# HPipe: Large Language Model Pipeline Parallelism for Long Context on Heterogeneous Cost-effective Devices

**Ruilong Ma***, **Xiang Yang***, **Jingyu Wang**, **Qi Qi**, **Haifeng Sun**†, **Jing Wang**†, **Zirui Zhuang**, **Jianxin Liao**
State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications
{maruilong,yangxiang,wangjingyu,qiqi8266,hfsun,wangjing
zhuangzirui,liaojx}@bupt.edu.cn

## Abstract

Micro-enterprises and individual developers emerge long context analysis demands with powerful Large Language Models (LLMs). They try to deploy the LLMs at local, but only possess various commodity devices and the unreliable interconnection between devices. Existing parallel techniques can not fully perform in limited environment. The heterogeneity of devices, coupled with their limited capacity and expensive communication, brings challenges to private deployment for maximized utilization of available devices while masking latency. Hence, we introduce HPipe, a pipeline inference framework that successfully mitigates LLMs from high-performance clusters to heterogeneous commodity devices. By ensuring a balanced distribution of workloads, HPipe facilitates the inference through pipelining the sequences on the token dimension. The evaluation conducted on `LLaMA-7B` and `GPT3-2B` demonstrates that HPipe holds the potential for long context analysis on LLM with heterogeneity devices, achieving an impressive speedup in latency and throughput up to 2.28 times.

## 1 Introduction

The emergence of LLMs has significantly enhanced automated content comprehension, as they adeptly capture semantic information within extensive contexts. Enterprises employ techniques such as sentiment analysis (Zhang et al., 2023; Deng et al., 2023; Wang et al., 2023) and content analysis (Gubelmann et al., 2023) to harness the potential value to facilitate the anticipation of user engagement and strategic decision-making. However, due to the stringent memory and computational requirements of LLMs, they are commonly deployed on high-performance computing clusters. The advanced devices and high-velocity transmission like NVlink, boasting transfer rates approaching 900 GB/s, enable rapid computation and efficient synchronization. While micro-enterprises introduce demands to leverage the private LLM, they only have inconsistent weaker devices. The interconnection among these devices also suffers from limited bandwidth. Devices connected via wireless network exhibits transfer rate merely up to 1 GB/s. Thus, the customized LLM deployment schema for micro-enterprises deserves further exploration.

For the demands of effective inference, inference engines (Aminabadi et al., 2022b; Li et al., 2023) provides hybrid data and pipeline parallelism (Huang et al., 2019; Narayanan et al., 2021) and combined with tensor parallelism (Shoeybi et al., 2019; Jia et al., 2019). In high-performance computing centers, they substantially alleviate computational and memory pressure, thereby augmenting inference speed and enhancing throughput.

However, existing methods cannot be directly applicable to the scenarios of micro-enterprises. The deployment for the micro-enterprises presents several problems. **1) Extended text:** As LLM support longer inputs, the expanded context window brings higher arithmetic pressure. The micro-batch pipeline struggles to maintain efficiency. Each stage of the pipeline demands longer processing durations, and the coarser granularity diminishes the parallelism. **2) Communication discrepancy:** The conditions for communication between devices are discrepant. GPUs within a device generally exchange data via PCIe, and GPUs between devices rely on the network. This impedes the efficacy of communication-intensive methods such as tensor parallelism. **3) Heterogeneous devices:** It is essential that integrating heterogeneous devices to employ all available resources for micro-enterprises. The dual heterogeneity of both computation and transmission, coupled with expensive communication, bring challenges to orchestrating the available devices of micro-enterprises for LLMs deployment.

To address these challenges, we propose *HPipe*,

*Equal Contribution.
†Corresponding Author.



*Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pages 1–9

a pipeline inference framework dedicated to content comprehension for private LLMs. It deploys the LLMs on heterogeneous devices with pipeline parallelism on the token dimension. HPipe shields the heterogeneity of devices by distributing LLMs based on computing capabilities and transmission conditions. For extended context, HPipe slices them into segments by a dynamic programming algorithm and pipelines the computation of segments to amplify the degree of parallelism. HPipe successfully mitigates LLMs from high performance clusters to heterogeneous devices, achieving up to a 2.28× increase in both latency and throughput, alongside a 68.2% reduction in energy consumption compared to other methods.

