6]. Wikitext-103 and BookCorpus contain 267,735 and 1,316,420 distinct tokens, occupying 257 GB and 1.263 TB storage, respectively. The top 3,495 tokens make up 1.30% and 0.26% of total tokens; however, accounting for 81.26% and 90.18% of all token access.

Baselines. (i) **LazzyLLM** develops a metric to skip computing unimportant tokens; (ii) **AWQ** develops PTQ to project high-bit weight tensors to low-bits to reduce the model size also the delay. (iii) **Vanilla** executes all layers exclusively on the GPU.

Metrics. (i) energy per token (mJ/token). (ii) delay (s): TTFT of output. (iii) accuracy (%): PPL score regarding to Vanilla.

4.2.2 *Evaluation Results.*

StoreLLM performance under LLMs and traces: Fig. 2 presents the energy comparison among StoreLLM and baselines across different LLMs and traces. The energy consumption for Vanilla is 72.31 mJ/token, 75.68 mJ/token, and 86.7 mJ/token for OPT-6.7B, Llama-7B, and StableLM-7B, respectively. In comparison, LazyLLM consumes 63.69 mJ/token, 67.51 mJ/token, and 76.19 mJ/token, and AWQ consumes 67.27 mJ/token, 72.21 mJ/token, and 80.49 mJ/token. StoreLLM is only 43.82 mJ/token, 46.44 mJ/token, and 55.36 mJ/token. Therefore, the energy of Vanilla, LazyLLM, and AWQ is 1.47×, 1.45×, and 1.40× to that of StoreLLM. We also present the energy breakdown of the LLM components, showing that the energy savings come from the KVQ-Linear. For example, in the OPT-6.7B model, the KVQ-Linear computing is 27.63mJ/token, and the storage access is 0.30mJ/token.

Fig. 3 shows the delay comparison among StoreLLM and baselines for various traces. We observe that StoreLLM exhibits a slight increase in delay. For instance, in BookCorpus, StoreLLM is 8.95 ms, with an increase of 5.05% and 2.87% compared to LazyLLM and AWQ, respectively. The variation in delay arises from the differing proportions of cached tokens in each dataset and the increased delay associated with accessing SSDs instead of computing by GPUs.

Fig. 4 shows the accuracy comparison among StoreLLM and baselines. We observe that StoreLLM has a similar accuracy performance, about 96.87%, compared to other baselines. It is because StoreLLM does not change the value of the KVQ matrices of tokens but only accesses them from the storage instead of re-computing them, as they have been computed before.

Performance of StoreLLM-MoE and StoreLLM-PTQ: Fig. 5 shows the energy comparison of StoreLLM-MoE and StoreLLM-PTQ with baselines for OPT-6.7B in Wikitext-103. Here, we set the number of MoE as 4 and 8, denoted as M-4E and M-8E, and the quantization level of PTQ is INT8 and INT4, denoted as Q-8B and Q-4B. M-4E and M-8E are 33.52 mJ/token and 25.56 mJ/token, which is 2.01×, 2.64× improvement compared to LazyLLM and 2.15×, 2.83× to AWQ, respectively. Q-8B and Q-4B are 37.84 mJ/token and 27.52 mJ/token, which is 1.78×, 2.45× improvement compared to LazyLLM and 1.91×, 2.62× to AWQ, respectively. The results demonstrate that with decreasing FFN, the energy saving could gain more with our design and exceed others.

5 Conclusion and Discussions

In this paper, we observed that in the LLM inference computing, the attention matrices of the tokens remain largely unchanged across different LLM inference. Intrinsically, the knowledge embedded in the tokens remains unchanged. We also noted that the computing operations are substantially energy consuming than the storage access operations due to their intrinsic differences in physics. We proposed StoreLLM, where the attention matrices are permanently stored, and the computing of attention matrices is substituted by the storage access of the pre-stored attention matrices. We believe that using storage access to substitute computing has applications beyond LLM inference. Human beings memorize and extract knowledge rather than computing it from scratch; this saves our energy.

Acknowledgments

Dan Wang's work is supported in part by RGC GRF 15200321, 15201322, 15230624, ITC ITF-ITS/056/22MX, ITS/052/23MX, and PolyU 1-CDKK, G-SAC8.

References

[1] Stability AI. 2023. StableLM: Stability AI Language Models. https://github.com/Stability-AI/StableLM.

[2] Anvita Bhagavathula et al. 2024. Understanding the Implications of Uncertainty in Embodied Carbon Models for Sustainable Computing. (2024).

