Efficient Deployment of Large Language Model across Cloud-Device Systems

Fan Yang[1,2*], Zehao Wang[1*], Haoyu Zhang[1], Zhenhua Zhu[1], Xinhao Yang[1], Guohao Dai[3], Yu Wang[1†]
[1]Tsinghua University, [2]SenseTime Inc., [3]Shanghai Jiao Tong University
[†]Corresponding author: yu-wang@tsinghua.edu.cn

***Abstract*—The capabilities of large language models (LLMs) in text comprehension and generation are advancing artificial intelligence. However, the growing number of parameters and computational demands challenge the efficient deployment of inference services. High-performance GPU clusters in the cloud can meet these requirements but incur high service costs and network stability issues, which struggle to meet service-level agreements (SLAs). The "cloud-device collaboration" approach leverages the heterogeneous hardware on both the cloud and device sides to satisfy SLAs efficiently. However, the varying operational intensity among different LLM operators and their dynamic nature complicate load scheduling for cloud-device systems.**

To address these challenges, we optimize LLM inference deployment on cloud-device systems through three aspects: scheduling algorithm, hardware modeling, and compilation deployment. For the scheduling algorithm, we analyze the LLM computation network, evaluate the computation-to-memory access ratio under different sequence lengths, and propose a greedy algorithm-based operator-level scheduling strategy. For the hardware modeling, we establish a relationship between operational intensity and GPU resource utilization to estimate operator running time. Finally, we designed a cloud-device LLM compiler framework for quantitative evaluation and efficient deployment across various hardware combinations and inference tasks. In specific inference scenarios, our framework satisfies the need for inference latency and achieves an average cost reduction of 20.7% compared to cloud-side-only inference.

***Index Terms*—Large Language Models, Cloud-Device Collaboration System, Efficient Inference**

I. Introduction

In recent years, large language models (LLMs) have revolutionized natural language processing tasks, such as language understanding, generation, and question answering, achieving widespread adoption across various industries [1]–[3]. However, the substantial computational requirements and parameter sizes of LLMs pose significant challenges for their efficient deployment. For example, GPT-3 released by OpenAI [4] has 175 billion parameters and requires about 350GB of memory (using FP16 data type), with approximately 660TOPs needed to complete one inference. Despite the efforts to optimize LLMs through techniques such as quantization and pruning to reduce the computational and memory requirements for inference, the end-side devices are still hard to deploy these LLMs. High-performance GPU clusters in the cloud can satisfy these requirements but incur significant hardware costs and rely on stable network connectivity. This dependency can hinder real-time performance and limit service accessibility in remote or disconnected environments, which means a struggle to meet service-level agreements (SLAs).

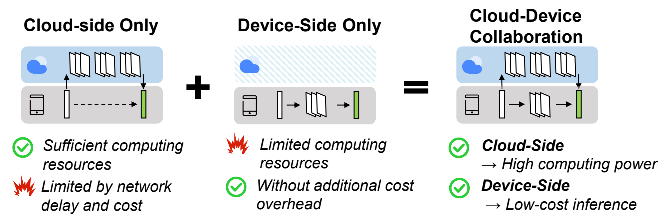
Cloud-side Only
Device-Side Only
Cloud-Device Collaboration
Sufficient computing resources
Limited by network delay and cost
Limited computing resources
Without additional cost overhead
Cloud-Side → High computing power
Device-Side → Low-cost inference

Fig. 1. device-side-only, cloud-side-only and cloud-device collaborative inference schemes.

To address the limitations of cloud-side and device-side, a cloud-device heterogeneous system can be adopted to efficiently deploy LLMs, as shown in Fig. 1. For Convolutional Neural Networks (CNNs), some efforts have been made to optimize the deployment on cloud-device systems, typically through large-small model collaboration and computation offloading [5]. Large-small model collaboration relies on techniques such as early stopping, where only a portion of the model is computed under constrained resources. However, this approach is not suitable for current LLMs. Computation offloading, which includes layer-wise and tensor-wise partitioning strategies, also fails to achieve optimal performance for LLMs. The diverse characteristics (e.g., computation-to-memory access ratios) of operators and the dynamism of workload characteristics (e.g., the length of input and output texts) make effective partitioning and offloading difficult. It is necessary to propose an approach to adaptively partition and distribute computational tasks based on real-time workload demands and resource availability.