## 2 Background and Motivation

### 2.1 Parallelism

Pipeline and tensor parallelism are two popular methods for accelerating the inference of LLMs as shown in Fig. 1. Matrix multiplication (MatMul) contributes to most of the overall computation amount. Solving a MatMul can be converted into the solving sum of several smaller MatMul. Tensor parallelism leverages this by dividing and distributing the weight matrix to multiple devices to enable the computation in parallel. Once the computation completes, devices will communicate to synchronize the results. Thus, tensor parallelism is commonly used when the transmission is guaranteed. The pipeline mechanism distributes LLMs across multiple devices, with each device dedicated to a stage of computation. The request is usually segmented into micro-batches and processed sequentially. Transmission is only required for intermediate result. While pipeline is communication lightweight, pipeline in batch dimension still bring challenge when LLMs are serving for micro-enterprises. Memory constraints limit the batch size of requests, which reduces space of dividing data and hinders the degree of parallelism. Moreover, as sequence length increases, each pipeline stage spends more time. The increasing execution time of stages introduces more idle waiting.

### 2.2 Utilization of Devices

As the emerging demands of analysis long sequence, the context window of LLMs continues to expand, occasionally surpassing 8000 tokens. Processing lengthy sequences at once can overburden the devices. Conversely, working with short

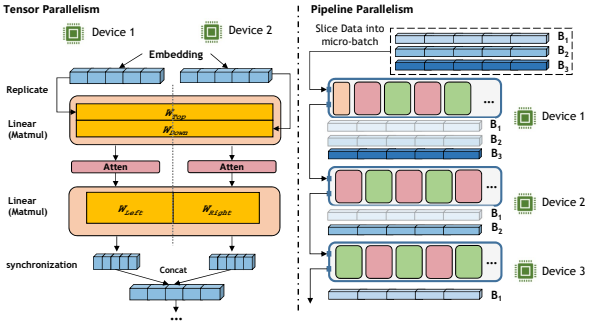



Figure 1: Two popular parallelism approaches: tensor parallelism (left) and pipeline parallelism (right).

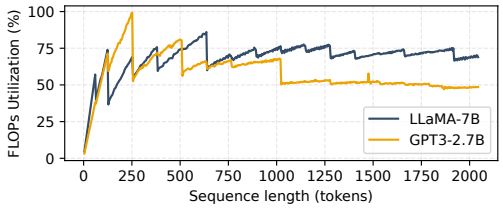



Figure 2: The FLOPs utilization for a transformer block with different sequence lengths on RTX3090 GPU.

sequences is prone to underutilizing the computational power. To explore the relationship between sequence length and resource utilization, we introduce FLOPs utilization, which refers to the ratio of actual floating-point operations per second (FLOPs) achieved to the maximum FLOPs supported by the hardware. Fig. 2 shows the results.As the sequence length expands, FLOPs utilization initially improves and undergoes a decrease before converging. At first, FLOPs utilization increases as more tokens are fed, leading to full utilization of resources. The gains are ultimately constrained by frequent I/O operations. The low-bandwidth memory access causes the bottleneck as the longer embedding involves. We also find fluctuations when the length increase. GPUs conduct MatMul by dividing matrices into tiles to parallel them on distinct thread blocks, which refers to a group of threads computing the same arithmetic operations. Therefore, MatMul achieves maximum GPU utilization when the matrix dimensions are divisible by the tile size. Otherwise, due to tile quantization (Nvidia), some thread blocks perform wasted computation. Therefore, selecting the appropriate length for every process can increase device utilization.

### 2.3 Motivation

On the basis of the discussion above, pipeline parallelism is advantageous for LLMs inference in constrained environments. It allows the reduction