[3] Andrew A Chien, Liuzixuan Lin, Hai Nguyen, Varsha Rao, Tristan Sharma, and Rajini Wijayawardana. 2023. Reducing the Carbon Impact of Generative AI Inference (today and in 2035). In *Proc. of workshop on sustainable computer systems*.

[4] Andrew A Chien, Liuzixuan Lin, Hai Nguyen, Varsha Rao, Tristan Sharma, and Rajini Wijayawardana. 2023. Reducing the Carbon Impact of Generative AI Inference (today and in 2035). In *Proc. of the 2nd Workshop on Sustainable Computer Systems (HotCarbon'23)*. Boston, MA, USA.

[5] Cybernetucs. 2024. Cybenetics Labs - PSU Efficiency Noise Level Certifications - Powenetics. https://www.cybenetics.com/index.php?option=powenetics. [Accessed 07-04-2024].

[6] Mark Davies. 2010. The Corpus of Contemporary American English as the first reliable monitor corpus of English. *Literary and linguistic computing* 25, 4 (2010), 447–464.

[7] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, et al. 2022. Glam: Efficient scaling of language models with mixture-of-experts. In *Proc. of the International Conference on Machine Learning (ICML'22)*. Baltimore, MD, USA.

[8] Mariam Elgamal, Doug Carmean, Elnaz Ansari, Okay Zed, Ramesh Peri, Srilatha Manne, Udit Gupta, Gu-Yeon Wei, David Brooks, Gage Hills, et al. 2023. Carbon-Efficient Design Optimization for Computing Systems. In *Proc. of the 2nd Workshop on Sustainable Computer Systems (HotCarbon'23)*. Boston, MA, USA.

[9] Ahmad Faiz, Sotaro Kaneda, Ruhan Wang, Rita Osi, Prateek Sharma, Fan Chen, and Lei Jiang. 2024. LLMCARBON: MODELING THE END-TO-END CARBON FOOTPRINT OF LARGE LANGUAGE MODELS. In *Proc. of International Conference on Learning Representations (ICLR'24)*.

[10] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research* 23, 120 (2022), 1–39.

[11] 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:2407.14057* (2024).

[12] Walid A Hanafy, Roozbeh Bostandoost, Noman Bashir, David Irwin, Mohammad Hajiesmaili, and Prashant Shenoy. 2023. The war of the efficiencies: Understanding the tension between carbon and energy optimization. In *Proc. of the 2nd Workshop on Sustainable Computer Systems (HotCarbon'23)*. Boston, MA, USA.

[13] Yuto Hayamizu, Masaru Kitsuregawa, and Kazuo Goda. 2024. Proactive Energy Management in Database Systems. (2024).

[14] Hongyu Hè, Michal Friedman, and Theodoros Rekatsinas. 2023. EnergAt: Fine-Grained Energy Attribution for Multi-Tenancy. In *Proc. of the 2nd Workshop on Sustainable Computer Systems (HotCarbon'23)*. Boston, MA, USA.

[15] Vijay Kandiah, Scott Peverelle, Mahmoud Khairy, Junrui Pan, Amogh Manjunath, Timothy G Rogers, Tor M Aamodt, and Nikos Hardavellas. 2021. AccelWattch: A power modeling framework for modern GPUs. In *Proc. of the IEEE/ACM International Symposium on Microarchitecture (MICRO'21)*. Virtual Event.

[16] Sven Köhler, Benedict Herzog, Henriette Hofmeier, Manuel Vögele, Lukas Wenzel, Andreas Polze, et al. 2023. Carbon-Aware Memory Placement. In *Proc. of the 2nd Workshop on Sustainable Computer Systems (HotCarbon'23)*. Boston, MA, USA.

[17] Wonbeom Lee, Jungi Lee, et al. 2024. {InfiniGen}: Efficient generative inference of large language models with dynamic {KV} cache management. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI'24)*. 155–172.

[18] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. *Proc. of Machine Learning and Systems (MLSys'24)* (2024).

[19] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, et al. 2024. CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving. In *Proc. of ACM SIGCOMM'24*.

[20] Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS'23)*. New Orleans, LA, USA.

[21] Priyanka Mary Mammen, Noman Bashir, Ramachandra Rao Kolluri, Eun Kung Lee, and Prashant Shenoy. 2023. Cuff: A configurable uncertainty-driven forecasting framework for green ai clusters. In *Proceedings of the ACM International Conference on Future Energy Systems (e-Energy'23)*.

[22] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv:1609.07843* (2016).

