

Endor: Exploit Nearly-Decode-Only Opportunities of LLM Reasoning on Near-Memory Architecture

Jun Liu^{1,2}, Tianlang Zhao¹, Shiyi Liu⁵, Jiancai Ye¹, Lin Li⁴, Zhen Yu¹, Li Ding¹, Hao Zhou¹,

Zhenhua Zhu⁴, Xuefei Ning⁴, Yuan Xie⁵, Yu Wang⁴, Guohao Dai^{1,2,3†}

¹Shanghai Jiao Tong University, ²Infinigence-AI, ³SII, ⁴Tsinghua University,

⁵Hong Kong University of Science and Technology

[†]Corresponding author: daiguohao@sjtu.edu.cn

Abstract—Reasoning with Large Language Models (LLMs) has become a pivotal research topic because their logical abilities significantly surpass those of standard LLMs. LLM reasoning typically forms multiple chains of thought, action-by-action, and selects the best one as the final answer. However, the inference overhead of LLM reasoning is more than an order of magnitude higher than that of LLM. Despite the emerging shift towards memory-optimized algorithms and near-memory hardware, we still face the following challenges: (1) Existing memory-centric algorithms (e.g., KV cache technique) have low computational utilization (< 4% on NVIDIA A100 GPU) due to intensive memory access for inter-action data. (2) Emerging hardware architectures (e.g., near-memory processing) fail to fully utilize the inherent parallelism due to dependencies among models, leading to low utilization of memory bandwidth.

To tackle these challenges, we propose *Endor*, a hardware-algorithm co-design to accelerate the inference of LLM reasoning efficiently. We identify that the auto-regressive decoding of LLM reasoning changes from the token level to the action level in terms of the computing paradigm. At the algorithm level, we propose a “nearly-decode-only” method which encompasses an efficient inter-action cache reuse method and a prediction-based pipeline optimization to reduce computation overhead. At the hardware level, we propose Endor-NMP, a near-memory accelerator featuring a score-aware cache management architecture and a heterogeneous mapping dataflow. Endor fully exploits both interaction and intra-action parallelism to improve memory bandwidth utilization. Experimental results demonstrate that neither existing algorithms nor hardware can achieve the expected acceleration. Endor achieves an end-to-end average speedup of 2.97× and 2.52× compared to the NVIDIA A100 GPU and advanced LLM accelerators on multiple models and datasets.

Index Terms—Large Language Model, Reasoning, Near-Memory Processing, Hardware Accelerator

I. INTRODUCTION

Improving the reasoning ability of Large Language Models (LLMs) has become an important research topic for general artificial intelligence [1], [2]. Many “X”-of-thought methods (e.g., Chain-of-thought, Tree-of-thought, and so on) [3]–[5] show that expressing step-by-step reasoning in natural language can improve the performance of LLMs on various logical inference tasks (e.g., mathematics, coding, and commonsense reasoning). Recent studies [6], [7] explicitly point out that while scaling training alone is challenging for enhancing LLM reasoning, scaling inference capabilities can significantly improve reasoning performance, further raising the interest in accelerating inference for LLM reasoning.

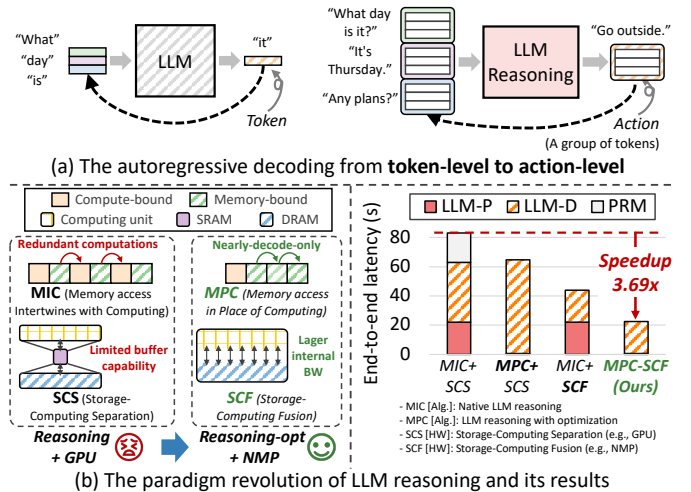


Fig. 1. The key insight and results of Endor. (a) The autoregressive decoding of LLM reasoning shifts from the token level to the action level in terms of the computing paradigm. (b) We need MPC algorithm and SCF hardware design paradigm, and (c) we achieve over 3.69× end-to-end speedup over NVIDIA A100 GPU (LLM-Reasoner [1] on GSM8K [8] dataset).

Generally speaking, LLM reasoning methods typically follow a “search-based output generation” pipeline [9]. In this approach, a graph is maintained as the current search state, where each node (i.e., action) represents a reasoning step. During each iteration, the pipeline selects the next node to sample new reasoning steps, guided by the evaluation results of a Process Reward Model (PRM). We point out that *the autoregressive decoding of LLM reasoning changes from the token level to the action level* (Fig. 1(a)). The inference overhead of this pipeline is more than an order of magnitude higher than that of a single plain LLM inference [10]. Existing optimization methods (e.g., sparsification [11] and quantization [12]) for LLM cannot bridge the huge gap among them.

