DOPD: A Dynamic PD-Disaggregation Architecture for Maximizing Goodput in LLM Inference Serving

Junhan Liao, Minxian Xu, *Senior Member, IEEE*, Wanyi Zheng, Yan Wang, *Member, IEEE*, Kejiang Ye, *Senior Member, IEEE*, Rajkumar Buyya, *Fellow, IEEE*, Chengzhong Xu, *Fellow, IEEE*

Abstract—To meet strict Service-Level Objectives (SLO), contemporary Large Language Models (LLMs) decouple the prefill and decoding stages and place them on separate GPUs to mitigate the distinct bottlenecks inherent to each stage. However, the heterogeneity of LLM workloads causes producer-consumer imbalance between the two instance types in such disaggregated architecture. To address this problem, we propose DOPD (Dynamic Optimal Prefill/Decoding), a dynamic LLM inference system that adjusts instance allocations to achieve an optimal prefill-to-decoding (P/D) ratio based on real-time load monitoring. Combined with an appropriate request-scheduling policy, DOPD effectively resolves imbalances between prefill and decoding instances and mitigates resource allocation mismatches due to mixed-length requests under high concurrency. Experimental evaluations show that, compared with vLLM and DistServe (representative aggregation-based and disaggregation-based approaches), DOPD improves overall system goodput by up to $1.5\times$, decreases P90 time-to-first-token (TTFT) by up to 67.5%, and decreases P90 time-per-output-token (TPOT) by up to 22.8%. Furthermore, our dynamic P/D adjustment technique performs proactive reconfiguration based on historical load, achieving over 99% SLO attainment while using fewer additional resources.

Index Terms—Large Language Models, LLM inference serving, dynamic, efficiency, resource management.

I. INTRODUCTION

SINCE Transformer-based [1] LLMs have demonstrated powerful capabilities in natural language processing [2], an increasing number of researchers have pursued deeper investigations [3], [4] to make LLMs such as GPT-5 [5], LLaMa-4 [6] more capable. Concurrently, the deployment of LLMs has become increasingly pervasive across a wide range of industries and commercial applications such as advanced Bing search engine [7], Google's AI assistant [8], AI code editor

J. Liao, M. Xu and K. Ye are with Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China, and University of Chinese Academy of Sciences, Beijing, China.

W. Zheng is with Southern University of Science and Technology, and is also a joint-training student at the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China.

Yan Wang is with the College of Computer Science, Inner Mongolia University, Inner Mongolia, China.

R. Buyya is with the Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, School of Computing and Information Systems, the University of Melbourne, Melbourne, Australia.

C. Xu is with State Key Lab of IOTSC, University of Macau, Macau, China.
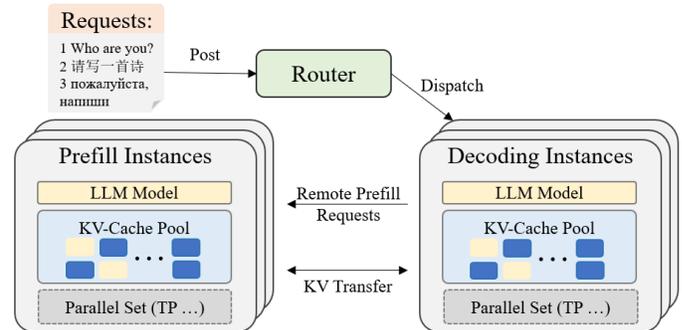
M. Xu is the corresponding author.

Figure 1: Prefill-decoding disaggregation diagram.

[9], log-based fault diagnosis [10]. However, more powerful models often possess extremely large parameter counts. For example, Kimi-k2 [11] contains over 1 trillion parameters. As model parameter counts grow, the computational and storage costs of inference increase substantially. Consequently, LLMs often run on expensive, energy-intensive GPUs [12].

Optimizing LLM inference to reduce these costs has therefore become a major research focus [13]–[15]. Both academic and industrial efforts have proposed several techniques [16]. For example, Key-Value (KV) caching stores historical KV tensors to avoid redundant recomputation. FlashAttention [17] reduces GPU memory-access overhead when computing attention. Mixture-of-Experts (MoE) [18] architectures substantially expand model capacity and improve performance without proportionally increasing computation.

Based on the unique computational characteristics of LLM inference, the process is typically divided into two stages: the prefill stage (compute-intensive) and the decoding stage (memory-intensive). To address the resource contention between these two stages, state-of-the-art systems adopt the PD-Disaggregation architecture (e.g., DistServe [19] and SplitWise [20]). Figure 1 illustrates the overall architecture of such Prefill-Decoding (PD)-Disaggregation systems, which are organized into P-instances (perform only prefill inference) and D-instances (perform only decoding inference). By decoupling the two stages of request inference onto different GPUs for processing, this architecture mitigates the interference between compute-intensive prompt processing and memory-intensive token generation. In this architecture, the P-instance acts as the producer, generating prefill-completed requests after finishing the prefill inference, while the D-instance acts as the consumer that receives these prefill-completed requests and continues with the subsequent decoding inference for each request.

Despite these advantages, current PD-Disaggregation methods face a critical resource optimization challenge: how to minimize resource consumption while satisfying SLO and goodput requirements. This issue is critical for organizations that provide LLM services or otherwise deploy LLMs in production, because solving it enables meeting user demand at substantially lower cost. Under current LLM workloads, a mismatch between the production capacity of P-instances and the consumption capacity of D-instances leads to severe resource inefficiency: over-provisioning results in GPU idleness and resource wastage, while under-provisioning leads to SLO violations and degraded user experience. Therefore, identifying P-instance and D-instance configurations that achieve a producer-consumer balance is the key to minimizing resource usage. Meanwhile, the heterogeneity of LLM workloads causes the optimal configuration to change over time and static deployments remain susceptible to producer-consumer imbalance. Consequently, to accommodate diverse workloads, dynamic and rationalized adjustments are essential. However, excessively frequent reconfiguring of resource allocation incurs substantial time costs, therefore mitigating the short-term mismatch between static PD-Disaggregation deployments and mixed-length request traffic remains a challenging problem.