[23] Vidyabhushan Mohan, Trevor Bunker, Laura Grupp, Sudhanva Gurumurthi, Mircea R Stan, and Steven Swanson. 2013. Modeling power consumption of nand flash memories using flashpower. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 7 (2013), 1031–1044.

[24] Mark EJ Newman. 2005. Power laws, Pareto distributions and Zipf's law. *Contemporary physics* 46, 5 (2005), 323–351.

[25] Sophia Nguyen, Beihao Zhou, Yi Ding, and Sihang Liu. 2024. Towards Sustainable Large Language Model Serving. In *Proc. of the Workshop on Sustainable Computer Systems (HotCarbon'24)*.

[26] Varsha Rao and Andrew A Chien. 2024. Understanding the Operational Carbon Footprint of Storage Reliability and Management. In *Proc. of the Workshop on Sustainable Computer Systems (HotCarbon'24)*.

[27] Hamid Sarbazi-Azad. 2016. *Advances in GPU research and practice.* Morgan Kaufmann. 471–505 pages.

[28] Weijia Shi, Julian Michael, Suchin Gururangan, and Luke Zettlemoyer. 2022. kNN-Prompt: Nearest Neighbor Zero-Shot Inference. *arXiv:2205.13792* (2022).

[29] Jovan Stojkovic, Esha Choukse, Chaojie Zhang, Inigo Goiri, and Josep Torrellas. 2024. Towards Greener LLMs: Bringing Energy-Efficiency to the Forefront of LLM Inference. *arXiv:2403.20306* (2024).

[30] Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. 2023. Llama: Open and efficient foundation language models. *arXiv:2302.13971* (2023).

[31] Grant Wilkins et al. 2024. Hybrid Heterogeneous Clusters Can Lower the Energy Consumption of LLM Inference Workloads. In *Proc. of the ACM International Conference on Future and Sustainable Energy Systems (e-Energy'24)*.

[32] Grant Wilkins et al. 2024. Offline Energy-Optimal LLM Serving: Workload-Based Energy Models for LLM Inference on Heterogeneous Systems. In *Proc. of the Workshop on Sustainable Computer Systems (HotCarbon'24)*.

[33] Thomas Wolf, Lysandre Debut, Victor Sanh, et al. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proc. of the Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP'20)*. Virtual Event.

[34] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv:2205.01068* (2022).

[35] Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2022. MoEfication: Transformer Feed-forward Layers are Mixtures of Experts. In *Findings of the Association for Computational Linguistics(ACL'22)*.

[36] Zhenyu Zhang, Shiwei Liu, Runjin Chen, et al. 2024. Q-Hitter: A Better Token Oracle for Efficient LLM Inference via Sparse-Quantized KV Cache. *Pro. of Machine Learning and Systems (MLSys'24)* (2024).

[37] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proc. of the IEEE international conference on computer vision (ICCV'15)*. Santiago, Chile.

A Background on the Energy of Computing Processors and Storage

Processors and storage have fundamental differences in terms of physics, leading to differences in their energy consumption. Processors consist of the Control Unit and the Arithmetic Logic Unit (ALU), with many logic circuits. Logic circuits are composed of logic gates with transistors. A current flows through when a logic gate processes a bit of data. A simple calculation operation will activate a large number of logic gates.

There are two types of storage, disk (e.g., SSD) and memory (e.g., DRAM). The primary storage component of an SSD is the NAND Flash Memory Chip, and its basic storage unit is the Floating Gate Transistor (FGT). An FGT has a control gate and a floating gate.

Take the reading operation as an example, the floating gate stores the bits by the electrons adhered to the floating gate. A reading operation will power up the control gates with various voltages; and if a voltage matches the corresponding threshold voltage, the current will pass through, and a reading operation completes. The basic storage unit of a DRAM consists of a transistor and a capacitor. The data is represented by whether the capacitor is charged or not. A reading operation will power up the transistor first, and if the capacitor has been charged (representing '1'), a current will pass through the transistor. Unlike the FGT, ordinary transistors have only one threshold voltage and it is normally very low. The refresh operation for maintaining the data integrity is also usually negligible. Hence, the operational power of the memory is much lower than that of the SSD. In general, the energy of a processor is 20× to 50× greater than that of an SSD, and the energy of an SSD is 50× greater than that of a DRAM [14].

B Details of Implementation.