In this paper, we propose a framework to efficiently deploy LLMs inference services in cloud-device systems under diverse hardware configurations and varying inference workloads. This framework divides computational tasks at the operator level for scheduling. Specifically, we conduct a quantitative analysis of the computation-to-memory access ratio of different operators (e.g., attention and linear layer) in LLMs and their dynamic changes relative to the input and output text lengths. Based on this analysis, we propose a cost- and latency-aware greedy-based scheduling algorithm. To evaluate the execution time of operators efficiently, we develop a hardware simulation model that considers bandwidth, computing power, and the computation-to-memory access ratio. Finally, we design a compilation framework that enables rapid evaluation and efficient deployment across different hardware combinations and dynamic inference tasks.

[*]Both authors contributed equally to this work.

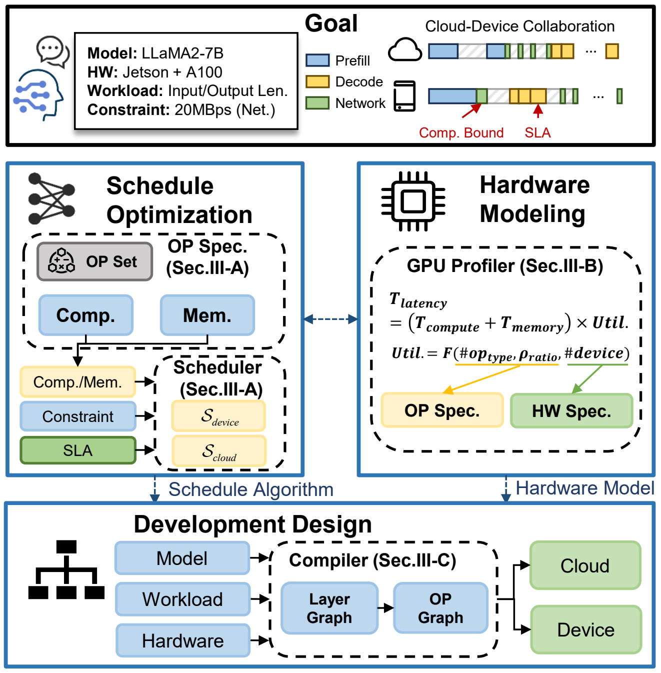

Fig. 2. The overview of our design. Our goal is to ensure that the inference latency (TTFT and TPOT) is within the inference SLA, while minimizing the inference cost.

We demonstrate an overview of this framework in Fig. 2 and summarize the contributions as follows:

1) We introduce a novel framework for efficient deployment of LLMs in cloud-device systems. Specifically, we partition the computation load at the operator granularity and propose a cost-latency balancing scheduling algorithm based on greedy algorithm.
2) We develop a hardware simulation model that enables rapid evaluation of the computation time for operators with varying computation-to-memory access ratios across diverse hardware platforms.
3) We present a compilation framework that facilitates quantitative evaluation and efficient deployment of diverse inference tasks and cloud-device hardware combinations. Experimental results demonstrate that our approach satisfies the need for inference latency and reduces costs up to 20.7% compared to cloud-side-only deployment.

II. Background and Related Work

A. Large Language Models (LLMs) Inference

In this section, we introduce the LLMs inference computation workflow and SLAs. A standard LLM generative inference process comprises two stages [6]: i) the *prefill stage*, where a given prompt sequence is used to generate the key/value cache (KV cache) for each transformer layer within the LLM; and ii) the *decode stage*, which leverages and updates this KV cache to produce tokens one by one, with the current token generation depending on previously generated tokens. In both stages, the generation of each token requires passing through the entire model, typically constructed by stacking multiple transformer blocks [7]. Transformer blocks comprise two primary operators: the attention operator and the Feed-Forward Network (FFN) operator.

During *prefill stage*, denote the input features of the i-th layer as $X_i \in R^{b\times s\times h}$, where b denotes the batch size, s denotes the input sequence length, and h denotes the hidden dimension of the transformer. For attention operator, the computation is:

$$Q_i = X_i W^{Q_i}, K_i = X_i W^{K_i}, V_i = X_i W^{V_i} \tag{1}$$

$$A_i = softmax(\frac{Q_i K_i^T}{\sqrt{h}})V_i,\ O_i = A_i W^{O_i} + X_i \tag{2}$$

where $W^{Q_i}, W^{K_i}, W^{V_i}, W^{O_i} \in R^{h\times h}$ are weight matrices. The computation equation of FFN operator is:

$$X_{i+1} = f_{act}(O_i W_1^i) W_2^i + O_i \tag{3}$$

where $W_1^i \in R^{h\times h'}$ and $W_2^i \in R^{h'\times h}$ are weight matrices (h' is the hidden dimension of the second MLP layer).