Our *key observation* is that **the paradigm of MIC+SCS leads to the inefficiency of LLM reasoning on GPUs**. MIC (Memory access Intertwines with Computing) indicates that memory access-dominated (e.g., decode stage) and computation-dominated (e.g., prefill stage) operations execute alternately in existing LLM reasoning algorithms. SCS (Storage-Computing Separation) refers to existing von Neumann hardware (e.g., GPUs), whose storage and computing units are separated. Specifically, the MIC algorithm suffers

from massive repetitive input prompts across LLM inference steps, leading to 89.1% average redundant computations. Besides, the SCS hardware with the limited on-chip buffer capability results in memory access becoming the bottleneck.

Our *key insight* is that there is an urgent need for **the paradigm revolution from MIC to MPC (Memory access in Place of Computing), and from SCS to SCF (Storage-Computing Fusion)**, to fully unleash the potential of LLM reasoning (Fig. 1(b)). However, we still face serious challenges when moving towards this new paradigm: **Challenge-1: The MPC algorithms (e.g., the KV cache technique [13]) face low computational utilization (<4% on NVIDIA A100 GPU) due to large memory access of the inter-action data in LLM reasoning.** Specifically, the caching requirements for the KV data of LLM reasoning (about 10GB) are one to two orders of magnitude higher than those of LLM. GPU’s limited on-chip buffer capability (about 40MB L2 cache on NVIDIA A100 GPU) results in unresolved inference overhead. **Challenge-2: The SCF hardware (e.g., near-memory architectures [14]–[16]) fails to fully utilize the inherent parallelism due to the dependencies among models, leading to low utilization of memory bandwidth.** Specifically, while SCF architectures boast large on-chip memory capacity and high internal bandwidth, performance is bottlenecked by algorithmic inefficiencies. The dependencies among models necessitate frequent operator switching and redundant computations, which prevent the hardware from saturating its bandwidth.

To address these challenges, we propose Endor, a MPC algorithm and SCF hardware co-design to accelerate LLM reasoning inference. Specifically, we design the heterogeneous architecture based on near-memory processing as a case study. The main contributions of this paper are as follows:

- At the *algorithm* level, we propose a novel MPC algorithm, namely **the nearly-decode-only for LLM reasoning**, which encompasses an inter-action cache reuse method and a prediction-based pipeline optimization method. The inter-action cache method nearly eliminates all the prefill computations of LLM in reasoning (except for the root action). The prediction-based pipeline optimization breaks the dependencies between LLM and PRM, enabling parallelism between them.
- At the *hardware* level, we propose **an NMP-based hardware architecture** that integrates score-aware cache management architecture and heterogeneous dataflow architecture. The score-aware cache management architecture dynamically adjusts priority order of cache blocks by calculating scores of cache blocks of different actions, maximizing overall reuse times. We design a heterogeneous dataflow to fully exploit the parallelism of both inter-action and intra-action to improve memory bandwidth utilization.
- The *experimental results* demonstrate that **neither MPC algorithm nor SCF hardware can achieve the expected acceleration for LLM reasoning** (Fig. 1(c)). Endor achieves end-to-end acceleration of average $2.97\times$ and $2.52\times$ compared to the NVIDIA A100 GPU and advanced LLM accelerators on multiple models and datasets.

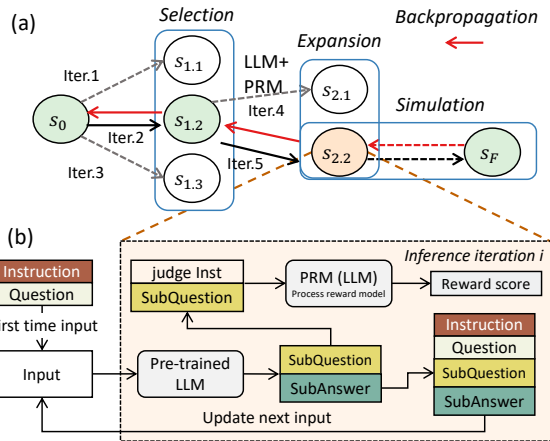


Fig. 2. (a) The four steps of LLM reasoning (MCTS-based). (b) The detailed process of the action level.

II. BACKGROUND AND MOTIVATION

A. Inference process of LLM

During the inference phase, LLMs adopt an auto-regressive manner to generate the tokens sequentially [17]. In each iteration, the LLM takes the initial input tokens of users and previously generated tokens as input sequence and predicts the next token. Key-Value (KV) cache [18] involves storing and reusing previous key and value pairs within the multi-head self-attention block. Based on the above methods and techniques, the inference process of LLMs can be divided into the prefill stage and the decode stage. In the prefill stage, the LLM calculates and stores the KV cache of the initial input tokens and generates the first output token. In the decode stage, the LLM generates the output tokens one by one with the KV cache, where the computing flow of self-attention for token prediction can be expressed by $QKV = XW_{QKV}$, $S = QK^T$, $P = \text{Softmax}\left(\frac{S}{\sqrt{d}}\right)$, and $O = PV$, and then updates KV cache with the key and value pairs of the newly generated token. Then the input features undergo the FFN layer for nonlinear transformation (e.g., Act0 is processed via two linear transformations and nonlinear activation to get Act2).