Offline Profilers. We implement the LLM energy profiler using Powenetics v2, a Power Measurements Device (PMD) developed by Cybernetics [5], see Fig. 1. It can measure and monitor the power of running computer hardware through its 13 sensors to read voltage and amperage up to 1000 times per second with the resolution $1mV$ and $5mA$, respectively. The results are shown in Appendix C, Table 5 and Table 6.

Runtime Inference. We implement StoreLLM by the Transformers [33]. During inference, we can either compute the KVQ matrices or access them from storage. This is achieved by intercepting *K,V,Q=Linear.forward()* and using *K,V,Q = torch.load(address)*. We maintain a hashtable to track the addresses of each cached KVQ token. If the KVQ is in DRAM, the *address* points to the DRAM location; if it's on the SSD, it indicates the file path. We implement MoE based on SwitchTransformer [10]. We extend the Auto-GPTQ to enable the quantization of the FFN. We modify the *BaseQuantizeConfig* class to provide model quantization functions.

C Details of Offline Profilers

Methodology: We employ a uniform sampling method with a defined delay cutoff to filter out infeasible configurations. Given the monotonic relationship between delay and frequency, we discard invalid configurations and redistribute saved samples, ensuring uniform exploration of valid configurations within the search space. Given the LLM profiles $\mathtt{M} = \{V, h, h_{FFN}, N_e\}$ in Table 1 with prompt length n, where V is the Vocabulary size, h is the hidden size, h_{FFN} is the hidden size of FFN and N_e is the number of experts, we can compute the computation complexity model $FLOP(\cdot)$ as follows:

$$FLOP(i,n) = \begin{cases} 2nhV, & i = \text{Embedding}, \\ 2nh^2 + n^2h, & i = \text{K/V/Q/O-Linear}, \\ 16nh_{FFN}^2, & i = \text{FFN}, \\ 16nh_{FFN}^2/N_e, & i = \text{MoE}, \\ 2nhV, & i = \text{Head}. \end{cases} \quad (6)$$

Algorithm 1: Read-compute Scheduling Algorithm

```
Input: $\{EC_i\}, \{ES_i\}, \{TC_i\}, \{TS_i\}, t_{max}$, $\mathbf{p} = \{t_0, t_0, \ldots, t_n\}$, Algorithm Step $\epsilon$.
Output: Read/Compute Controller $\mathbf{W}$
1  $T = t_{max} - (\sum_i^N TC_i - \sum_i^{K/V/Q} TC_i)$;
2  Initialize a 2D array $dp[n+1][\lfloor T/\epsilon \rfloor + 1]$ with $\infty$;
3  for $j \leftarrow 1$ to $n$ do
4      $e[j][0], e[j][1] = \sum_i^{K/V/Q} ES_i/n, \sum_i^{K/V/Q} EC_i/n$ ;
5      $t[j][0], t[j][1] = \sum_i^{K/V/Q} TS_i/n, \sum_i^{K/V/Q} TC_i/n$ ;
6  for $i \leftarrow 1$ to $n$ do
7      for $j \leftarrow 0$ to $\lfloor T/\epsilon \rfloor$ do
8          for $k \leftarrow 1$ to $2$ do
9              if $j \cdot \epsilon \geq t[i-1][k-1]$ then
10                 $dp[i][j] \leftarrow \min(dp[i][j], dp[i-1][j - \lfloor t[i-1][k-1]/\epsilon \rfloor] + e[i-1][k-1])$;
11 $\mathbf{W} \leftarrow argmin_n(dp[n][j]$ for $j = 0$ to $\lfloor T/\epsilon \rfloor)$;
12 Return $\mathbf{W}$;
```

Using the generated samples and the GPU profiles G (shown in Table 2), we can quantify the energy parameters ($\{\beta_i\}$, $\{\tau_i\}$) listed in Table 3 and the hardware efficiency $\{eff_i\}$ in Table 4.

For the Quantization of LLM, assuming that the hardware efficiency of existing LLM M is in FP32 is eff_i, after the quantization of LLM, the quantized hardware efficiency can be computed as:

$$eff_i^F = eff_i \times Q_F, \quad (7)$$

Table 1: Profiles of LLMs $\mathtt{M} = \{V, h, h_{FFN}, N_e\}$.

LLM	V	h	h_{FFN}	N_e
OPT	50272	4096	16384	6
Llama	32000	4096	11008	152
StableLM	50432	6144	24576	87

where $F \in$ {FP16, FP8, INT16, INT8, INT4} is the target quantized data format. Note that after quantization, the Q_F is also required to re-profile. This is because the hardware efficiency is influenced by factors such as the implementation of CUDA or the use of accelerators, which can significantly impact performance. However, the quantization of experts plays a crucial role in reducing the model size and the computation required for each expert. By lowering the precision of certain model components, quantization can lead to substantial savings in both memory and computational costs.