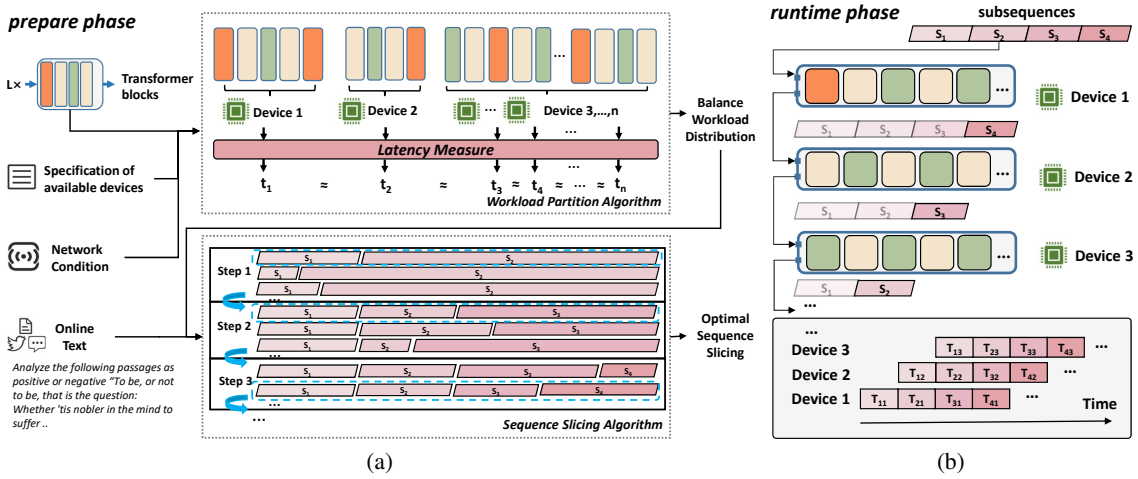



Figure 3: HPipe workflow consists of two phases. In the prepare phase, HPipe determines the optimal schema of workload distribution and the sequence slicing through dynamic programming. In the run-time phase, HPipe pipelines the inference on the token dimension as scheduled.

of massive computational loads and only incurs tolerant communication. Meanwhile, decoder-based transformers inherently facilitate pipeline inference. It enables pipeline on the token dimension for long context, which does not affect the results as the subsequences are fed in sequentially. The K,V values of each subsequence are cached for the calculations of subsequent tokens. Segmenting lengthy sentences into multiple fragments for fine granularity execution maximizes resource utilization. We leverage these observations and design HPipe.

## 3 Method

### 3.1 Workflow

Fig. 3 shows the HPipe workflow. Taking into account the specifications of the devices and network conditions, LLM is properly distributed across multiple devices to maximize the utilization of each device and avoid heavy transmission overhead. HPipe preprocesses the optimal slicing schemes for inputs of all supporting lengths. Once a sequence $S$ arrives, it is divided into subsequences $s_0, ..., s_m$ and executed sequentially across devices. Device $d_i$ can handle the computation task for $s_i$ involving $s_{i+1}$ and $s_{i-1}$ is processing on $d_{i-1}$ and $d_{i+1}$. This effectively reconstructs the pipeline, allowing for parallel on the token dimension.

### 3.2 Formulation

Assuming that the LLM is composed of $n$ layers $\{l_1, ...l_n\}$, they are divided into $N$ blocks $\{b_1, ..., b_N\}$ and distributed across $N$ devices. Meanwhile, the input sequence will be segmented into $M$ subsequences in the token dimension. We use $t_{ij}$ to denote the execution time of each stage in the pipeline, which is the computation time of each subsequence $s_i$ in device $d_j$ plus the transmission time to the successor $d_{j+1}$. The computation of the embedding for subsequences consists of two steps: computing the initial embedding for tokens and combining information from the previous tokens with the relevance scores. The transmission time is related to the size of the intermediate activation derived by the last layer $l_j$ and the bandwidth $B$. The execution time $t_{ij}$ can be presented as :

$$t_{ij} = t_c\left(s_i, \sum_{m=1}^{i-1} s_m; d_j\right) + t_t(l_j, s_i, B). \quad (1)$$

We use $t_c$ to denote the whole computation latency for given $s_i$ and the previous subsequences $s_1, ..., s_i - 1$, and $t_t$ to denote the transmission time.

Our goal is finding a balanced workload partition $\{b_1, ..., b_N\}$ and the proper slicing scheme $\{s_0, ..., s_M\}$ that achieves optimal latency $\mathcal{T}_O^*$ to close the ideal state as shown in Fig. 3. To improve the efficiency of pipeline, it is essential to equalize the stage execution times. We establish a constraint to progressively approach the optimal schema:

$$\mathcal{T}^* \leq \max_{i \in N}\left\{\sum_{j=0}^{M} t_{ij}\right\} + (N-1)\max_{\substack{0 \leq i < M, \\ 0 \leq j < N}}\{t_{ij}\}. \quad (2)$$

The first term is the complete inference latency on the slowest device; The second term is the overhead brought by the pipeline execution, which is

determined by the slowest stage. The constraint allows us to determine the optimal solution by restricting the upper limit of latency. It is obvious that the slowest device and device $t_{ij}$ dominates the total latency. Hence, eliminating the gap between devices and stages will facilitate the pipeline inference. We equalize the pipeline inference by distribution balance and sequence schedule.

### 3.3 Distribution Balance

A balanced model partition minimizes the impact of heterogeneity present in both devices and transmission conditions. We first optimize the pipeline by distributing the LLMs to align with capabilities of devices while considering transmission overhead. We take layer as the partition granularity instead of transformer block, which provides the opportunity to explore more balanced partition.