B. Reasoning with LLMs

Some methods enhance the reasoning ability of LLMs, and the MCTS (Monte Carlo Tree Search) strategy performs best among these [1]. The MCTS strategy, as shown in Fig. 2(a), consists of four steps: (1) **Selection**. Choose a child state node based on a specific strategy, such as the Upper Confidence Bound (UCB) score. (2) **Expansion**. If the selected node is not in a terminal state, use LLM to generate the next step, and PRM to generate the score for the next step. (3) **Simulation**. Repeat the previous two steps until reaching a terminal state. (4) **Backpropagation**. Tracing back along the tree from the termination node, updating each node’s scores and other states.

In the MCTS strategy, we focus on the expansion phase, as shown in Fig. 2(b) and Fig. 3(a). First, we concatenate the instruction, question, and historical output records to form the current input. Then, we use the LLM to generate the following

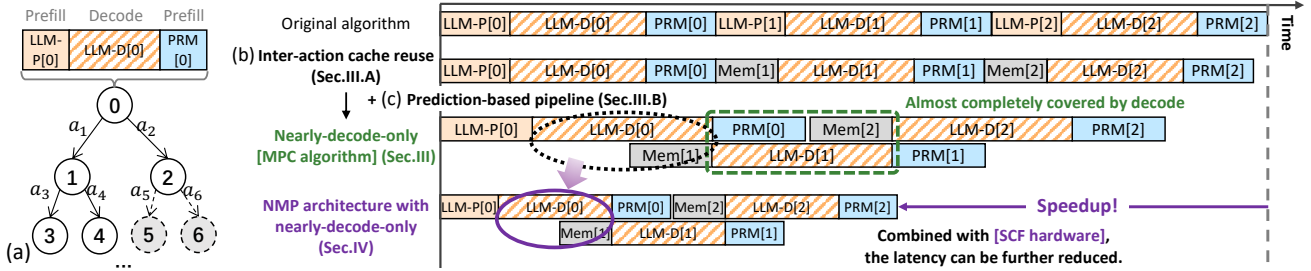


Fig. 3. Algorithm overview of Endor. (a) An example of LLM reasoning. (b) Inter-action cache reuse. (c) Prediction-based pipeline optimization.

action. Afterward, we concatenate the judge instruction, historical output records, and the current action to form the input for the PRM. The PRM is used to score the action, yielding a reward score. Finally, we assemble the child node using the action and the score.

C. Near-Memory Processing

Near-memory processing (NMP) is an emerging computational paradigm that significantly alleviates the memory wall problem. In NMP, computation occurs in locations that are closer to the memory, thereby reducing memory access latency and energy consumption. Common NMP approaches include processing in DRAM bank [19], in buffer chips on DDR DIMMs [20], [21], and on the logic die under the 3D-stacked DRAM dies [16], [22]. Prior works like [23], [24] have exploited the feasibility to accelerate LLM inference by placing NMP units near DRAM banks. Other work, like [25], implements an NMP system for efficient LLM inference. Besides, hybrid bonding techniques are also introduced by [26], [27] in optimizing LLM inference.

In this work, we design our architecture based on a hybrid approach, where NMP units are placed near DRAM banks and in buffer chips of DDR DIMMs. NMP units near DRAM banks primarily focus on performing GEMV operations during the decoding stages of LLMs. In contrast, other units in buffer chips handle functions such as reduction and special operations. Detailed architecture design will be introduced in Section IV.

III. NEARLY-DECODE-ONLY METHOD

A. Inter-action Cache Reuse

As shown in Fig. 2(b), the LLM reasoning relies on the action output (subquestion and subanswer) of the previous round in the inference process, which leads to a significant amount of computational redundancy (over 80%). Motivated by the inference characteristics of LLM reasoning, we design inter-action cache reuse method for inference acceleration in LLM reasoning. We observe that in LLM reasoning, the output of the previous action is part of the input for the next node, and updated content about input in different inference steps is the action. This means the input for a new node only adds the output of its parent node to the parent’s historical input. The model’s encoding of this new input can be achieved by splicing the encoding of the historical input with the encoding of the newly added output content. This entire process fully

follows the mathematical logic of the attention mechanism and is completely equivalent to the result of full re-encoding.

This inspires us to propose the nearly-decode-only method for LLM reasoning to reuse the cache due to the dependencies between interactions. As shown in Fig. 3(b), Previously, generating a new node required going through the prefill phase. With the nearly decode-only method, all the prefill computations of the LLM in reasoning (except for the root action) are eliminated. We divide caches into two types: the cache corresponding to the instruction and the action. These caches are composed of the KV values corresponding to the prefill stage of the previous inference and the KV values corresponding to the tokens generated during the decoding stage. In addition, we also reuse the KV cache corresponding to the input prefix of the PRM, as in conventional approaches.

B. Prediction-based Pipeline Optimization

The inference process of LLM reasoning often involves one LLM, which generates multiple candidate actions, and a critique PRM, which selects one action with the highest reward for further reasoning. Considering that the generation of subsequent actions depends on the selection of previous actions, traditional LLM reasoning often adopts a sequential reasoning process, which alternately calls the LLM and the PRM to advance reasoning. The dependency between the two models hinders the parallelism of hardware.