To address these challenges, we tackle the problem of dynamically adjusting the P/D ratio in a PD-Disaggregation deployment to improve goodput. Concretely, we (i) derive a principled method to compute an optimal P/D ratio for a given load regime, (ii) apply intelligent scheduling to mitigate the short-term resource mismatch caused by mixed-length requests and improve the effectiveness of systems in high-load scenarios, and (iii) leverage the monitoring data to re-evaluate the system P/D ratio and decide whether the current configuration remains optimal or requires elastic scaling of P-instances and D-instances. Our contributions are as follows:

- We design DOPD, an intelligent and dynamic LLM inference framework that continuously tunes the number and configuration of P-instances and D-instances.
- We present a comprehensive system model and in-depth analysis of PD-Disaggregation, identifying the key challenges faced by disaggregated inference architectures.
- We propose an optimal P/D ratio calculation method, and we design an intelligent, length-aware request-scheduling algorithm to mitigate resource mismatch caused by mixed-length inference workloads.
- We perform extensive experiments demonstrating that DOPD substantially increases system goodput ($1.5\times$) and the SLO attainment (from 80.8% to 99.4%) under realistic production traces, while validating its effectiveness across diverse scenarios.

II. MOTIVATION

This section summarizes the issues we observe in existing PD-Disaggregation systems and motivates our design choices.

A. Motivation: Difficulty of Predicting Inference Workloads

Modern LLM inference workloads are highly heterogeneous and nonstationary. Individual user requests exhibit large variability in input and output sequence lengths, which makes
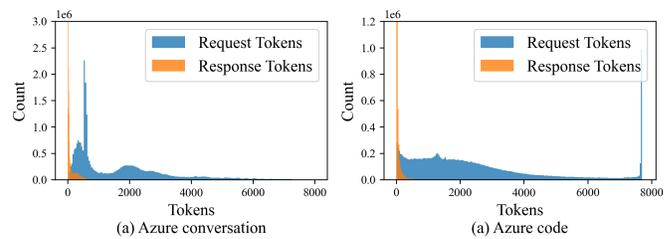

Figure 2: The input/output sequence length of request tokens in Microsoft Azure conversation and code traces.

accurate load prediction challenging. Figure 2 illustrates the input/output sequence length distributions from a production trace (Microsoft Azure conversation and code traces)[1]. These traces were collected from real user interactions with LLM services and exhibit wide length variance, lack clear periodic patterns, and frequently contain sudden bursts. Such characteristics render many traditional prediction methods ineffective, yet accurate short-term forecasts of inference load are essential for prudent resource pre-allocation. For LLM deployments, knowing the near-term inference load in advance enables proactive adjustment of instance allocations, which is critical for preventing resource waste and avoiding overload in PD-Disaggregation systems.

To mitigate this challenge, applying a multiplicative scaling (correction) factor can be effective. By comparing the most recent forecast with observed measurements and adjusting future forecasts accordingly, the scaling factor attenuates systematic bias between predicted and actual values and keeps prediction error within a bounded, small range.

B. Motivation: Bad P/D Ratio Waste Resources

Under the PD-Disaggregation architecture, a mismatch between the system P/D ratio and the current workload leads to severe resource wastage, thereby causing SLO violations and significantly impairing system goodput performance. Figure 3 illustrates the direct impact of different P/D configurations on system performance. Experimental results indicate that different P/D ratios induce significant performance disparities: A PD-Disaggregation system configured with an optimal P/D ratio (the green curve) achieves goodput comparable to a collocated system deployed with 8 GPUs (the blue curve) while utilizing only 6 GPUs. Conversely, a suboptimal P/D ratio (the orange curve), despite utilizing 8 GPUs, fails to leverage the inherent advantages of PD-Disaggregation, resulting in a drastic decline in the utilization of surplus GPUs and consequently yielding poor performance. Evidently, an improper P/D ratio not only significantly deteriorates goodput and GPU utilization but also wastes limited GPU resources, thereby necessitating the system to allocate more resources to satisfy the same SLOs.

To mitigate the producer-consumer imbalance, we model resource allocation as an optimization problem. Given short-term workload statistics, our objective is to minimize the total number of provisioned GPUs while satisfying stringent
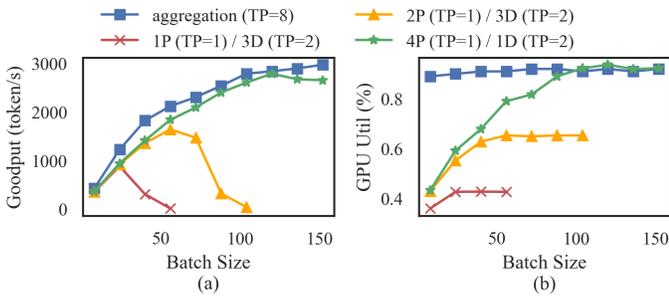
[1]https://github.com/Azure/AzurePublicDataset

Figure 3: The goodput (a) and GPU utilization (b) comparison with different configurations. The "2P (TP=1) / 3D (TP=2)" represents a PD-Disaggregation deployment comprising 2 P-instances each deployed on a single GPU (tensor parallel size = 1) and 3 D-instances each deployed on two GPUs (tensor parallel size = 2). The "aggregation (TP=8)" denotes an aggregation-based system deployed on 8 GPUs (tensor parallel size = 8).