The objective of balance distribution is to find the the $N-1$ cut points to partition a LLM into $N$ subsets. Each has consecutive layers and is assigned to a specific device. In the heterogeneous environment, this can be established as a device placement problem and has been proven as NP-hard in (Benoit and Robert, 2008). To address this challenge, we make the assumption that the sequence of devices remains constant, that is, the block $b_j$ corresponds to the device $d_j$. Since the LLM is composed of repeating blocks, the constant sequence of devices barely loses the optimal solution, and the problem can be simplified.

The execution time for processing the layers from $l_{a+1}$ to $l_b$ on device $d_m$ encompasses two components: the cumulative computation time of the layers and the communication time to transfer the intermediate activation. It can be obtained by:

$$T(a,b,m) = \sum_{k=a}^{b} t_{comp}(l_k; d_m) + t_{comm}(l_j, m). \quad (3)$$

For the optimal partition, it can be broken into an optimal sub-pipeline consisting of layers from $l_1$ through $l_k$ with $m-1$ devices followed by a single stage with layers $l_{k+1}$ to $l_b$ on device $d_m$. Using the optimal sub-problem property, we can determine a placement scheme that strives to equalize the execution time among devices in stepwise manner:

$$\mathbb{A}[b][m] = \min_{1 \le k < j} \{\max\{\mathbb{A}[k][m-1], T(k+1, b, m)\}\}, \quad (4)$$

where $\mathbb{A}[b][m-1]$ is the time taken by the slowest stage of the optimal sub-pipeline from $l_1$ to $l_b$ with former $m-1$ edge devices. Algorithm 1 in Appendix A.1 shows the pseudocode of how we use dynamic programming to obtain balanced partition.

### 3.4 Sequence Schedule

With the balanced workload distribution, the execution time of the sequence on the devices is similar. Thus, pipeline efficiency now is determined by the most expensive subsequence. We further improve the pipeline by optimally slicing the sequence.

Some studies (Zheng et al., 2023; Li et al., 2021) observed that executing time of token is linearly increase as the location index grows since more previous tokens involves in computation. Therefore, an ideal slicing should include longer slices at the beginning and shorter slices toward the end. Furthermore, the granularity of dividing the sequence also is of significance, as discussed in Section 2.2. Employing a finer-grained slicing approach, characterized by smaller values of $|s_i|$ results in the underutilization of the computational power of GPUs. In contrast, adopting a coarser slicing approach, involving higher values of $|s_i|$, reduces the number of pipeline stages, which decreases the degree of parallelism and may overburden the devices. Thus, it is necessary to find the most suitable slicing granularity to fully leverage devices.

The $t_m = \max\{t_{ij}\}$ is the key to minimize the overall latency. We enumerate possible $t_m$ to find the optimal slicing $S^*$ from slicing space $\mathbb{S}$:

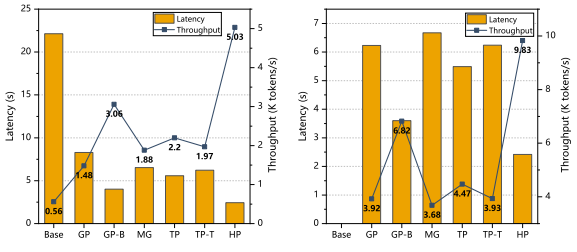
$$\mathcal{T}^* \le \min_{t_m}\{\max_{i \in N}\{\min_{S^* \in \mathbb{S}}\{\sum_{j=0}^{M} t_{ij} | t_{ij} \le t_m\}\} + (N-1)t_m\}. \quad (5)$$

$t_m$ restricts each slice to have the similar execution time, which lead to minimum pipeline latency. Since the optimization of sequence $S$ can derive from $S - s_n$, we employ a dynamic programming algorithm to produce an optimal slicing schema in all possible $t_m$. The details are provided in Appendix A.2 Algorithm 2.

## 4 Evaluation

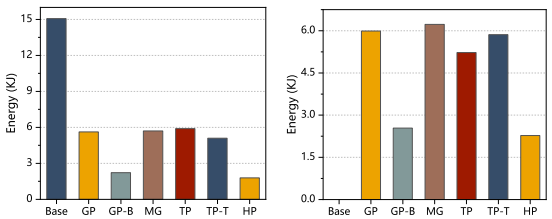
### 4.1 Experimental Setup

We established the HPipe prototype with a computational cluster of two host machines. The first machine contains four Pascal100 (P100), while the second is fitted with two RTX3090 . Communication between hosts is via a wired network with a bandwidth of 1000 Mbps, and intra-host communication is via PCIe. We use this heterogeneous




(a) (b)

Figure 4: The latency and throughput of different approach on the LLaMA-7B (left) and GPT3-2B (right).