We then calculate the computing delay TC_i and energy EC_i of an LLM layer i using the formulation in Eq. 3 and Eq. 4 as shown in Table 5, as well as the storage reading delay TS_i and energy ES_i in Table 6 allowing us to determine energy consumption. This approach enhances profiling efficiency while ensuring accurate and reliable delay and power consumption models, ultimately improving performance and energy management for LLM inference tasks.

Table 2: Profiles of typical GPU G = $(FLOPS\_peak, P_0, f)$.

GPU	$FLOPS\_peak$ (FLOPs)	P_0 (W)	f (MHz)
NVIDIA RTX 3090	35.58T	109	1995
NVIDIA RTX 4090	83T	61.8	2880
NVIDIA RTX 4070Ti	40.09T	33	2835

Table 3: Energy consumption parameters (βC, τ) for LLM components on different GPUs.

GPU	LLM	***EMB*** $(\beta C, \tau)$	***LIN*** $(\beta C, \tau)$	***FFN*** $(\beta C, \tau)$
RTX 3090	OPT	$(1.71E-8, -7.08E-2)$	$(1.47E-8, -5.85E-2)$	$(1.49E-8, -6.05E-2)$
	Llama	$(1.61E-8, -5.43E-2)$	$(1.66E-8, -5.97E-2)$	$(1.67E-8, -6.01E-2)$
	StableLM	$(9.57E-9, -2.07E-2)$	$(9.6E-9, -2.09E-2)$	$(9.79E-9, -2.2E-2)$
RTX 4090	OPT	$(1.81E-9, -5.82E-3)$	$(1.82E-9, -5.9E-3)$	$(1.84E-9, -6.06E-3)$
	Llama	$(1.82E-9, -6.12E-3)$	$(1.82E-9, -6.04E-3)$	$(1.81E-9, -6E-3)$
	StableLM	$(2.03E-9, -7.22E-3)$	$(2.03E-9, -7.22E-3)$	$(2.03E-9, -7.22E-3)$
RTX 4070Ti	OPT	$(1.75E-9, -6.95E-3)$	$(1.73E-9, -6.54E-3)$	$(1.73E-9, -6.62E-3)$
	Llama	$(1.31E-9, -2.38E-3)$	$(1.34E-9, -2.74E-3)$	$(1.33E-9, -2.73E-3)$
	StableLM	$(1.73E-9, -5.84E-3)$	$(1.75E-9, -6.27E-3)$	$(1.75E-9, -6.26E-3)$

EMB: Embedding Layer. ***LIN***: K/V/Q/O Linear Layers. ***FFN***: Feed Forward Layers.

Table 4: Hardware efficiency (eff_i) for LLM components on different GPUs.

GPU	LLM	***EMB*** (*eff*)	***LIN*** (*eff*)	***FFN*** (*eff*)
NVIDIA RTX 3090	OPT	2%	1%	9%
	Llama	59%	36%	4%
	StableLM	71%	41%	20%
NVIDIA RTX 4090	OPT	9%	2%	29%
	Llama	37%	65%	10%
	StableLM	59%	76%	65%
NVIDIA RTX 4070Ti	OPT	2%	1%	8%
	Llama	52%	32%	3%
	StableLM	74%	40%	19%

EMB: Embedding Layers. ***LIN***: K/V/Q/O Linear Layers. ***FFN***: Feed Forward Layers.

Table 5: Computing Energy Profiling Results (EC_i, TC_i) per token of LLM components on different GPUs.

GPU	LLM	***EMB*** (EC_i, TC_i)	***LIN*** (EC_i, TC_i)	***FFN*** (EC_i, TC_i)
RTX 3090	OPT	$(388.61mJ, 4.18ms)$	$(438.84mJ, 5.94ms)$	$(34.15mJ, 0.40ms)$
	Llama	$(358.41mJ, 3.17ms)$	$(76.80mJ, 0.79ms)$	$(52.31mJ, 0.54ms)$
	StableLM	$(678.32mJ, 7.33ms)$	$(119.23mJ, 1.11ms)$	$(46.74mJ, 0.42ms)$
RTX 4090	OPT	$(887.69mJ, 13.75ms)$	$(39.88mJ, 0.59ms)$	$(12.03mJ, 0.18ms)$
	Llama	$(88.11mJ, 1.24ms)$	$(26.08mJ, 0.39ms)$	$(17.03mJ, 0.25ms)$
	StableLM	$(95.41mJ, 1.38ms)$	$(38.56mJ, 0.54ms)$	$(12.63mJ, 0.17ms)$
RTX 4070Ti	OPT	$(111.23mJ, 3.45ms)$	$(103.04mJ, 2.90ms)$	$(12.30mJ, 0.33ms)$
	Llama	$(59.34mJ, 1.57ms)$	$(23.72mJ, 0.60ms)$	$(16.71mJ, 0.42ms)$
	StableLM	$(76.67mJ, 2.05ms)$	$(29.75mJ, 0.72ms)$	$(14.47mJ, 0.34ms)$