To break such dependency, we apply a speculative approach by leveraging auxiliary information to predict which action the PRM will choose, allowing the LLM to try to perform further reasoning without the PRM as shown in Fig. 3(c). Specifically, the log-likelihood of LLM-generated sequences quantifies the probability distribution over the LLM’s sequence selection process. Higher log-likelihood scores indicate that the LLM perceives these sequences as more likely to be “correct”, thereby increasing their probability of being selected by the PRM. So we use the log-likelihood of the LLM output sequence as a score for the current reasoning action, and select the action with the highest log-likelihood for further reasoning. The following formula can mathematically express this:

$$\mathcal{A}_i \leftarrow \text{LLM}(a_{i-1}, a_{i-2}, \dots, a_0)$$

$$\hat{a}_i = \arg \max_{a \in \mathcal{A}_i} \log \mathcal{L}_{\text{LLM}}(a|a_{i-1}, a_{i-2}, \dots, a_0)$$

Where a_i is the action at the i -th step, and \mathcal{A}_i is the set of candidate actions generated by the LLM. The term $\log \mathcal{L}_{\text{LLM}}$

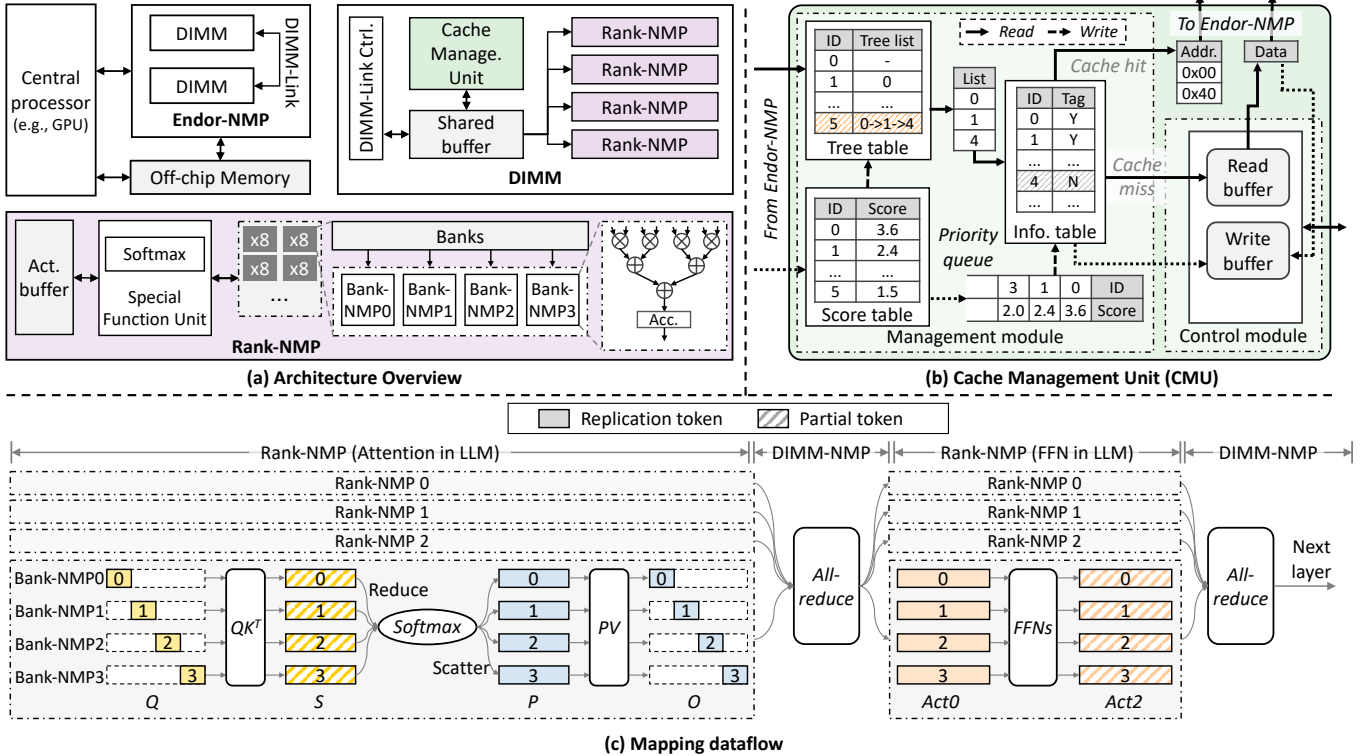


Fig. 4. (a) Hardware architecture overview of Endor-NMP. (b) The Cache Management Unit (CMU) alleviates the problem of low off-chip memory access due to cache misses. (c) The execution dataflow aligns the inherent parallelism between the model and Endor.

denotes the action log-likelihood, which is directly accessible during decoding. \hat{a}_i represents the predicted action used for lookahead. When the predicted action \hat{a}_i aligns with the PRM's selection (i.e., $a_i = \hat{a}_i$), reasoning proceeds seamlessly, leveraging model parallelism to mask the PRM's computational cost. If the prediction fails to match the PRM's selection, the speculative generation is discarded, and actions are regenerated based on the correct path. To optimize efficiency, we fully generate the discarded action only if it has a high probability of being reused in future tree search iterations; otherwise, decoding is terminated early.