(a) (b)

Figure 5: The Energy consumption of cluster during inference on the LLaMA-7B (left) and GPT3-2B. (right)

cluster to mimic a commodity hardware setup. We evaluate HPipe on GPT3-2B, LLaMA-7B. The length of the input sequence is set as 2048 tokens to simulate content analysis for long sequence. The batch size of GPT3-2B and LLaMA-7B are set as 12 and 6.

## 4.2 Performance

We compare HPipe (HP) with the following method (1) Base: LLM is uniformly distributed across each GPU, and inference is performed sequentially across the cluster. (2) GPipe (GP) (Huang et al., 2019): Evenly distribute the LLM across GPU and pipeline the inference with micro-batch (3) GP-B: GPipe with the workload distribution proposed by HPipe. (4) Megatron-LM (MG) (Shoeybi et al., 2019): combine tensor parallelism with GPipe (5) Terapipe (Li et al., 2021): Evenly distribute the LLM across GPU and pipeline the inference on the token dimension. (6) TP-T: Combine tensor parallelism with TeraPipe.

### 4.2.1 Latency and Throughput

Fig. 4 presents the latency and throughput of different methods. Harnessing multiple devices for parallelism allows efficient LLM inference. On LLaMA-7B, HP markedly reduces latency to 2.24s, achieving a speedup of 9.06× compared to Base. It also increases the throughput from 0.56k to 5.03k tokens/s, greatly improving the efficiency. GP pipelines inference in micro-batch. The coarse granularity of parallel remains room for optimization. MG introduces tensor parallelism to share the computation but is limited to the transmission cost. While small volumes of synchronized data enable acceleration through tensor parallelism, larger volumes suffer from significant transmission overhead, thereby impeding performance. With a balanced workload distribution, GP-B and HP demonstrate the latency reduction of 51~56% and the throughput enhancement of 2.06~2.28×. These improvements are attributed to judiciously managing the computing resources of the cluster. What is more, pipelining on the token dimension further expedites the inference, a result of the smaller execution granularity achieved by HPipe. It facilitates higher parallelism degree, minimizes device idle time, and optimizes device utilization during inference, leading to latency reduction by 33.1~39.3%. Comparsion of TP and TP-T shows tensor parallelism is not suitable to combine with pipeline on token dimension. This is because slicing tokens into fine-granularity segments introduces more frequent synchronization, which causes additional overhead.

### 4.2.2 Energy Consumption

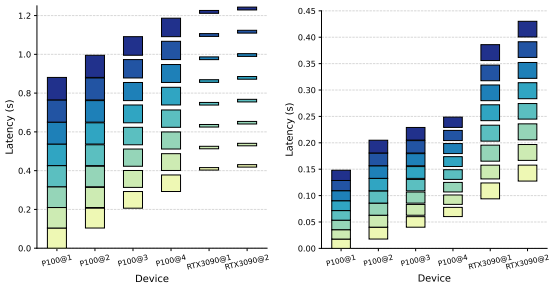
Energy consumption is an important metric of inference performance. Fig. 5 shows the least dynamic energy consumption that HPipe takes. The optimization of GP, MG and TP does not consider the power characteristics of different types of devices so that the workload is processed in an energy-lavish manner. In contrast, by jointly optimizing the trade-off between computation and communication provided devices' computing capabilities and network conditions, HPipe achieves the lowest energy costs. It comes that HPipe finds the sequence length that approximates the maximum utilization of cluster execution through a two-step optimization. The inference is executed under high resource utilization, thus reflecting less energy consumption.

### 4.2.3 Memory Footprint

We record the memory footprint of devices as shown in Table 1. Tensor Parallelism can reduce the memory pressure by distributing the weight. MeanWhile, with balanced workload distribution, LLMs are apportioned among machines according to their computing capabilities, thereby mitigating the memory burden per machine as the increased devices. We also find that the memory of P@4 and R@1 is relatively lower compared to peer de-

Table 1: **Memory footprint of different methods during inference on devices. OOM means device is out of memory during the runtime. P denotes P100 and R denotes RTX3090**