EMB: Embedding Layers. ***LIN***: K/V/Q/O Linear Layers. ***FFN***: Feed Forward Layers.

D The Computing-Storage Scheduler

Our scheduler utilizes offline profiles and time deductions from the pipeline parallel scheme, employing a dynamic programming search algorithm, as introduced in Algorithm 1. This problem resembles a bounded Knapsack structure, where each token has a "value" representing energy savings (from storage access instead of computation) and a "weight" reflecting execution time. The value/energy saving of a token differs for different prompts. Specifically, the input of this algorithm is the profiling results from Section C, including $\{EC_i\}$, $\{ES_i\}$, $\{TC_i\}$, $\{TS_i\}$. The LLM inference system also needs to set up a maximum delay for the inference as t_{max}, for the prompt $\mathbf{p} = \{t_0, t_0, \ldots, t_n\}$. We also need to setup an algorithm step ϵ to control the algorithm iteration. The output of this algorithm is the Read/compute controlling parameter matrix $\mathbf{W} = \{w_i\}$ to notice if the KVQ matrices read from storage. $w_i = 0$ if computing from storage, otherwise, $w_i = 1$.

Table 6: Reading Energy Profiling Results (ES, TS) per token of KVQ-Linears on different GPUs.

Storage	LLM	***LIN*** (ES, TS)
DRAM	OPT	$(0.092mJ, 0.083ms)$
	Llama	$(0.086mJ, 0.079ms)$
	StableLM	$(0.129mJ, 0.116ms)$
SSD	OPT	$(2.64mJ, 0.789ms)$
	Llama	$(2.61mJ, 0.779ms)$
	StableLM	$(3.89mJ, 1.17ms)$

Line 1 first computes the actual maximum necessary delay T, excluding the necessary computing time of each token, such as the computing time of embedding, FFN, etc. Line 2 initializes a 2D dynamic programming array dp, which will store the minimum energy required for each combination of items and time values. All values are initially set to infinity, indicating that no solution has been found yet. Lines 3 to 5 initialize the array to store the possible energy consumed from storage or re-computing. From lines 6 to 10, if the time condition is satisfied, the algorithm updates the DP table at $dp[i][j]$. It compares the current energy value and the new potential value formed by choosing the current process for item i. This takes the minimum of The existing value $dp[i][j]$, or the energy from the previous item at $i-1$, with the remaining time after considering the current execution times. In line 11, after populating the DP table, the algorithm looks for the minimum energy value and saves the controlling parameter from the last item across all possible time slots. Figure 13 illustrates the effectiveness of our scheduling strategy in reducing output delay. Under two different data sources, our algorithm achieves a 17.77% improvement in delay reduction compared to methods without the hierarchical scheduling strategy.

E Additional Evaluation

Impact of the prompt with distinct length. We now present an experiment of the delay per token of StoreLLM under the different lengths of input prompts, as shown in Fig. 6. We select prompts with 128, 256, 512, and 1024 lengths from the dataset WikiText103. The results show that the overall delay of prompts and the length are positively correlated.

Impact of the available size of DRAM. We present an experiment of the delay per token of StoreLLM under the available size of DRAM ($S_{DRAM} \cdot P_{DRAM}$), as shown in Fig. 7. We select 16 GB, 32 GB, and 128 GB that can store top-28k, top-56k, and top-224k of frequently used tokens in COCA. We also set StoreLLM with no hierarchical cache model as 0 GB. The results show that the delay of 0 GB DRAM size is 1.84× greater than that of StoreLLM. When improving to 16 GB, the delay reduces to 10.12ms. This is because the reading speed of DRAM is 10 times faster than that of SSD, and with our hierarchical cache model, the most frequently used tokens are accessed from DRAM, and this reduces the delay significantly.