IV. SCF-BASED NEAR-MEMORY ARCHITECTURE

A. Overview

We propose Endor to efficiently accelerate the LLM reasoning inference by exploiting the characteristics of both the algorithm and hardware. As shown in Fig. 4(a), Endor mainly consist of three components, including Endor-NMP, central processor, and off-chip memory. The central processor consists of high-performance hardware architectures (e.g., GPUs), which can well handle compute-bound operators (e.g., the prefill stage in LLMs). Off-chip memory is used to provide a larger storage capacity for holding the entire KV cache. The Endor-NMP architecture mainly includes two levels: rank-NMP and bank-NMP, with the rank-NMP modules on different DIMMs connected via DIMM-Link [28]. The Endor-NMP architecture mainly implements cache management and pipeline scheduling to support the previously proposed algorithm optimizations. Furthermore, a special functional unit (SFU) is designed in the

rank-NMP architecture to handle certain non-linear operations (e.g., softmax). The bank-NMP architecture is equipped with multiple sets of parallel GEMV computation units to fully utilize the high bandwidth inside the DIMM.

B. Score-aware Cache Management Architecture

As shown in Fig. 4(b), the Cache Management Unit (CMU) primarily performs two functions: processing the prediction results and replacing the required KV cache. When the CMU identifies that the currently computed action is incorrect, it will save the result of the current action to off-chip memory for backup, and then load the correct data into Endor-NMP. After that, the CMU will check whether there is a complete KV cache in the current Rank-NMP. The specific replacement strategy will be elaborated in the following.

1) *Cache Replacement Policy*: Although the inter-action cache strategy can almost eliminate the computational overhead in the LLM prefill stage, it consequently increases runtime memory access. In an entire inference process, the demand for inter-action cache is greater than $2\text{-}3\times$ the remaining DRAM's capacity, excluding the storage of weights (about 14GB). *It should be noted that with the increase of model parameters and the context length, there is an urgent need for an effective caching scheme.* Therefore, we need an appropriate replacement strategy to swap the cache between the buffer and off-chip memory, enabling dynamic memory management. Inspired by Huffman coding in information theory, we propose a metric for organizing the inter-action cache:

$$score = reward + \frac{\#reuse}{\#total} \quad (1)$$

The *score* is used to measure the priority of the cache block with the corresponding index, and the *reward* represents the reward value calculated by the PRM. The $\frac{\#reuse}{\#total}$ indicates the current reuse frequency of the action.

2) *Cache Management Unit*: We design the CMU to alleviate the problem of low off-chip memory access due to cache misses. The CMU primarily comprises a management module for managing read/write requests from the Endor-NMP and a control module for transferring the action KV cache between Endor-NMP and off-chip memory. The management module comprises a tree table, an info. table, a score table, and a priority queue. The tree table stores all the action IDs on the reasoning path of each action, which is used to collect historical action KV caches. The info. table outputs the addresses of the action KV caches in Endor-NMP (the hit case) or reads the corresponding data from off-chip memory (the miss case). The score table records the score of each action and updates the values after each action according to Eq. 1. Similarly, we also synchronously update the priority queue, which stores the IDs of the actions currently on Endor-NMP and their caches.

C. Mapping Dataflow

As shown in Fig. 4(c), we design a tailored dataflow on our architecture to meet the characteristics of the decode stage, aiming to align the inherent parallelism between the model and our architecture.

By employing DIMM-Link [28] for efficient communication and reduction among DIMMs, we can achieve TP (Tensor Parallelism) among DRAM ranks. In the Attention component, the input token is multiplied by three matrices to produce the corresponding q, k, v vectors, which will be sliced into several heads for individual computation. Due to the independent computation among heads, we assign these heads to different DRAM ranks. NMP units near DRAM banks are used for GEMV operations, while other operations like e^x and *Softmax* are performed by those NMP units in buffer chips. After that, we perform an all-reduce operation among the results in different DRAM ranks. In the FFN component, the weight matrices are evenly distributed among the DRAM ranks, each being responsible for part of the computation. Similar to the Attention component, GEMV operations are performed on near-bank NMP units, while the SFUs in buffer chips apply the activation function like SiLU. Also, results of the FFN component are collected and reduced, then ready for being inputs to the next layer.

V. EXPERIMENT SETUP

A. Evaluation Methodology

1) *Evaluation Setup*: To evaluate Endor performance, we develop a simulator based on Ramulator 2 [29], [30]. We align our parameter setup with the previous works [25]. To evaluate the additional hardware overhead introduced by Endor, we implemented prototypes of these modules using System Verilog.

TABLE I
HARDWARE SPECIFICATIONS.

NMP Cores	
Configurations	256 MACs, two softmax units, cache management unit
Buffer size	256KB
Frequency	1GHz
Area overhead	1.17 mm^2 per DIMM
DIMM Parameters	
Configuration	DDR4 4Gb x8 3200
Memory	8GB/DIMM \times 2, 2 DIMMs/Channel
Hierarchy	2 Ranks \times 4 Bank Groups \times 4 Banks
DRAM Timing	BL=4;CL=22;RCD=22;RP=22;RAS=52;RC=74; WR=24;RTP=12;CWL=16;CCD=4/8;RRD=4/6;WTR=4/12

TABLE II
INFERENCE-RELATED STATISTICS ACROSS DATASETS

Dataset	GSM8K [8]	AQuA [37]
Average Instruction Length	1503	1206
Average Action Length	86	42
Average Node Number	70	

We used Synopsys Design Compiler [31] to assess the area under the TSMC 28nm CMOS process with a 1GHz frequency. The hardware modules of Endor only occupy 1.17 mm^2 , representing a negligible additional overhead compared with the area of a DIMM [15], [32]. The details are summarized in Table I. Besides, we use DIMM-Link [28] for efficient communication among different DIMMs. We have 25Gb/s/Lane and 8 \times Lanes, finally getting a 25GB/s bandwidth inter-DIMM.