SLO requirements for TTFT, TPOT, and goodput, subject to available hardware constraints. By framing the selection of P-instances and D-instances in this manner, we ensure cost-efficiency without compromising service quality. The specific algorithmic implementation and solver details are further elaborated in Section IV-B.

C. Motivation: Mixed-length Requests Undermine PD-Disaggregation

Prior work [21], [22] and our observations show that when the workload is a mixture of requests with widely varying lengths, PD-Disaggregation can perform substantially worse than deployments in which all requests share a fixed length. The root cause is that different request lengths have different suitable allocations between prefill and decoding resources. Short requests tend to complete decoding quickly while consuming relatively little prefill capacity, whereas long requests require a higher proportion of prefill resources to sustain pipeline efficiency. If requests with very different lengths (e.g., 100 tokens and 1900 tokens) are mixed and a single global P/D ratio is chosen based on the average length (e.g., 1000 tokens), the resulting static allocation typically cannot satisfy the local optima for both short and long requests simultaneously. Therefore, static configurations based on average length of requests therefore produce either resource shortages or long queuing for subsets of requests, degrading overall throughput and tail latency and eroding the benefits of PD-Disaggregation.

At the same time, for extremely short requests whose prefill compute time is negligible, the KV-cache transfer latency between P-instances and D-instances becomes a nontrivial fraction of end-to-end latency. Table I reports, for a single H100 GPU [12] running an LLaMa-3.3-70B-FP8 [23] model, the per-request prefill time and the per-step decoding time (under maximum batching) for several sequence lengths. For example, a prefill for a prompt of length 100 tokens may take approximately 36ms, while a single decoding step at 248 batch size can take approximately 47ms. Even with NIXL's [24] efficient GPU-to-GPU transfers, KV-cache transmission can add on the order of 15-80ms depending on block fragmentation and count, rendering PD-Disaggregation inefficient for such

Table I: For different sequence lengths, we measure the single-run prefill inference latency and the single-step decoding inference latency at different concurrency, where cc refers to request concurrency in the system.

Sequence Length	T_p (ms)	T_d (ms)		
		$cc = 104$	$cc = 200$	$cc = 248$
100	36	28	45	47
200	46	31	46	49
700	125	33	47	53
1200	193	35	50	57
1700	269	37	53	61

requests. In such cases, executing these ultra-short requests in-place (i.e., without remote prefill) typically produces negligible interference with other decoding work, whereas routing them through remote prefill incurs unnecessary KV-cache transfer overhead and increases TTFT. Hence, offloading all ultra-short requests to remote P-instances is often suboptimal.

These two observations motivate the need for length-aware scheduling and workload-aware adaptation. Designing scheduling policies that account for request-length heterogeneity is essential to improve the robustness and efficiency of practical inference systems under mixed workloads.

Motivated by the foregoing, we propose **DOPD**, a PD-Disaggregation framework that (i) produces accurate short-term forecasts of future load, (ii) computes the optimal P/D ratio for the current workload, (iii) leverages workload-aware request scheduling to mitigate the interference introduced by mixed-length requests in PD-Disaggregation under high concurrency, and (iv) dynamically scales the numbers of P-instances and D-instances according to the computed optimal P/D ratio. By reacting to complex and rapidly varying loads, DOPD maximizes goodput using fewer GPU resources, reduces TPOT and end-to-end latency, and maintains excellent SLO attainment.

III. RELATED WORK

This section surveys recent techniques that mitigate interference between prefill and decoding requests and discusses their limitations.

A. Aggregation-based Approaches

Yu et al. propose Orca [25], which uses continuous batching (iteration-level batching and selective batching) applied to generation to mitigate interference from prefill requests within the same batch. Kwon et al. introduce vLLM [26], which separates prefill and decoding requests into different batches and provides optimized operators for the two request types to improve computational efficiency. Both approaches partially alleviate prefill-to-decoding interference, but long-context prefill requests can still delay fast decoding requests.

Agrawal et al. present Sarathi-Serve [27], introducing chunked prefill and continuous batching to split long prefill work into smaller chunks that can be interleaved with decoding work. Building on mixed-batch techniques, Kamath et al. propose POD-Attention [28], a kernel-level redesign of the attention operator that enables true prefill–decoding overlap

Table II: Comparison of related work which is disaggregation-based Approach.

Approaches	Dynamic P/D	Efficient KV-cache Transfer	Optimal P/D
Zhong et al. [19]	✕	✕	✕
Patel et al. [20]	✕	✓	✕
Qin et al. [31]	✕	✓	✕
Jin et al. [32]	✓	✓	✕
Wu et al. [33]	✓	✓	✕
Dynamo [34]	✓	✓	✕
Our Work (DOPD)	✓	✓	✓

on a single GPU by partitioning computation and memory resources at the streaming-multiprocessor level. In practice, POD-Attention further mitigates low decoding throughput and prefill resource domination, yet these methods exhibit stability limitations under high concurrency, long-context workloads, or strict TTFT requirements.

Additionally, some other works propose reducing prefill-stage computation to mitigate the interference caused by co-processing prefill and decoding requests, for example Radix-Attention [29] and Native Sparse Attention [30]. However these approaches are not universally applicable and some may introduce accuracy degradation.

In summary, aggregation-based approaches, such as continuous batching, chunked prefill, kernel-level attention redesign, and sparse attention techniques yield tangible gains but rely on complex, hardware-dependent kernel implementations, introduce additional implementation or memory overheads, and may compromise accuracy in some scenarios. More importantly, they do not fundamentally eliminate the conflict between prefill and decoding requests. By contrast, DOPD employs targeted, length-aware request scheduling to selectively separate or aggregate requests, achieving stronger SLO attainment while preserving high goodput.