Impact of the maximum delay. We then present an experiment of the energy consumption under distinct maximum delay t_{max} of

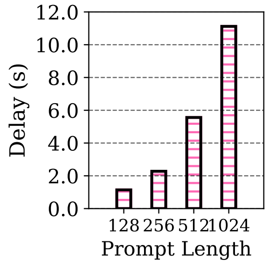

Figure 6: The impact of prompts with distinct length.

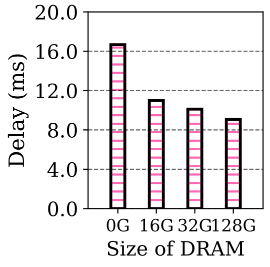

Figure 7: The impact of the computing and disk access coordination.

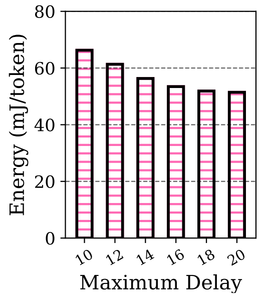

Figure 8: Impact of the maximum delay.

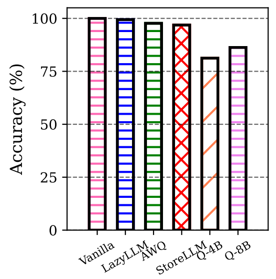

Figure 9: Accuracy performance of FFN compression.

inference. We set the t_{max} from 10s to 20s, and the input prompt is from the Wikitext dataset with a 1024 average length. The results are shown in Fig. 8. We observe that when the t_{max} is lower than 10, the energy that is consumed is 1.29 times greater than when $t_{max} = 20$. It is because, in this scenario, all the tokens are required to be computed since reading from the storage is time costly. When t_{max} is over 20, the energy per token achieves the minimal values at about 51mJ/token. It is because the maximum delay is larger than reading all tokens from storage, which can save 28.76% of energy overall.

Accuracy performance of FFN compression. We present an experiment comparing the accuracy of StoreLLM-PTQ with baseline models as shown in Fig. 9. The quantization levels for PTQ are INT8 and INT4, referred to as Q-8B and Q-4B. Q-8B and Q-4B achieve 81.23% and 86.27% accuracy. Accuracy decreases as the quantization level shifts from Q-8B to Q-4B, while energy consumption reduces by 27.2%. The results indicate that FFN compression has a limited impact on accuracy, making it applicable in practice. Note that here, do not evaluate the accuracy performance of StoreLLM-MoE since it requires the retraining of the applied LLMs. We leave it as our future works.

Number of Experts. We evaluate the impact of the number of experts on energy savings. Fig. 5 shows that increasing the number of experts leads to greater energy savings. When the number of expert is 8, energy per token is 31.14% lower than 4. This can be attributed to the higher sparsity of the MoE model, as a larger number of non-activated experts reduces the overall computational overhead.

Prompts Complexity. We also studied how the complexity of prompts affects the energy. We created five different prompts based on the usage frequency of the words comprising the prompts. In each prompt, the word usage frequency fell into the following ranges: 0%-20%, 20%-40%, 40%-60%, 60%-80%, and 80%-100%. Results in Figure 10 showed that the energy consumption for each group of prompts was consistent. The results indicate that prompt complexity has no significant impact on energy consumption. This result is also intuitive, as more complex prompts do not necessarily contain more complex words. By caching the tokens within these prompts, we can achieve the same energy savings regardless of their complexity.

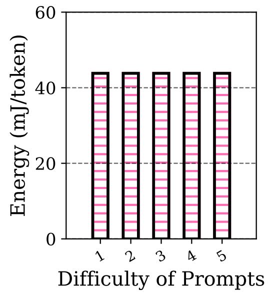

Figure 10: Impact of the different difficulty.

Quantization Level. Figure 14 shows the energy impact on different quantization levels. Compared with naive StoreLLM, model quantized to 4 bit can save more than 68.75% energy. We observe that quantization to no more than 8 bits has minimal impact on performance while offering significant energy savings.

Model Hidden Size. Different models have different hidden sizes, which result in varying KVQ sizes, as shown in Table 7. For example, the Llama 2 7B and 13B models have hidden sizes of 4096 and 5120, respectively, leading to a 25% difference in their KVQ sizes. Results on Wikitext-103 show that excluding only the tokens with frequencies in the last 20% from the cache system led to less than a 5% difference in energy consumption between the models. Therefore, regardless of the KVQ size, StoreLLM demonstrates consistently good performance.