| Model | Methods | Memory footprints (MB) | | | | | |
|---|---|---|---|---|---|---|---|
| | | P@1 | P@2 | P@3 | P@4 | R@1 | R@2 |
| LLaMA-7B | Base | 11479 | 11479 | 11019 | 11019 | 11461 | 11461 |
| | GP | 7031 | 7031 | 6593 | 6593 | 5509 | 5509 |
| | GP-B | 2897 | 3135 | 3655 | 3031 | 9691 | 10739 |
| | MG | 5851 | 5851 | 5493 | 5493 | 5943 | 5943 |
| | TP | 5459 | 5459 | 4505 | 4505 | 4957 | 4957 |
| | TP-P | 4869 | 4869 | 4583 | 4583 | 5013 | 5013 |
| | HP | 1873 | 2977 | 3143 | 1991 | 8713 | 10087 |
| GPT3-2B | Base | OOM | OOM | OOM | OOM | - | - |
| | GP | 7031 | 7031 | 6593 | 6593 | 5509 | 5509 |
| | GP-B | 3665 | 3505 | 3495 | 3177 | 8525 | 8627 |
| | MG | 4695 | 4695 | 4595 | 4595 | 5057 | 5043 |
| | TP | 6601 | 6601 | 6629 | 6629 | 6681 | 6681 |
| | TP-P | 4952 | 4952 | 5032 | 5032 | 5433 | 5437 |
| | HP | 4693 | 4651 | 3153 | 2953 | 9757 | 9855 |




(a) (b)

Figure 6: The performance of the pipeline inference with or without HPipe. Distinct colored blocks represent the execution time of subsequences. The gaps between blocks are the communication time for transferring intermediate activation.

vices. This disparity is attributed to the inclusion of the heterogeneous communication environment. Devices with higher communication overhead are allocated fewer layers to offset the increased burden of communication, which is reflected in the memory with fewer parameters.

### 4.3 Resource Utilization

To affirm HPipe in leveraging computational resources, we visualize the inferences in Fig. 6, which are measured on `LLaMA-7B` and batch size is set as 1. Fig. 6a shows the result of equal distribution of the LLM, along with the evenly slicing of sequences. RTX3090 exhibits a tiny execution time compared to P100, ascribed to LLM distribution failing to fully harness the device's capabilities. RTX3090 rapidly completes the computation task of each subsequence and falls into a waiting state for the next subsequence. A significant portion of the computational resources remain idle. Moreover, uniform slicing sequences lead to longer execution times for subsequent subsequences, causing a bottleneck in the pipeline efficiency. Fig. 6b demonstrates that HPipe schedules the execution of subsequences. Computationally powerful devices are burdened with heavier computational tasks, which gives an approximate execution time for each subsequence. Meanwhile, increasingly shorter subsequences balance the pipeline.

## 5 Related Work

Parallel acceleration on deep neural networks has been widely studied. Only using the data parallelism (Hou et al., 2022; Zhang et al., 2021; Ma et al., 2023) is not enough as parameters of LLMs expand. Pipeline parallelism (Huang et al., 2019; Aminabadi et al., 2022a; Li et al., 2021) and tensor parallelism (Shoeybi et al., 2019; Bian et al., 2021) distribute the model to multiple GPUs, thus reducing the memory burden of the device and allowing efficient scaling of LLM inference. On the basis of them, lots of work achieve inference speedup. Byte-Transformer (Zhai et al., 2023) proposes a padding-free algorithm that liberates inference from redundant computations on zero padded tokens when faced with variable-length sequences. Kernel fusion (Choi et al., 2022; Dao et al., 2022) optimized CUDA kernels to reduce memory access and improve computation speed. These methods focus on latency-oriented scenarios with advanced devices, limiting their deployment to easily accessible hardware with weaker computing capability and memory storage. In comparison, this paper derives the parallelism schema on a heterogeneous cluster of commodity devices to cater to the private application requirements. In addition, techniques proposed by HPipe are orthogonal to the optimized methods, including quantization (Dettmers et al., 2022) and kernel optimization (Li et al., 2022), hence they can be combined with them for better performance.

## 6 Conclusion

This paper introduces HPipe, an inference framework to accelerate the content analysis with LLMs prototyped on the cluster of commodity devices. It effectively integrates computing resources, allowing a fine-granularity pipeline on heterogeneous devices. HPipe demonstrates the potential to accelerate LLMs inference with long sequence input, offering a solution for LLMs deployment in hetero-

geneous commodity hardware environments.

## 7 Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grants (62201072, 62101064, 62171057, U23B2001, 62001054, 62071067), the Ministry of Education and China Mobile Joint Fund (MCM20200202, MCM20180101)

## References

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and Yuxiong He. 2022a. Deepspeed- inference: Enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 2022b. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE.

Anne Benoit and Yves Robert. 2008. Mapping pipeline skeletons onto heterogeneous platforms. *Journal of Parallel and Distributed Computing*, pages 790–808.

Zhengda Bian, Hongxin Liu, Boxiang Wang, Haichen Huang, Yongbin Li, Chuanrui Wang, Fan Cui, and Yang You. 2021. Colossal-ai: A unified deep learning system for large-scale parallel training. *CoRR*.