B. Disaggregation-based Approaches

Zhong et al. formalize the cost of colocated execution in DistServe [19], demonstrating that colocating prefill and decoding causes strong interference and proposing fully disaggregated deployments to improve goodput. Patel et al. build on PD-Disaggregation to present SplitWise [20], which maximizes computation–communication overlap via a hierarchical KV-cache transfer design and thereby reduces the impact of KV-cache transfer overhead. Similarly, Qin et al. propose Mooncake [31], a KV-cache–centric disaggregated architecture that optimizes prefill-to-decoding KV-cache migration via D2D/RDMA transport and KV-aware scheduling. However, these methods are largely static and do not support real-time dynamic adjustment of instance counts, which can lead to substantial resource waste and transient overloads under variable workloads.

Jin et al. propose PD-Serve [32], which constructs fine-grained P/D grouping by scenario so that requests with similar prefixes are processed within the same group, and which optimizes KV-cache transfer and encoding parallelism while supporting dynamic scaling. Likewise, Wu et al. propose Arrow [33], which designs adaptive scheduling and an elastic instance pool that dynamically adjusts P-instance and D-instance counts based on cluster metrics, improving robustness to bursts and fluctuations. More recently, NVIDIA released the Dynamo framework [34], which advocates prefill–decoding separation to enable tailored tensor parallel size and memory strategies and parallelism. Although these systems introduce dynamic expansion, they still fall short in selecting optimal P/D instance counts and configurations after scaling: achieving post-scale optimal P/D configurations remains difficult, often resulting in significant GPU idleness or overload.

A comparison of representative characteristics of prominent PD-Disaggregation studies is presented in Table II. In summary, the disaggregation-based of approaches decouple prefill and decoding and, on that basis, optimize KV-cache migration and transfer to overlap computation and communication (e.g., SplitWise's hierarchical transfer), and introduce fine-grained grouping and elastic/adaptive scheduling (e.g., PD-Serve's P/D grouping). These measures reduce direct resource contention between the two request types. But many early solutions lack support for real-time, fine-grained elastic scaling, causing resource waste or short-term overloads under fluctuating load. And even systems that support dynamic scaling still face the unresolved challenge of accurately selecting the optimal number and configuration of P/D instances after scaling to avoid GPU idleness or overload, in addition to the engineering cost of complex schedulers and KV-cache migrations.

The method proposed in this paper addresses these gaps by forecasting near-term load and proactively scaling P-instances and D-instances according to an analytically derived optimal P/D ratio, thereby mitigating resource waste and overload caused by suboptimal P/D instance scheduling.

IV. SYSTEM MODEL

This section presents the system model of DOPD, covering the following aspects: 1) Workload characterization. 2) The optimal P/D ratio calculation. 3) Modeling of P-instance. 4) Modeling of D-instance.

A. Workload Characterization

Unlike traditional machine-learning workloads with fixed input dimensionality, LLMs workloads are substantially more heterogeneous. Each user request to an LLM may have a different input sequence length, which in turn yields different per-request computational costs. Because mainstream LLMs are Transformer-based, the computational pressure concentrates in the attention and feed-forward components. The attention computation can be written as

$$Output_{Attn} = \text{softmax}(QK^T)V, \tag{1}$$

where Q, K, V are the query, key, and value matrices produced from the input sequence by linear projections. The asymptotic complexity of the attention operation is

$$C_{attn}(ISL) = \mathcal{O}\big(ISL \cdot H^2 + ISL^2 \cdot H\big), \tag{2}$$

where ISL denotes the input sequence length and H denotes the model hidden size. The feed-forward computation can be abstracted as

$$Output_{FFN} = Linear(Linear(Linear(Output_{Attn}))), \tag{3}$$

whose complexity is approximately

$$C_{ffn}(ISL) = \mathcal{O}(ISL \cdot H \cdot H'), \tag{4}$$

where H' denotes the intermediate size of the feed-forward layer (typically $H' \geq H$). Thus the per-request computational cost can be expressed as

$$C(ISL) = C_{attn}(ISL) + C_{ffn}(ISL). \tag{5}$$

To reason about heterogeneous requests we lift this per-request cost to the level of the input sequence length distribution f_{ISL} and consider moments of ISL. In particular the expected per-request cost is

$$\mathbb{E}[C] = \int C(x) f_{ISL}(x) \, dx$$
$$= \kappa_1 H^2 \mathbb{E}[ISL] + \kappa_2 H \mathbb{E}[ISL^2] + \kappa_3 H H' \mathbb{E}[ISL], \tag{6}$$

where constants $\kappa_1, \kappa_2, \kappa_3$ hide implementation-level factors. Note that

$$\mathbb{E}[ISL^2] = \mathrm{Var}(ISL) + \mathbb{E}[ISL]^2, \tag{7}$$

and because of the ISL^2 term in Eq. (2), the cost function $C(ISL)$ is convex in ISL, so by Jensen's inequality we have

$$\mathbb{E}[C] \geq C(\mathbb{E}[ISL]), \tag{8}$$

which quantifies how heterogeneity (and in particular variance) amplifies average computation beyond what a mean-length estimate would predict.

Empirically, inference workloads commonly exhibit a long-tail distribution: a small fraction of extremely long requests requires the service to retain the capability to handle such tails. A standard heavy-tail model is the Pareto law

$$\Pr\{ISL > x\} = \left(\frac{x_m}{x}\right)^\alpha, \; s.t. \; x \geq x_m, \; \alpha > 0, \tag{9}$$

where the tail index α controls the existence of moments (for example, the variance is finite only if $\alpha > 2$). When α is small the second moment in (Eq. (7)) can be very large (or formally divergent), explaining why a tiny fraction of requests can dominate overall compute and memory pressure.