During *decode stage*, each iteration is designed only to consider the current token. Different from *prefill stage*, the matrix-vector multiplications (MV) operations are the majority for decoding instead of the matrix-matrix multiplications (MM). Denote the feature of the current generated token in the i-th layer as $t_i \in R^{b\times 1\times h}$. The equations for this process are below:

$$q_i = t_i \cdot W^{Q_i}, k_i = t_i \cdot W^{K_i}, v_i = t_i \cdot W^{V_i} \tag{4}$$

$$K_i \leftarrow \text{Concat}\left(K_i, k_i\right), V_i \leftarrow \text{Concat}\left(V_i, v_i\right) \tag{5}$$

$$a_i = softmax(\frac{q_i K_i^T}{\sqrt{h}})V_i,\ o_i = a_i \cdot W^{O_i} + t_i \tag{6}$$

$$t_{i+1} = f_{act}(o_i \cdot W_1^i) W_2^i + o_i \tag{7}$$

For LLMs inference services, SLAs commonly employ two metrics [8]: i) Time to the First Token (TTFT), which refers to the latency incurred from a query is input until the first output token is generated, and ii) Time Per Output Token (TPOT), which measures the average time required to generate each subsequent output token after the first one. TTFT is related to *prefill stage* and TPOT is decided by *decode stage*.

B. Cloud-Device Inference System

Existing approaches to deploying model inference services in cloud-device systems can be broadly categorized into two categories: i) large-small model collaboration [9]–[12], which leverages the collaboration between large models deployed in the cloud and small models residing at the devices, and ii) computation offloading [13]–[17], which dynamically allocates computational tasks between the cloud and the devices.

For large-small model collaboration, one common technique is the early-exit mechanism, which allows the model to terminate its inference process prematurely when a certain level of confidence or accuracy threshold is met. For example, SPINN [9] dynamically selects the exit points and task division between devices and clouds based on resources and network conditions. Another common method is knowledge distillation, where a large, complex model in the cloud is

TABLE I
OPERATOR COMPUTATION, MEMORY ACCESS AND COMPUTATIONAL INTENSITY IN PREFILL AND DECODE STAGES.

Operator Name	Stage	Computation (OPs)	Memory Access (Bytes)	Compute-to-Memory Access Ratio (OP/Byte)
$Q/K/V/O$ (Linear)	Prefill*	$s_{in}h^2$	$2s_{in}h + h^2$	$s_{in}h/(h + 2s_{in})$
	Decode**	h^2	$2h + h^2$	$h/(h + 2)$
$S \cdot V$ (Attention)	Prefill	$s_{in}^2 h$	$2s_{in}h + s_{in}^2$	$s_{in}h/(2h + s_{in})$
	Decode	$(s_{in} + 1)h$	$h + (s_{in} + 1)h + s_{in} + 1$	$(s_{in} + 1)h/[h + (s_{in} + 1) \cdot (h + 1)]$
FFN	Prefill	$s_{in}hh'$	$s_{in}h + s_{in}h' + hh'$	$s_{in}hh'/[s_{in}(h + h') + hh']$
	Decode	hh'	$h + h' + hh'$	$hh'/(h + h' + hh')$
LM Head	Decode	hh_v	$2h + hh_v$	$hh_v/(2h + hh_v)$

* s_{in} represents the prefill size.
** The decode here is the first stage after the prefill stage.

used to train a smaller, more efficient model on the device-side. However, for LLMs, which are characterized by immense size and complexity, it is hard to construct a model supporting early-exit or distilling knowledge.

For computation offloading, the two primary methods are tensor parallelism and pipeline parallelism. For example, Co-Edge [13] proposes a tensor offloading strategy, which divides the computation workloads at the tensor level and allocates them between clouds and devices. Some works partition tasks in tensor and model levels [15]–[17], using tensor parallelism and pipeline parallelism simultaneously. However, these works cannot fully consider the different computation-to-memory access ratios of different operators and the dynamics of workloads, which compromises the efficiency of the service.