Model Name	Hidden Size	KVQ Size per Token
Llama2 7B	4096	786KB
Llama2 13B	5120	983KB
OPT 6.7B	4096	786KB
OPT 13B	5120	1228KB
StableLM 7B	6144	1179KB
StableLM 12B	6144	1327KB

Table 7: Hidden Size of Different Models.

Cached Ratio. The proportions also affect the energy saving. Essentially, caching more tokens increases the potential for energy savings but also results in higher latency. Figure 11 and Figure 12 show the impact of different cache ratios on both energy and delay. As the cache ratio gradually decreases, energy consumption increases significantly while delay decreases slowly. This is because StoreLLM reduces energy consumption substantially at the cost of only a small increase in delay.

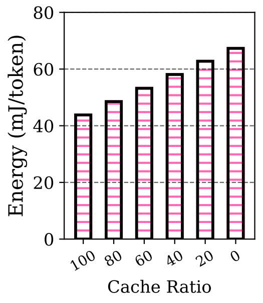

Figure 11: Impact of different cache ratios on energy.

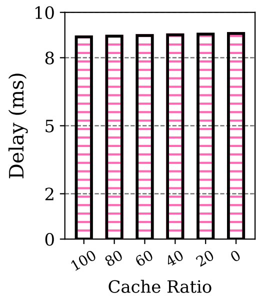

Figure 12: Impact of different cache ratios on delay.

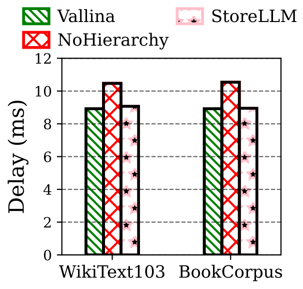

Figure 13: Ablation study of the scheduling algorithm.

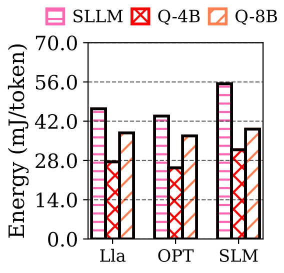

Figure 14: Impact of the different quantization level.

F Discussion and Future Works

Tokens Distribution Shift. Token distributions shift in two situations: when language habits evolve over time and in specific language domains or scenarios. The first is a slow process, taking years, and can be ignored during runtime inference. The second is more common, like in multi-language conversations or professional discussions. For example, the most frequently used tokens in French-English bilingual conversations or professional medical discussions have distinct token distributions compared to those we ranked in the general English corpus. As a result, for optimal system performance the cached KVQ matrices of frequently used tokens are required to dynamically adapt the contents inputted during inferences, which is the subject of our future work.

Dynamic DRAM Allocation. We plan to incorporate dynamic DRAM allocation into our system to improve memory efficiency and system performance. Specifically, we aim to develop an algorithm that monitors the memory requirements of tasks in real-time and adjusts the DRAM allocation accordingly, thereby minimizing memory contention and optimizing resource utilization. This will involve designing a resource allocation strategy based on workload characteristics, allowing the system to dynamically allocate memory based on task priority and demand. Additionally, we will explore memory access pattern analysis to optimize memory access paths, reducing latency and energy consumption. With these improvements, we expect to improve system stability and responsiveness in complex operational environments significantly.

StoreLLM for Latency-sensitive Tasks. In StoreLLM, we control the balance between latency and energy consumption by the Compute-storage Scheduler. It allows StoreLLM to sacrifice more energy to fulfill the latency constraints. When meeting latency-sensitive tasks, it could be unsolvable under a specific hardware. There are several trade-offs to study in depth in the future, e.g., adopting faster storage hardware to reduce latency. We plan to test our StoreLLM system on the latest high-performance disks to accelerate disk access. Specifically, we aim to leverage advanced storage technologies, such as NVMe 2.1 and PCIe 5.0, which offer significantly higher speed and lower latency, as shown in Tab. 8. For example, if using Corsair MP700 SSD, the latency is expected to be reduced 42.86%, while the energy consumption will reduce over 30%. Additionally, we will analyze the interaction between disk performance and workload characteristics to optimize StoreLLM for different scenarios, further enhancing its scalability and responsiveness.

Disk	**Read (MB/s)**	**Capacity (TB)**	PCIe Gen
Samsung 980 Pro	7,000	0.25/0.5/1/2	4.0
WD Black SN850X	7,300	1/2/4/8	4.0
Seagate FireCuda 530	7,300	0.5/1/2/4	4.0
Crucial T700	12,400	1/2/4	5.0
Corsair MP700	10,000	1/2	5.0

Table 8: SSD to be tested.