The high variance in input length makes aggregating prefill and decoding requests problematic. Because the per-step inference time of the two request types can differ dramatically, batching them together causes severe mutual interference and substantially degrades system performance. This effect can be formalized through queueing approximations. If requests arrive with (time-varying) rate $\lambda(t)$ and service time S satisfies $\mathbb{E}[S] \propto \mathbb{E}[C]/COMPUTE_{speed}$, then Little's law $L = \lambda W$ links occupancy L and mean sojourn W, and Kingman's heavy-traffic approximation for a $G/G/1$ queue gives the mean waiting time

$$W_q \approx \frac{\rho}{1-\rho} \cdot \frac{c_a^2 + c_s^2}{2} \mathbb{E}[S], \tag{10}$$

where $\rho = \lambda \mathbb{E}[S]$ is load, c_a^2 is the squared coefficient of variation of inter-arrival times and $c_s^2 = \mathrm{Var}(S)/\mathbb{E}[S]^2$ is that of service time. Because c_s^2 grows with $\mathrm{Var}(ISL)$ via (Eq. (5)) and (Eq. (7)), mixing requests with very different ISL (or mixing prefill and decoding) inflates c_s^2, increasing W_q and thus end-to-end latency.

The complexity of LLM inference service workloads extends further. Requests arrive in short-lived bursts and the instantaneous concurrency seen by the service varies over time. Such burstiness can be modeled by Markov-Modulated Poisson Processes (MMPP) or compound-Poisson models, which capture non-stationary and over-dispersed arrivals (i.e., $c_a^2 > 1$). The peak concurrency in the next service epoch may be orders of magnitude larger (or smaller) than in the previous epoch. Consequently, adaptive resource provisioning that accounts for both the distribution of request lengths (via moments such as $\mathbb{E}[ISL]$, $\mathrm{Var}(ISL)$ and tail index α) and rapidly changing concurrency (via time-varying $\lambda(t)$ or MMPP parameters) is essential for maximizing resource utilization. This non-stationary and bursty nature can cause serious resource waste in PD-Disaggregation systems because a fixed P/D ratio is suitable only for a restricted class of workloads.

To address these challenges, DOPD periodically feeds background-collected telemetry into an ARIMA-based [35] load predictor and issues regular forecasts of the future average request length and concurrency. These forecasts are combined with the moment-based cost estimates in Eq. (6) and the queueing-delay estimates such as Eq. (10) to drive DOPD's dynamic P/D ratio computation and elastic resizing of P-instances and D-instances, enabling the system to pre-allocate resources commensurate with the anticipated heterogeneous and time-varying workload.

B. The Optimal P/D Ratio Calculation

1) The Object of this Optimal P/D Ratio: When all requests are served via remote prefill (perform prefill in P-instance), the prefill-to-decoding workflow naturally forms a producer-consumer pair. P-instances produce prefill-completed requests (together with KV-cache), and D-instances consume those prefill-completed requests to perform token generation. Therefore, balancing production and consumption is essential to reduce resource waste. And system throughput for disaggregated inference is largely determined by the concurrency (the number of in-flight requests in system) that each D-instance can sustain, because each generated token must be produced by a D-instance and returned to the frontend. The maximum concurrency a D-instance can support depends on request length as well as on GPU memory capacity and memory bandwidth. Therefore, the optimal P/D ratio should (i) maximize the aggregate concurrency supported by D-instances and (ii) select a corresponding number of P-instances so that minimize idle time for P-instances and D-instances.

2) Method for Computing the Optimal P/D Ratio: To simplify the following calculations, assume all incoming requests have the same input sequence length ISL and output sequence length OSL. First, based on GPU memory bandwidth BW, GPU memory capacity M_{gpu}, model size M_{model}, and the

CUDA graph inference configuration in use, determine a suitable parallelization configuration (e.g., tensor parallelism). Note that the maximum parallelism of a D-instance must be bounded by both memory capacity and memory bandwidth constraints. From the memory-apacity perspective, for tensor parallel size TP, the remaining memory available for KV-cache on a D-instance is

$$V_{mem} = (M_{gpu} - M^{'}) \times TP - M_{model}, \tag{11}$$

where $M^{'}$ encompasses peak activation-parameter memory (which varies with the model's maximum sequence length) and other ancillary GPU memory usage.

From the perspective of memory bandwidth, as the GPU must continuously retrieve the KV-cache from its own High Bandwidth Memory (HBM) during inference, and given that the computational load for each request during the decoding stage is minimal, the vast majority of execution time is spent accessing model weights and KV-cache. Consequently, the constrained memory bandwidth hinders the retrieval of model weights and KV-cache for all concurrent requests within the TPOT interval specified by the SLO, leading to inevitable SLO violations. To conservatively estimate the maximum concurrency under memory bandwidth limitations, we can neglect the time spent on computation and assume that the entire decoding inference latency for a request under high concurrency is dedicated to reading the KV-cache and model weights. So, we can calculate the maximum volume of KV-cache data that a D-instance can retrieve from HBM within the time interval t_d^{SLO} (the TPOT SLO limit) as followed:

$$V_{BW} = t_d^{SLO} \times \alpha \times TP \times BW, \tag{12}$$

Based on this data volume, we can calculate how many requests' KV-cache can be retrieved from HBM by a single D-instance within t_d^{SLO}. Consequently, by taking the minimum of the KV-cache data volume storable on the D-instance and the total KV-cache data volume retrievable within a single decoding step, and dividing this value by the memory footprint of a single request's KV-cache, we can derive the maximum concurrency cc_d of the D-instance under both memory capacity and memory bandwidth constraints.

$$cc_d = \frac{\min(V_{mem}, V_{BW})}{(ISL + OSL/2) \times M_{token}}, \tag{13}$$