III. METHODOLOGY

In this section, we present our proposed framework for the efficient deployment of LLM inference services. We introduce our innovative scheduling strategy (Sec. III-A), tailored to dynamically allocate computational tasks between the cloud and devices, ensuring optimal performance under varying workloads. Then we elaborate on our hardware modeling and simulation methodology (Sec. III-B). Lastly, we outline our compilation and deployment system, which streamlines the process of adapting and deploying LLMs across diverse architectures (Sec. III-C).

A. Schedule Optimization Based on Operator Properties

Instead of layer-wise partition of the computational workload, we segment the load at operator granularity. That is because of the heterogeneity observed in the characteristics of various operators during the computation process of LLMs. When confronted with the SLAs violation imposed by resource-constrained devices, our algorithm would prioritizes the migration of operators with high computational intensity and memory access requirements to the cloud. This is driven by the fact that GPUs on the cloud side exhibit significantly higher utilization when handling operators with substantial workloads. This strategic relocation not only mitigates the resource limitations at the device-side, but also strives to minimize the associated cloud costs, ensuring that the SLA targets are met with minimal overhead.

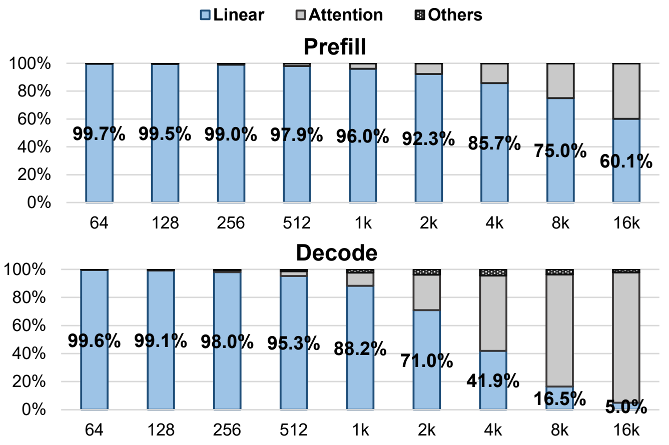

Fig. 3. The proportion of computing time requirements of different operators in the prefill and decode stages varies with the length of the processed text. The horizontal axis represents the length of the processed text.

Fig. 3 illustrates the computation timeshare of different operators in inference for LLMs. It can be observed that the linear and attention operators dominate the inference process and that the distribution is affected by the inference stages and the length of text. In the context of short text inference requests, linear operators predominantly contribute to the overall latency during both the prefill and decode phases. However, the share of attention computation gradually increases as the text sequence grows in length, with this trend being more pronounced in the decode stage.

Table I lists the equation of computation load, memory access volume and computation-to-memory access ratio for linear and attention operators. It can be observed that the computation load of attention operators is correlated with the square of the text length. In contrast, it grows linearly with the length of the text for linear operators, which also explains the change in the workload proportion of Fig. 3. Regarding the performance constraints, the prefill phase is characterized by a computational bottleneck, whereas the decode stage is marked by a bandwidth bottleneck. For prefill stage, the computation-to-memory access ratio is usually much greater than 1. In contrast, for decode stage, the computation-to-memory access ratio of the linear operators is approximately equal to 1 and independent of the text length. Similarly, as the text length increases, the computation-to-memory access ratio of the attention operators also converges to 1. Moreover,

Algorithm 1 Greedy-based algorithm for tasks scheduling on cloud-device system

```
1: Input: The LLM inference computation tasks set in current iteration $\mathcal{S}$, parameters of cloud and device platforms $D_{cloud,device}$, target latency $\tau$.
2: Output: Cloud-side tasks subset $\mathcal{S}_{cloud}$, device-side tasks subset $\mathcal{S}_{device}$.
3: Sort $\mathcal{S}$ by linear and attention.
4: $\mathcal{S}_{cloud} \leftarrow \emptyset$
5: $\mathcal{S}_{device} \leftarrow \mathcal{S}$
6: $T \leftarrow 0$
7: for $s \in \mathcal{S}$ do
8:     $T = T + f(s, D_{device})$
9: end for
10: for $s \in \mathcal{S}$ do          ▷ Traverse linear first, then attention
11:     if $T < \tau$ then
12:         break
13:     else
14:         $\mathcal{S}_{device} \leftarrow \mathcal{S}_{device} \setminus s$
15:         $\mathcal{S}_{cloud} \leftarrow \mathcal{S}_{cloud} \cup s$
16:         $T = T - f(s, D_{device}) + f(s, D_{cloud})$
17:     end if
18: end for
19: return $\mathcal{S}_{cloud}$, $\mathcal{S}_{device}$
```

the computational complexity and memory access volume of operators during the prefill stage significantly exceed those in the decode stage, primarily due to the fact that the prefill stage processes multiple input tokens simultaneously, whereas the decode stage handles a single output token at a time.