2) *Baselines*: We select three representative hardware as baselines: general-purpose hardware, ASIC-based LLM accelerators, and NMP-based LLM accelerators. (1) We choose NVIDIA A100 GPU [33] as the general-purpose hardware baseline and run several LLM reasoning tasks to obtain the execution time. (2) For ASIC-based LLM accelerators, we select two SOTA works, DFX [34] and FlightLLM [35]. (3) We also consider Hermes [25], a state-of-the-art GPU-PIM system, where NMP units are placed in buffer chips.

B. Models and Datasets

1) *Models*: We evaluate the OpenR [2] and LLM-Reasoner [1] methods for different baselines. The reason for selecting them as algorithmic baselines is that these two algorithms represent the typical reasoning paradigm scenario: RL-driven (*e.g.*, OpenR) and MCTS-driven (*e.g.*, LLM-Reasoner). This enables comprehensive verification of the architecture’s adaptability to LLM reasoning.

2) *Datasets*: We choose the MATH [36] and GSM8K [8] datasets for OpenR. The MATH dataset is an open-source collection that measures a model’s ability to solve mathematical problems. The GSM8K dataset is specifically designed to evaluate model performance in multi-step mathematical reasoning. For LLM-Reasoner, we use AQuA [37] and GSM8K datasets. The AQuA dataset provides a set of algebraic problems and options, requiring the LLM to compute the problem and select the correct option. The average number of nodes during inference and the average lengths of instructions and actions related to input and output are shown in the Table II.

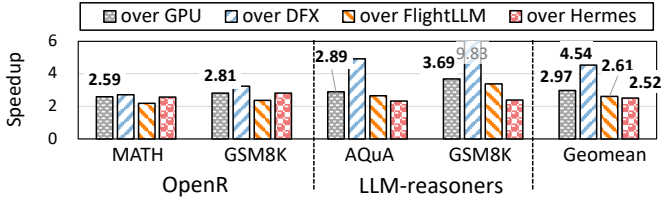


Fig. 5. The comparison of overall performance.

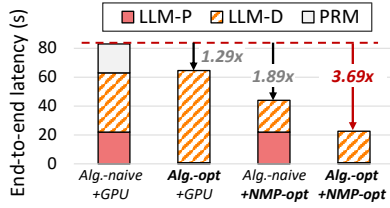


Fig. 6. Latency breakdown for LLM-D, LLM-P, and PRM with different optimization setups (LLM-Reasoner [1] on GSM8K [8]).

VI. EVALUATION

A. Performance Evaluation

1) *End-to-end Speedup*: As shown in Fig. 5, we evaluate Endor on multiple models and datasets. Endor achieves an average acceleration of $2.97\times$, $4.54\times$, $2.61\times$, and $2.52\times$ compared to NVIDIA A100 GPU, DFX, FlightLLM, and Hermes, respectively. GPU and DFX can handle the prefill computations in the prefill stage quite well, but their overall bottleneck lies in the decode computations. FlightLLM is equipped with a dedicated decode unit and achieves a $1.68\times$ acceleration compared to the A100 GPU. Further, Hermes, by leveraging its larger internal memory bandwidth, is more suitable for handling LLM reasoning than other hardware architectures. In contrast to these works, Endor achieves a significant acceleration thanks to the computation-reduced algorithm and memory-enhanced hardware design approach.

2) *Performance breakdown*: Fig. 6 shows the latency breakdown for LLM-D, LLM-P, and PRM across different optimization setups on Endor. We find that optimizing only the algorithm or using only the NMP architecture achieves limited speedups of $1.29\times$ and $1.89\times$, respectively. The *Alg.-opt* and *NMP-opt* co-design eliminate LLM-P latency, overlap reduced PRM latency with LLM-D, and achieve a $3.69\times$ speedup over the naive algorithm on the GPU architecture.

B. Ablation Studies

1) *Benefits of algorithm optimizations*: We compare the acceleration effects of our algorithm optimizations on the same hardware setup. As shown in Fig. 7(a), with our nearly-decode-only algorithm, NMP achieves a speedup ranging from $1.93\times$ to $2.82\times$, with an average speedup of $2.29\times$ compared with the original algorithm. We contributed to this by eliminating redundant computation, reducing data transfer provided by the inter-action cache management method, and providing more parallelism for the hardware to hide latencies.

2) *Benefits of hardware optimizations*: We also demonstrated the feasibility of our hardware optimization through ablation experiments. As shown in Fig. 7(b), it achieves an

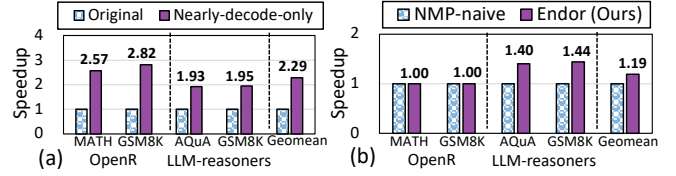


Fig. 7. Ablation studies for our (a) algorithm and (b) hardware optimization.