In practice, to improve headroom for bursty traffic one may choose TP that an instance can support up to $0.9 \times cc_g$ concurrency. After obtaining the D-instance's suitable TP and maximum concurrency, we calculate the required production capacity of a single P-instance. Since the prefill stage often becomes the compute bottleneck, some systems set the P-instance batch size as low as 1. For simplicity of calculation, here we assume that each P-instance inference only generates one prefill-completed request. Let t_p denote the time for a P-instance to perform a single prefill operation, and t_d denote the time for a D-instance to perform a single decoding step at its maximum concurrency. Within the time interval required for a D-instance to complete the full decoding sequence of a request (which is $t_d \times OSL$), the P-instance generates a new prefill-completed request every t_p. Each time the D-instance

receives a prefill-completed request, its concurrency increases by one. Therefore, the maximum concurrency that a single P-instance can drive on a D-instance is equal to $\frac{t_d \times OSL}{t_p}$. Since our objective for the optimal P/D ratio is to ensure that every D-instance reaches its maximum concurrency cc_d, the optimal P/D ratio satisfies:

$$n_p \times \frac{t_d \times OSL}{t_p} = n_d \times cc_d, \tag{14}$$

Thus, we derive an analytical formula for the optimal P/D ratio.

Applying this method yields a deployment configuration that, for a given load, selects the resource allocation between P-instances and D-instances so that both roles exhibit near-zero idle time during steady-state inference and the overall system throughput is maximized. The proposed DOPD system employs an ARIMA-based predictor to forecast near-term workload characteristics from historical telemetry, including the future average ISL, the future average OSL, and the expected average concurrency. DOPD maps these forecasts to profiled Prefill and Decode performance metrics, computes the optimal P/D ratio and the corresponding instance configurations, and then enacts non-disruptive instance adjustments. The detailed calculation and execution steps are described in subsection V-D and Algorithm 2.

C. Modeling of P-instance

For P-instances, an excessively large TP (tensor parallel size) can be detrimental. Prefill requests are compute-bound, which imposes high demands on GPU compute capability. Using tensor parallelism allows parallel execution of the same request across multiple GPUs, thereby distributing the computational load and providing acceleration. However, tensor parallelism also introduces two All-Reduce operations per Transformer layer during inference communication-heavy operations that synchronize partial results across GPUs. To model this trade-off briefly, let $C(ISL))$ denote the compute work for a prefill request of input length ISL. We formulate the prefill latency under a given TP as the sum of an ideally accelerated compute term and a communication term:

$$T_p(ISL, TP) = \frac{C(ISL)}{S(TP)} + \tau(TP), \tag{15}$$

where $S(TP)$ is the effective speedup (with $S(TP) \leq TP$ and typically showing diminishing returns) and $\tau(TP)$ models tensor parallelism-induced communication overhead (which is non-decreasing in TP). Eq. (15) explains why, for small batch sizes or small models, increasing TP can increase T_p when $\tau(TP)$ dominates the compute acceleration. When ISL is small, $\tau(TP)$ dominates $T_p(ISL, TP)$, and tensor parallelism may have a counterproductive effect. As ISL increases, $C(ISL)$ grows faster than $\tau(TP)$ and thus comes to dominate $T_p(ISL, TP)$; in this regime increasing $S(TP)$ reduces $T_p(ISL, TP)$ and acceleration begins to manifest. However, on GPUs with lower inter-GPU communication bandwidth, $\tau(TP)$ constitutes a larger fraction of the total latency, the communication-dominated region expands, and

---

Algorithm 1 Prefill Requests Scheduler

1: **Input:**
2: i_{next}: Length threshold for immediate processing
3: *queue*: Remote prefill requests queue
4: *timer*: Record request wait time
5: *batch*: Collection of requests to be batched
6: **Initialize** *timer*, *batch*, *queue*
7: **loop**
8:　　Set waiting timeout for *timer* based on *batch*
9:　　**if** batchEnough(i_{next}, *batch*) **then**　　▷ Check the size of *batch*
10:　　　　Inference and reset *batch*　　▷ Process the requests in *batch*
11:　　　　*timer*.clear()
12:　　**else if** *timer*.isTimeout() **and** $|batch| > 0$ **then**
13:　　　　Inference and reset *batch*
14:　　　　*timer*.clear()
15:　　**end if**
16:　　*req* ← *queue*.pop()
17:　　**if** *req* == *null* **then**
18:　　　　**if** $|batch| > 0$ **then**
19:　　　　　　Inference and reset *batch*
20:　　　　　　*timer*.clear()
21:　　　　**else**
22:　　　　　　**continue**
23:　　　　**end if**
24:　　**end if**
25:　　**if** *req*.length $< i_{next}$ **then**
26:　　　　*batch*.add(*req*)
27:　　　　**if** $|batch| == 1$ **then**
28:　　　　　　*timer*.start()
29:　　　　**end if**
30:　　**else**
31:　　　　Inference *req*
32:　　**end if**
33: **end loop**

---

tensor parallelism on a P-instance becomes less favorable. Conversely, on accelerators with high inter-GPU communication bandwidth, τ accounts for a smaller proportion, but the acceleration effect is still basically sublinear. To illustrate this effect empirically, we validated the behavior on small-model prefill experiments with varying ISL: when using Nvidia H100 GPUs to serve a QWen-1.5B model [36], for $ISL <$ 400 the configuration with $TP = 4$ yields larger TTFT than the configuration with no tensor parallelism. Although the acceleration from tensor parallelism becomes more apparent as ISL increases, the observed speedup (approximately 2.38) still falls well short of a linear TP factor. So, for P-instance, adding new instances is far more efficient than increasing TP. Therefore, it is recommended to set TP only as large as required to support the model's maximum input length.