Based on the above observations, we propose a greedy-based scheduling algorithm, as illustrated in Alg. 1. The objective of the algorithm is to allocate operators ($\mathcal{S}$) required by LLM within a specified iteration cycle, aiming to minimize the cost, i.e., minimize the use of cloud hardware, while satisfying latency constraints (τ). Initially, the algorithm assumes all tasks are assigned to the device ($\mathcal{S}_{device} = \mathcal{S}$) and estimates the total latency T using a function $f(s, D_{device})$. This function is obtained from hardware modelling, which computes latency considering operator and hardware parameters. Following this, the algorithm iterates through the task set, assessing whether the cumulative latency T would exceed the target τ under the current task assignment. If so, the algorithm offloads the current task to the cloud by moving it from $\mathcal{S}_{device}$ to $\mathcal{S}_{cloud}$ and updates the T. This process continues until the latency constraint τ is met. In this algorithm, we prioritize traversing and assigning linear operators to the cloud before attention operators. This decision stems from the strategic objective of migrating operators with higher computational and memory access requirements to the cloud.

Regarding the communication between the cloud and the device, when the cloud processes linear operators, only activations need to be transmitted, resulting in a relatively low communication volume that can be effectively concealed through pipelining. When the cloud computes attention operators, given that we have already prioritized the placement of linear layers on the cloud, the KV cache required by the attention operators is expected to be already resident in the cloud. As a result, only activations need to be transmitted maintaining a manageable communication overhead.

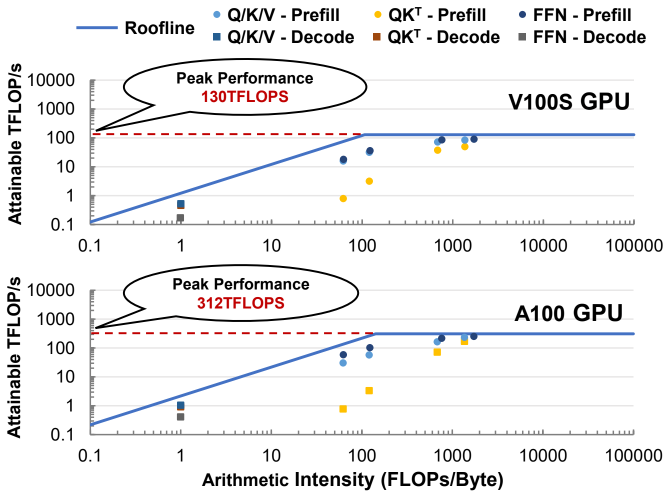

Fig. 4. NVIDIA V100S GPU and A100 GPU theoretical roofline curve and actual computational intensity of different operators.

B. Hardware Modeling for The GPU

Modern GPU architectures are uniform in design and have commonalities in the operation mechanism. Considering the GPU's pipeline design [18], the theoretical computation latency of the GPU can be calculated using the following Eq. 8:

$$\begin{aligned} T_{GPU} &= \sum_{i=0}^{N_{op}} max(T^{(i)}_{Compute}, T^{(i)}_{Memory}) \\ &= \sum_{i=0}^{N_{op}} max(\frac{Operation_i}{P_i \times Parallelism_i}, \frac{Data\ Size_i}{BW_i}) \end{aligned} \tag{8}$$

where N_{op} is the number of operators of the task, $Operation_i$ is the computation amount of different operations within the operator, P_i is the computing power of the operation specified by the GPU, and $Data\ Size_i$ indicates the amount of data accessed, and BW_i indicates the bandwidth of data accessed.

Fig. 4 shows the roofline model of the GPU and the attainable performance in running different inference operators. It is demonstrated that for operators with low computational intensity, the attainable performance of GPUs falls short of their theoretical peak performance. As a result, when theoretical models are used for derivations, substantial deviations can occur in inference tasks.