Jaewan Choi, Hailong Li, Byeongho Kim, Seunghwan Hwang, and Jung Ho Ahn. 2022. Accelerating transformer networks through recomposing softmax layers. In *2022 IEEE International Symposium on Workload Characterization (IISWC)*, pages 92–103.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, pages 16344–16359.

Xiang Deng, Vasilisa Bashlovkina, Feng Han, Simon Baumgartner, and Michael Bendersky. 2023. Llms to the moon? reddit market sentiment analysis with large language models. In *Companion Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, pages 1014–1019.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*.

Reto Gubelmann, Aikaterini-Lida Kalouli, Christina Niklaus, and Siegfried Handschuh. 2023. When truth matters - addressing pragmatic categories in natural language inference (NLI) by large language models (llms). In *Proceedings of the The 12th Joint Conference on Lexical and Computational Semantics, *SEM@ACL 2023, Toronto, Canada, July 13-14, 2023*.

Xueyu Hou, Yongjie Guan, Tao Han, and Ning Zhang. 2022. Distredge: Speeding up convolutional neural network inference on distributed edge devices. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1097–1107.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32.

Zhihao Jia, Matei Zaharia, and Alex Aiken. 2019. Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems*, 1:1–13.

Gongzheng Li, Yadong Xi, Jingzhen Ding, Duan Wang, Ziyang Luo, Rongsheng Zhang, Bai Liu, Changjie Fan, Xiaoxi Mao, and Zeng Zhao. 2022. Easy and efficient transformer: Scalable inference solution for large NLP model. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track, NAACL 2022, Hybrid: Seattle, Washington, USA + Online, July 10-15, 2022*.

Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E Gonzalez, et al. 2023. {AlpaServe}: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 663–679.

Zhuohan Li, Siyuan Zhuang, Shiyuan Guo, Danyang Zhuo, Hao Zhang, Dawn Song, and Ion Stoica. 2021. Terapipe: Token-level pipeline parallelism for training large-scale language models. In *International Conference on Machine Learning*, pages 6543–6552.

Ruilong Ma, Xiang Yang, Qi Qi, Jingyu Wang, Zirui Zhuang, Jing Wang, and Xin Wang. 2023. Brief announcement: Accelerate cnn inference with zoning graph at dynamic granularity. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 295–298.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. 2021. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the*

*International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15.

Nvidia. Matrix multiplication background user's guide. docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.

Zengzhi Wang, Qiming Xie, Zixiang Ding, Yi Feng, and Rui Xia. 2023. Is chatgpt a good sentiment analyzer? A preliminary study. *CoRR*.

Yujia Zhai, Chengquan Jiang, Leyuan Wang, Xiaoying Jia, Shang Zhang, Zizhong Chen, Xin Liu, and Yibo Zhu. 2023. Bytetransformer: A high-performance transformer boosted for variable-length inputs. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 344–355.

Shuai Zhang, Sheng Zhang, Zhuzhong Qian, Jie Wu, Yibo Jin, and Sanglu Lu. 2021. Deepslicing: collaborative and adaptive cnn inference with low latency. *IEEE Transactions on Parallel and Distributed Systems*, pages 2175–2187.

Wenxuan Zhang, Yue Deng, Bing Liu, Sinno Jialin Pan, and Lidong Bing. 2023. Sentiment analysis in the era of large language models: A reality check. *CoRR*, abs/2305.15005.

Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. 2023. Response length perception and sequence scheduling: An llm-empowered llm inference pipeline. *arXiv preprint arXiv:2305.13144*.

# A Appendix

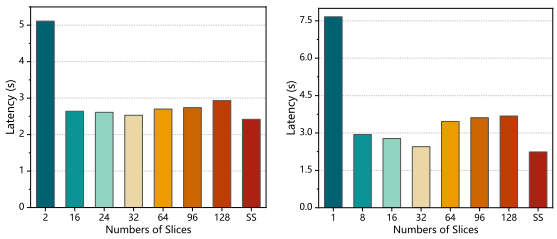
## A.1 Workload distribution Algorithm

Algorithm 1 shows the pseudocode of balance workload distribution to shield the heterogeneity of cluster. **Line 1-2** initializes the execution time of different numbers of layers assigned to the first device. **Line 3-5** outlines the dynamic programming approach for balanced workload distribution. $\mathbb{A}[N][j]$ record the the minimum execution time of the stages that assign the first $N$ layers to the first $j$ layers, which is determined by the lesser assignment of the first $k$ layers of the model to the first $n-1$ devices and the $k+1$ to $m$ layers to the device $n$. The cut-off points are recorded in $p_i$. **Line 6-9** derives the workload distribution schema according to the cut points.