Under a high-concurrency mixed-length workload, a system configuration obtained from the average input and output lengths may become mismatched as lengths vary. To address this, short requests in the pending workload can be batched up
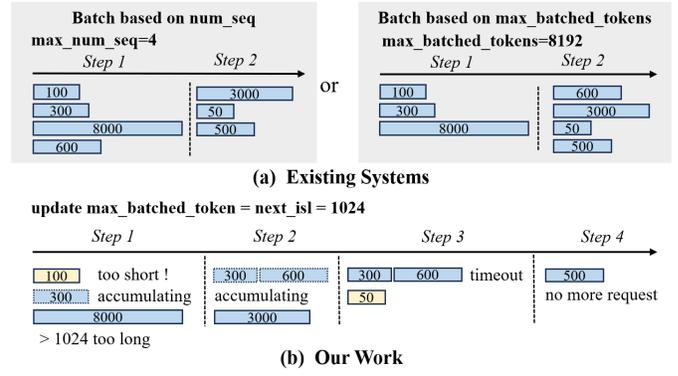

(a) Existing Systems

(b) Our Work

Figure 4: Comparison of prefill request scheduling solutions. All blocks represent the prefill inference process for sequences, and where the blue block represents the prefill inference for that sequence performed in a P-instance, and the yellow block represents the prefill inference performed in a D-instance. The solid block represents processing, and the dashed block represents unprocessed at the current step and still pending processing.

to the input length corresponding to the system configuration so that the system adapts to the current load. To avoid degrading SLO attainment due to such batching, we should first model and analyze request queuing and timeout selection. To reason about queuing delay and timeout selection formally, let incoming requests follow an average arrival rate λ, and denote the mean prefill service time by $\mathbb{E}[S]$ (where $\mathbb{E}[S] = \mathbb{E}[T_p]$ given the chosen TP and batching strategy). Then the system utilization is

$$\rho = \lambda \mathbb{E}[S]. \tag{16}$$

By Little's law the average number of requests in system L and the mean sojourn time W satisfy $L = \lambda W$. For non-exponential arrival/service statistics, the (Kingman) heavy-traffic approximation gives the queueing waiting time W_q same as Eq. (10), where c_a^2 and c_s^2 are the squared coefficients of variation of inter-arrival and service times respectively. Eq. (10) and Eq. (15) allow us to (i) evaluate how a choice of TP affects $\mathbb{E}[S]$ and hence ρ, and (ii) set the maximum waiting timeout $T_{timeout}$ to control the contribution of W_q to TTFT. For example, a conservative choice is to enforce for a safety factor $\alpha \in (0, 1]$ to bound expected extra waiting:

$$T_{timeout} \geq \alpha \cdot W_q. \tag{17}$$

Because LLM request lengths vary widely, the conventional first-come-first-served batching policy, taking the first n queued requests and forming a batch, can be suboptimal (as shown in Figure 4(a)). Such a policy may batch long and short requests together, causing long requests to push the GPU into a compute bottleneck and mismatch of system instances resource configuration, resulting in uneven resource utilization. This can be mitigated by a length-aware internal scheduling policy. When a P-instance dequeues requests from the prefill queue, it inspects each request's input length and applies the following logic.

If a request's length exceeds a predefined threshold, the P-instance dispatches it immediately for inference (as shown in step 1 and step 2 in the flow of Figure 4(b)). In DOPD, we

set this threshold to the ARIMA-based predictor's one-step forecast of the next average input sequence length (denoted $next\_isl$ in Figure 4(b)), so that naturally long requests are not delayed by batching short ones. If a request is short, the instance accumulates short requests until either an accumulated-length constraint or a waiting-time constraint is met (as shown in step 1 and step 2 in Figure 4(b)). The batching-selection can be formalized as a (0-1) knapsack problem: given a set of candidate short requests indexed by i with input lengths w_i and benefit values v_i (e.g., estimated throughput improvement or reduced per-request overhead), choose binary indicators $x_i \in \{0, 1\}$ to

$$\max_{\{x_i\}} \quad \sum_i v_i x_i \qquad (18)$$
$$\text{s.t.} \quad \sum_i w_i x_i \leq W_{\text{batch}},$$
$$x_i \in \{0, 1\},$$

where W_{batch} is the accumulated-length threshold (an operational counterpart in Algorithm 1 i_{next}). In online settings we recommend a greedy heuristic that orders candidates by v_i/w_i (value per token) to maintain low scheduling overhead.

To avoid excessive waiting for short requests, a maximum waiting time is enforced. Once accumulated requests have waited longer than this timeout, the batch is executed immediately (as shown in step 3 in Figure 4(b)). Combining the above models—queueing delay estimates ((10)), and knapsack-style batching selection (Eq. (18))—yields a principled length-aware requests scheduler that (i) sets $T_{timeout}$ as a function of W_q to bound TTFT contribution from queuing, and (ii) forms batches to maximize utilization while limiting per-request added waiting. Algorithm 1 implements this scheduling process. It can be observed that the scheduling routine incurs very low overhead. Its dominant cost lies in queue maintenance, so the time complexity per dequeue operation is $\mathcal{O}(N)$, where N denotes the number of requests in the queue.

D. Modeling of D-instance

The decoding stage is primarily constrained by memory rather than compute. Therefore, D-instances typically process multiple requests concurrently using batching to improve device utilization and system throughput. For LLM inference services, the D-instance batch size directly determines aggregate throughput: larger batch sizes generally yield higher throughput. The maximum feasible batch size for a D-instance is mainly determined by the GPU memory available for storing the KV-cache. Consequently, the distributed deployment parallelism for D-instances must be chosen carefully.