In this paper, we employ spline interpolation on empirically measured data to formulate an accurate hardware modeling approach. Specifically, by applying spline interpolation to the collected data points, we define a practical modeling formula that accurately reflects the hardware's performance. Notably, under diverse task loads, the average absolute error of the simulation for a single operator is found to be 4.78%, demonstrating the effectiveness and accuracy of our modeling strategy.

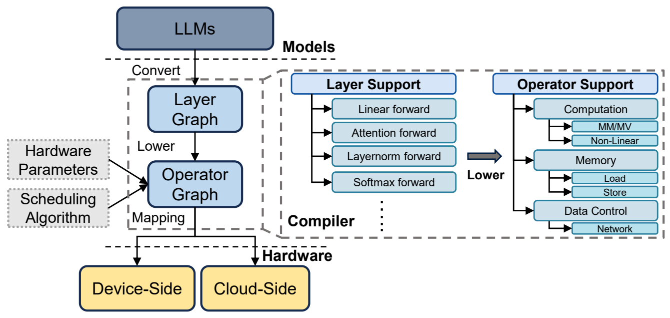

Fig. 5. Cloud-device LLMs compilation workflow.

C. Development Design for Cloud-Device Collaboration

In conjunction with the cloud-device collaboration system, LLMs' inference scenarios exhibit diverse and complex trends, primarily manifested in the interplay between hardware (device-side computing power and inference cost) and inference task (prefill size and decode size). This necessitates the deployment of convenient tool frameworks to facilitate software-to-hardware mapping.

We design a two-layer intermediate representative (IR) compilation process to achieve a more fine-grained optimization purpose. This process implements a structural analysis of the model, an operator dependency analysis, and an optimization of the cloud-device scheduling strategy, as summarized in Fig. 5.

In the layer computation graph IR, the compiler can realize the abstract representation of the computation layers and complete the dependency analysis. Further, the layer computation graph IR lowers for the operator computation graph IR, where the compiler will define specific operator computation properties, access properties, and data control properties for different computation monolayers, which can be directly utilized by the hardware model in Sec.III-B. The compiler also assigns the execution platform for each operator, adds network communication operations, and completes the inference scheduling optimization in conjunction with the algorithm mentioned in Sec. III-A.

After compilation, we get the set of tasks on the cloud-device computing platform. In addition, while obtaining the operator scheduling strategy, we can also calculate the inference performance of the cloud-device system based on the hardware model simulation.

IV. Experiment

A. Experiment Setup

Hardware Platform. To demonstrate the effect of the design works with different hardware combinations, we experimented with two different configurations:

a) Group-L: For the cloud-side platform, this configuration includes an NVIDIA V100S GPU with a peak memory bandwidth of 1134GB/s and a peak computing power of 130TFLOPS using Tensor Core at FP16 precision. The cost of use is 1.5×10^{-5} \$/s. For the device-side platform, we chose an NVIDIA Jetson Nano with a peak memory bandwidth of 25.6GB/s and a peak computing power of 0.47TFLOPS (GPU).

TABLE II
Hardware parameters of Cloud and Device platforms.

	Cloud-Side		Device-Side	
Platform	NVIDIA V100S	NVIDIA A100	NVIDIA Jetson Nano	Snapdragon 8s Gen 3
Frequency	1597MHz	1410MHz	1430MHz	921MHz
Computing Power	130TFLOPS (Tensor Core)	312TFLOPS (Tensor Core)	0.47TFLOPS (GPU)	8.50TFLOPS (GPU)
Bandwidth	1134GB/s	2039GB/s	25.6GB/s	76.6GB/s

b) Group-H: For the cloud-side platform, this configuration includes an NVIDIA A100 GPU with a peak memory bandwidth of 2039GB/s and a peak computing power of 312TFLOPS using Tensor Core at FP16 precision. The cost of use is 4×10^{-5} \$/s. We choose the Snapdragon 8s Gen 3 for the device-side platform with a peak memory bandwidth of 76.6GB/s and a peak computing power of 8.5TFLOPS (GPU).

Models. We use the the LLaMA2-7B [19]. The model's parameters for our experiments use precision in FP16 data format, with intermediate activations in FP16.