**Algorithm 1 Workload distribution**

**Input:** Computation and communication time per layer of each device.
**Output:** Minimal slowest execution time $\mathbb{A}[N][M]$ and corresponding workload distribution schema.

1: **for** $i$ from 1 to $N$ **do**
2: calculate $\mathbb{A}[i][1]$ using (3)
3: **for** $j$ from 2 to $M$ **do**
4: $\mathbb{A}[N][j] \leftarrow \min_{1\le k\le N}\{\max\{\mathbb{A}[k][j-1], T(k+1, N, j)\}\}$
5: $p_i \leftarrow \arg\min_{1\le k\le N}\{\max\{\mathbb{A}[k][j-1], T(k+1, N, j)\}\}$
▷ *Dynamic programming for the balance workload distribution*
6: $i \leftarrow N, p \leftarrow \{\}$
7: **while** $i > 0$ **do**
8: $p.append(p_i)$
9: $i \leftarrow i - p_i$ ▷ *Derive the workload distribution scheme*




(a) (b)

Figure 7: The latency of the pipeline inference with uniform slice from 1 to 128 in the token dimension and the sequence schedule (SS). (a) `GPT3-2B` (b) `LLaMA-7B`

## A.2 Sequence Slicing Algorithm

Algorithm 2 shows the detail of sequence slicing. **Line 4-13** shows the iteration that finds the optimal slicing with $t_{max}$. Each time we slice a subsequence in the front and treat the remaining sequence as a new sequence until the sequence is divided. The least latency of a sequence with different lengths is stored in $\mathbb{L}[s_{cur}]$ and the length of the just segmented subsequence is stored in $\mathbb{S}[s_{cur}]$. **Line 16-19** derives the optimal sequence slicing based on the record in $\mathbb{S}$. **Line 20-22** gets the optimal slicing scheme among the enumeration of different $t_{max}$.

## A.3 Dynamic Sequence Schedule

We conduct an ablation study on the dynamic sequence schedule (SS) introduced in Section 3.4. To

**Algorithm 2 Sequence slicing**

**Input:** The maximum execution time of slices $t_{max}$, execution time for slices of different lengths $\mathbb{G}$. Arrays to record the latency and trace the sequence slicing $\mathbb{L}, \mathbb{S}$

**Output:** The optimal slicing $\{s_0, ..., s_n\}$

$T \leftarrow$ all possible latency in $\mathbb{G}$
$\mathcal{T}^* \leftarrow \infty, S^* \leftarrow None$
**for** $t_{max}$ in $T$ **do**
  **for** $s_{cur}$ from 1 to $N$ **do**
    $\mathbb{L}[s_{cur}] \leftarrow \infty$
    **for** $s_{step}$ from 1 to $s_{cur}$ **do**
      $l_{step} \leftarrow \mathbb{G}[s_{cur}][s_{cur} - s_{step}]$
      $l_{total} \leftarrow \mathbb{L}[s_{cur} - s_{step}] + l_{step}$
      **if** $s_{cur} \leq t_{max}$ && $l_{total} < \mathbb{L}[s_{cur}]$ **then**
        $\mathbb{L}[s_{cur}] \leftarrow l_{total}$
        $\mathbb{S}[s_{cur}] \leftarrow s_{step}$
  ▷ *Dynamic programming for the optimal slicing under the* $t_{max}$
  $i \leftarrow |S|, S \leftarrow \{\}$
  **while** $i > 0$ **do**
    $S.append(\mathbb{S}[i])$
    $i \leftarrow i - \mathbb{S}[i]$ ▷ *Derive the sequence slicing*
  $\mathcal{T} = (M-1) * t_{max} + \mathbb{L}[N]$
  **if** $\mathcal{T} < \mathcal{T}^*$ **then**
    $\mathcal{T}^* \leftarrow \mathcal{T}, S^* \leftarrow S$ ▷ *Select the optimal schema* $S^*$

contrast the inference latency of the slicing scheme determined by the sequence schedule with that of a heuristic that slices the input sequence uniformly, we tested both the `GPT3-2B` and `LLaMA-7B` models using a sequence length of 2048 tokens. The batch sizes were set at 12 and 6, respectively. In the uniform slicing approach, the entire input was sliced on the token dimension, with the number of slices ranging from 1 to 128. We measured the inference latency for each slicing configuration. The findings are illustrated in Fig. 7 and align with our hypotheses. Pipelines with fine granularity suffer from GPU underutilization, whereas those with coarser granularity present large pipeline bubbles, culminating in increased inference latency. Moreover, due to the mask mechanism of the decoder-based transformer, the uniform slice hides the discrepancy in computational volume between front and rear subsequences. HPipe with a proper sequence schedule outperforms the best uniform slicing configuration.