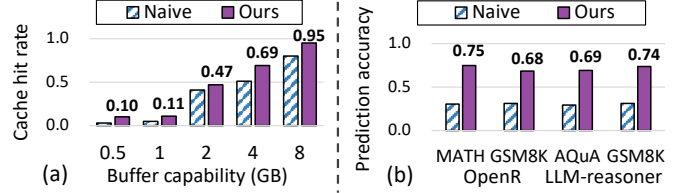


Fig. 8. Analysis for (a) the hit rate of score-based cache replacement strategy and (b) prediction accuracy.

average speedup of $1.19\times$ and up to $1.44\times$ across four datasets compared to the naive NMP implementation. We attribute this performance enhancement to the reduction in inter-chip data transfer overhead and the increase in computational parallelism, which collectively improve the utilization of on-chip computational resources and DRAM memory bandwidth.

C. Analysis of Algorithm Effectiveness

As shown in Fig. 8 (a), we analyze the cache hit rate under different buffer capacities using the score-based cache replacement strategy, with random replacement as a naive baseline. With increasing buffer capacity, the hit rates of both strategies keep rising. Under the same buffer capacity, our score-based cache replacement strategy achieves an average increase of 0.10 compared to the naive strategy. As shown in Fig. 8 (b), we evaluate prediction accuracy across multiple reasoning methods and datasets. Using log-likelihood achieves an average accuracy of 0.72. Compared to the naive method that relies only on the previous round’s score, our approach improves prediction accuracy by 0.41.

VII. CONCLUSION

In this paper, we propose *Endor*, a computation-reduced algorithm and memory-enhanced hardware co-design to accelerate the inference of LLM reasoning efficiently. We propose an NMP-based accelerator based on the nearly-decode-only method for LLM reasoning. Endor achieves an end-to-end acceleration of average $2.97\times$ compared to the NVIDIA A100 GPU on multiple models and datasets.

ACKNOWLEDGMENT

This work was sponsored by the National Natural Science Foundation of China (No. 62561160156, 62325405, U24B6015), Shanghai Rising-Star Program (No. 24QB2706200), Beijing Natural Science Foundation (L257010), ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR (HKSAR), and Research Grants Council of HKSAR (16213824).