Key Metrics & Baseline. The simulator will evaluate the cloud-device collaboration system's latency and cost. We use device-side-only inference as a baseline for TTFT and TPOT and cloud-side-only inference as a baseline for inference cost. We observe the performance effects of hardware combination and inference workloads on cloud-device collaboration.

B. TTFT & TPOT Evaluation

Fig. 6 shows the inference latency by applying the method to deploy different operators of LLMs on the cloud-side and device-side. This experiment evaluates the TTFT and TPOT of different inference tasks. The results show that under the different hardware combinations and prefill/decode size, the TTFT and TPOT meet the needs of inference services (less than 400ms and 100ms).

C. Inference Cost Evaluation

Fig. 7 shows the cost reduction of the prefill and decode stages, achieved by applying our method to deploy different computing workloads for LLMs on the cloud-side and device-side. This experiment evaluated the cost of the prefill and decode stage for LLaMA2-7B. The results show that the computational costs of the prefill and decode stage for different inference tasks are reduced under different hardware combinations. The average end-to-end inference cost for both hardware combinations is calculated to be 11.6% and 20.7% lower than cloud-side inference, respectively.

V. Conclusions

In this paper, we propose a cloud-device collaborative inference framework for LLMs inference. We summarize the computation and memory access characteristics of operators and the characteristics of cloud-device hardware. Additionally, we verify the effect of the collaborative inference strategy under different cloud-device hardware combinations and different inference tasks using our compilation workflow. In the future, with the continuous improvement of computing power and bandwidth of devices, it is expected to achieve low latency and high throughput LLMs inference services with the help of the cloud-device collaborative inference framework.

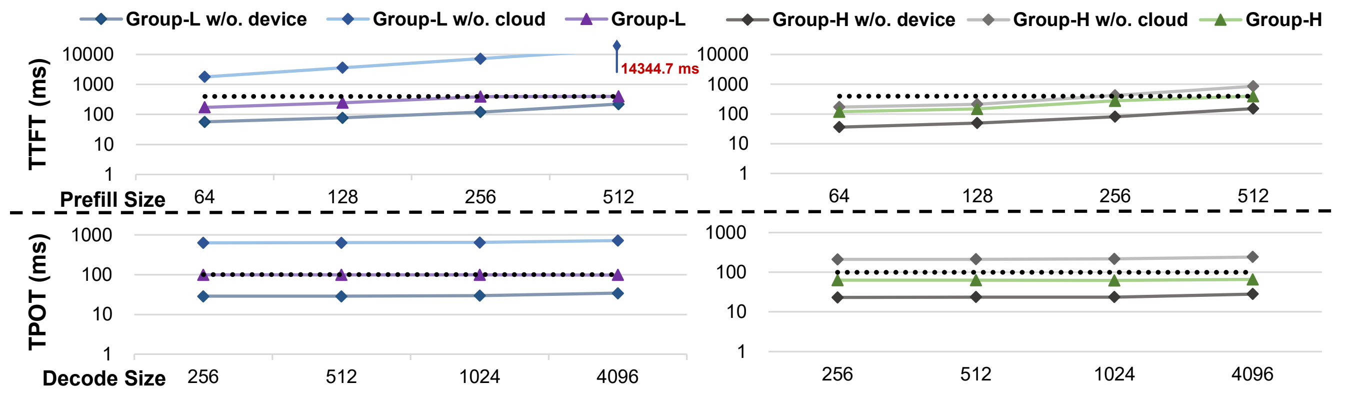

Fig. 6. TTFT and TPOT of Group-L and Group-H. The horizontal axis represents the prefill size or decode size. The vertical axis represents the latency.

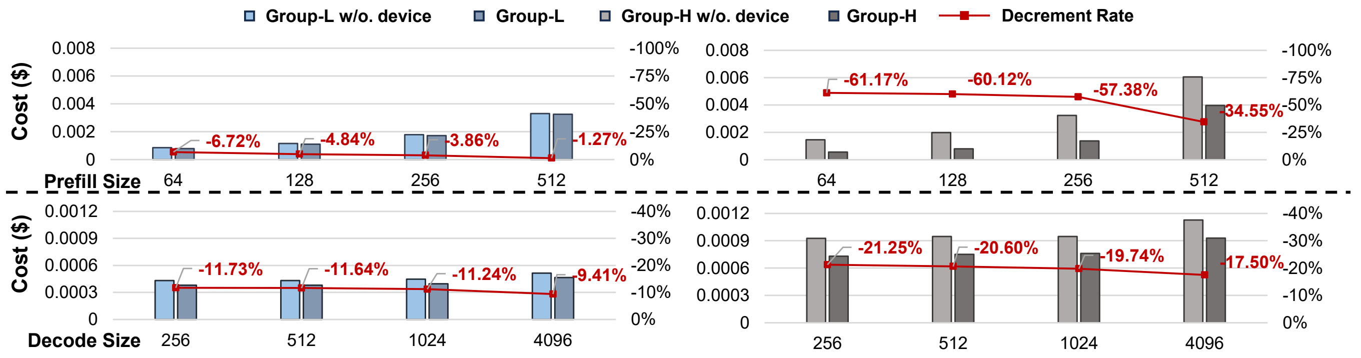

Fig. 7. Prefill stage and decode stage cost of Group-L and Group-H. The horizontal axis represents the prefill size or decode size. The right vertical axis represents the decrement rate of the cost.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (No. 62325405, 62104128, U21B2031, 62204164), the National Key R&D Program of China (2023YFB4502200), Tsinghua EE Xilinx AI Research Fund, Tsinghua-Meituan Joint Institute for Digital Life, and Beijing National Research Center for Information Science and Technology (BNRist).

References

[1] C. Wang, J. Hu, C. Gao, Y. Jin, T. Xie, H. Huang, Z. Lei, and Y. Deng, "Practitioners' expectations on code completion," *arXiv preprint arXiv:2301.03846*, 2023.

[2] S. Chen, M. Wu, K. Q. Zhu, K. Lan, Z. Zhang, and L. Cui, "Llm-empowered chatbots for psychiatrist and patient simulation: application and evaluation," *arXiv preprint arXiv:2305.13614*, 2023.

[3] C. Jeong, "A study on the implementation of generative ai services using an enterprise data-based llm application architecture," *arXiv preprint arXiv:2309.01105*, 2023.

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[5] J. Yao, S. Zhang, Y. Yao, F. Wang, J. Ma, J. Zhang, Y. Chu, L. Ji, K. Jia, T. Shen *et al.*, "Edge-cloud polarization and collaboration: A comprehensive survey for ai," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 7, pp. 6866–6886, 2022.

[6] 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.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[8] M. Agarwal, A. Qureshi, L. L. N. Sardana, J. Quevedo, and D. Khudia, "Llm inference performance engineering: Best practices," 2023.

[9] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "Spinn: synergistic progressive inference of neural networks over device and cloud," in *Proceedings of the 26th annual international conference on mobile computing and networking*, 2020, pp. 1–15.

[10] Y. Hu, Z. Li, Y. Chen, Y. Cheng, Z. Cao, and J. Liu, "Content-aware adaptive device-cloud collaborative inference for object detection," *IEEE Internet of Things Journal*, 2023.

[11] J. Yao, F. Wang, K. Jia, B. Han, J. Zhou, and H. Yang, "Device-cloud collaborative learning for recommendation," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 3865–3874.

[12] C. Ding, A. Zhou, Y. Liu, R. N. Chang, C.-H. Hsu, and S. Wang, "A cloud-edge collaboration framework for cognitive service," *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 1489–1499, 2020.

[13] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, 2020.

[14] Y. Gong, Z. Jiang, Y. Feng, B. Hu, K. Zhao, Q. Liu, and W. Ou, "Edgerec: recommender system on edge in mobile taobao," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 2477–2484.

[15] S. Ye, J. Du, L. Zeng, W. Ou, X. Chu, Y. Lu, and X. Chen, "Galaxy: A resource-efficient collaborative edge ai system for in-situ transformer inference," *arXiv preprint arXiv:2405.17245*, 2024.

[16] Z. Yang, Y. Yang, C. Zhao, Q. Guo, W. He, and W. Ji, "Perllm: Personalized inference scheduling with edge-cloud collaboration for diverse llm services," *arXiv preprint arXiv:2405.14636*, 2024.

[17] M. Zhang, J. Cao, X. Shen, and Z. Cui, "Edgeshard: Efficient llm inference via collaborative edge computing," *arXiv preprint arXiv:2405.14371*, 2024.

[18] J. G. Cornelis and J. Lemeire, "The pipeline performance model: a generic executable performance model for gpus," in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2019, pp. 260–265.

[19] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.