REFERENCES

- [1] S. Hao, Y. Gu, H. Luo, T. Liu, X. Shao, X. Wang, S. Xie, H. Ma, A. Samavedhi, Q. Gao, Z. Wang, and Z. Hu, "Llm reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models," 2024.
- [2] J. Wang, M. Fang, Z. Wan, M. Wen, J. Zhu, A. Liu, Z. Gong, Y. Song, L. Chen, L. M. Ni, L. Yang, Y. Wen, and W. Zhang, "Openr: An open source framework for advanced reasoning with large language models," 2024.
- [3] D. Zhang, S. Zhoubian, Z. Hu, Y. Yue, Y. Dong, and J. Tang, "Rest-mcts*: Llm self-training via process reward guided tree search," 2024.
- [4] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24824–24837, 2022.
- [5] X. Ning, Z. Lin, Z. Zhou, Z. Wang, H. Yang, and Y. Wang, "Skeleton-of-thought: Prompting LLMs for efficient parallel generation," in *The Twelfth International Conference on Learning Representations*, 2024.
- [6] C. Snell, J. Lee, K. Xu, and A. Kumar, "Scaling llm test-time compute optimally can be more effective than scaling model parameters," *arXiv preprint arXiv:2408.03314*, 2024.
- [7] OpenAI, "Learning to reason with llms." <https://openai.com/index/learning-to-reason-with-llms/> Accessed Sept 12, 2024.
- [8] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, *et al.*, "Training verifiers to solve math word problems, 2021," URL <https://arxiv.org/abs/2110.14168>, 2021.
- [9] X. Ning, Z. Wang, S. Li, Z. Lin, P. Yao, T. Fu, M. B. Blaschko, G. Dai, H. Yang, and Y. Wang, "Can llms learn by teaching for better reasoning? a preliminary study," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- [10] OpenAI, "Openai o1-mini advancing cost-efficient reasoning." <https://openai.com/index/openai-o1-mini-advancing-cost-efficient-reasoning/> Accessed Sept 12, 2024.
- [11] T. Fu, H. Huang, X. Ning, G. Zhang, B. Chen, T. Wu, H. Wang, Z. Huang, S. Li, S. Yan, *et al.*, "Moa: Mixture of sparse attention for automatic large language model compression," *arXiv preprint arXiv:2406.14909*, 2024.
- [12] S. Li, X. Ning, L. Wang, T. Liu, X. Shi, S. Yan, G. Dai, H. Yang, and Y. Wang, "Evaluating quantized large language models," *arXiv preprint arXiv:2402.18158*, 2024.
- [13] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," 2023.
- [14] G. Dai, Z. Zhu, T. Fu, C. Wei, B. Wang, X. Li, Y. Xie, H. Yang, and Y. Wang, "Dimming: pruning-efficient and parallel graph mining on near-memory-computing," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pp. 130–145, 2022.
- [15] Z. Zhu, J. Liu, G. Dai, S. Zeng, B. Li, H. Yang, and Y. Wang, "Processing-in-hierarchical-memory architecture for billion-scale approximate nearest neighbor search," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2023.
- [16] Zhiheng Yue, Huizheng Wang, Jiahao Fang, Jinyi Deng, Guangyang Lu, Fengbin Tu, Ruiqi Guo, Yuxuan Li, Yubin Qin, Yang Wang, Chao Li, Huiming Han, Shaojun Wei, Yang Hu, and Shouyi Yin, "Exploiting Similarity Opportunities of Emerging Vision AI Models on Hybrid Bonding Architecture," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 396–409, June 2024.
- [17] Y. Li, F. Wei, C. Zhang, and H. Zhang, "Eagle: speculative sampling requires rethinking feature uncertainty," in *Proceedings of the 41st International Conference on Machine Learning*, pp. 28935–28948, 2024.
- [18] Z. Zhou, X. Ning, K. Hong, T. Fu, J. Xu, S. Li, Y. Lou, L. Wang, Z. Yuan, X. Li, *et al.*, "A survey on efficient inference for large language models," *arXiv preprint arXiv:2404.14294*, 2024.
- [19] Peng Gu, Xinfeng Xie, Yufei Ding, Guoyang Chen, Weifeng Zhang, Dimin Niu, and Yuan Xie, "iPIM: Programmable In-Memory Image Processing Accelerator Using Near-Bank Architecture," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 804–817.
- [20] J. Park, B. Kim, S. Yun, E. Lee, M. Rhu, and J. H. Ahn, "TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, pp. 268–281, Association for Computing Machinery.
- [21] S. Feng, X. He, K.-Y. Chen, L. Ke, X. Zhang, D. Blaauw, T. Mudge, and R. Dreslinski, "MeNDA: A near-memory multi-way merge solution for sparse transposition and dataflows," in *Proceedings of the 49th Annual International Symposium on Computer Architecture, ISCA '22*, pp. 245–258, Association for Computing Machinery.
- [22] X. Xie, Z. Liang, P. Gu, A. Basak, L. Deng, L. Liang, X. Hu, and Y. Xie, "SpaceA: Sparse Matrix Vector Multiplication on Processing-in-Memory Accelerator," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 570–583.
- [23] H. Lee, D. Baek, J. Son, J. Choi, K. Moon, and M. Jang, "Paise: Pim-accelerated inference scheduling engine for transformer-based llm," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1707–1719, IEEE, 2025.
- [24] Y. Gu, A. Khadem, S. Umesh, N. Liang, X. Servot, O. Mutlu, R. Iyer, and R. Das, "Pim is all you need: A cxl-enabled gpu-free system for large language model inference," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 862–881, 2025.
- [25] L. Liu, S. Zhao, B. Li, H. Ren, Z. Xu, M. Wang, X. Li, Y. Han, and Y. Wang, "Make llm inference affordable to everyone: Augmenting gpu memory with ndp-dimm," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1751–1765, IEEE, 2025.
- [26] W. Sun, M. Gao, Z. Li, A. Zhang, I. Y. Chou, J. Zhu, S. Wei, and L. Liu, "Lincoln: Real-time 50' 100b llm inference on consumer devices with lpdrr-interfaced, compute-enabled flash memory," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1734–1750, IEEE, 2025.
- [27] C. Li, Y. Yin, X. Wu, J. Zhu, Z. Gao, D. Niu, Q. Wu, X. Si, Y. Xie, C. Zhang, *et al.*, "H2-llm: Hardware-dataflow co-exploration for heterogeneous hybrid-bonding-based low-batch llm inference," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pp. 194–210, 2025.
- [28] Z. Zhou, C. Li, F. Yang, and G. Sun, "Dimm-link: Enabling efficient inter-dimm communication for near-memory processing," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 302–316, IEEE, 2023.
- [29] H. Luo, Y. C. Tuğrul, F. N. Bostancı, A. Olgun, A. G. Yağlıkcı, and O. Mutlu, "Ramulator 2.0: A modern, modular, and extensible dram simulator," *IEEE Computer Architecture Letters*, vol. 23, no. 1, pp. 112–116, 2023.
- [30] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.
- [31] I. Synopsys, "Design compiler (synopsys.com)." <https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html> Accessed May 22, 2023.
- [32] Q. Liu, L. Chen, Y. Yang, H. Wang, D. Du, Z. Mao, N. Jing, Y. Xia, and H. Chen, "L3: Dimm-pim integrated architecture and coordination for scalable long-context llm inference," *arXiv preprint arXiv:2504.17584*, 2025.
- [33] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "Nvidia a100 tensor core gpu: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021.
- [34] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 616–630, IEEE, 2022.
- [35] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang, *et al.*, "Flightllm: Efficient large language model inference with a complete mapping flow on fpgas," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 223–234, 2024.
- [36] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, "Measuring mathematical problem solving with the math dataset," *arXiv preprint arXiv:2103.03874*, 2021.
- [37] W. Ling, D. Yogatama, C. Dyer, and P. Blunsom, "Program induction by rationale generation: Learning to solve and explain algebraic word problems," *arXiv preprint arXiv:1705.04146*, 2017.