A prevalent strategy is to deploy D-instances using tensor parallelism because with TP way tensor parallelism each GPU holds only $\frac{1}{TP}$ of the model weights, substantially reducing per-GPU memory footprint. This configuration both frees GPU memory for larger KV-cache and distributes computation across GPUs. When batch sizes are large, the relative communication overhead of tensor parallelism is small, making tensor parallelism deployment particularly suitable for D-instances. In general, increasing TP allows a D-instance to
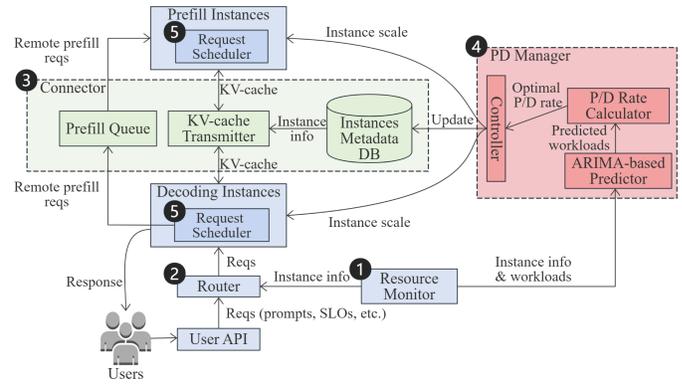

Figure 5: DOPD system architecture.

support a larger maximum batch size and thus improves the system's ability to handle high-concurrency workloads. However, setting TP excessively large may lead to resource waste. Guidelines for selecting an appropriate TP for D-instances are provided in subsection IV-B.

V. SYSTEM DESIGN AND IMPLEMENTATION

We propose a method to accurately compute an appropriate P/D ratio based on current load and to dynamically reconfigure the P-instances and D-instances accordingly, so as to improve throughput with minimal GPU resources and meet the SLO attainment, thereby achieving higher goodput in PD-Disaggregation LLM inference systems. Building on these objectives, we extend the Dynamo framework to realize a dynamically optimal PD-Disaggregation inference system. The system comprises five principal components: (1) Resource Monitor, (2) Router, (3) Connector, (4) PD Manager, and (5) Request Scheduler.

A. Resource Monitor

The Resource Monitor (denoted by ❶ in Figure 5) is a core component responsible for collecting cluster- and instance-level performance metrics and load indicators to construct profiles for short-term load prediction and scheduling decisions. Both the Request Scheduler and the PD Manager rely on the Resource Monitor for up-to-date telemetry. Monitored signals include the number of P-instances, available GPU memory on D-instances, KV-cache utilization on D-instances, the size of the prefill queue, and so on. The Resource Monitor also records recent averages of TTFT, TPOT, ISL, and OSL with Prometheus [37] to meet the SLO attainment and to provide inputs used by the PD Manager and the Request Scheduler.

B. Router

The Router (denoted by ❷) aggregates KV-cache information across all D-instances and routes incoming user requests to the most suitable instance. The Router is integrated with prefix-caching logic. For each request it computes the expected KV-cache hits on each D-instance, combines that information with per-instance load reported by the Resource Monitor, and preferentially forwards the request to the instance with the highest expected cache-hit count and the lowest inferred load.

C. Connector

The Connector (denoted by ❸) provides inter-instance connectivity, maintains the queue of requests awaiting remote prefill, and implements end-to-end KV-cache transfers between P-instances and D-instances. The Connector contains three primary components: (1) an instances metadata DB, (2) a prefill queue, and (3) a KV-cache transmitter. The instances metadata DB is implemented based on etcd service that registers the IDs of all P-instances and D-instances together with the addresses of their KV-cache blocks whenever a new instance joins the cluster. The prefill queue is implemented using a NATS service acting as a queue that holds requests awaiting remote prefill (prefill inference in P-instance). The KV-cache transmitter leverages NVIDIA's open-source NIXL [24] communication library to perform high-performance GPU-to-GPU transfers of KV-cache blocks, thereby migrating KV-cache transfer latency as a dominant bottleneck in PD-Disaggregation.

D. PD Manager

The PD Manager (denoted by ❹) is a component that can calculate the optimal P/D ratio based on historical load and adjust the instances in the system to this ratio. To improve the fidelity of the P/D ratio computation, it is essential to perform pre-deployment profiling of the target model and GPU. Specifically, we first profile a single P-instance across a range of input sequence lengths to obtain accurate TTFT measurements as a function of input length. Similarly, we profile a single D-instance across combinations of sequence lengths and batch sizes to obtain precise TPOT measurements as a function of sequence length and batch size. These profiles guide parallel-configuration choices during scaling and provide expected TTFT/TPOT values for given sequence lengths and concurrency. "Initialize interpolator" in Algorithm 2 performs this process. This profiling overhead is determined by the model size and the GPU specifications; generally, larger model dimensions or lower GPU compute power lead to increased profiling overhead. Taking the combination of the NVIDIA H100 GPU and the LLaMa-3.3-70B-FP8 model as an example, the profiling overhead for TTFT is approximately 90 minutes, while the profiling overhead for TPOT is approximately 150 minutes.

At runtime, the Resource Monitor maintains a recent distribution of request lengths. To avoid the overhead of expensive predictions at scale, we employ a lightweight time-series model (ARIMA) to predict the future average input and output lengths. We compare the predicted values with the recently observed averages from the Resource Monitor to derive correction factors (defined as the ratio of the predicted value to the ground-truth value, used to calibrate subsequent predictions and update ARIMA parameters; this enables the ARIMA predictor to adapt more rapidly to new workloads during periods of high volatility). To reduce forecast error, we combine the derived correction factors with the predicted average sequence lengths and the profiled performance parameters to compute the projected optimal P/D ratio. Practical testing shows that ARIMA's average latency

---

Algorithm 2 Dynamic P/D Ratio Adjustment

1: **Input:**
2: CC_{max}: The D-instance's concurrency cap
3: predictor: ARIMA predictor
4: interpolator: Interpolator for TTFT, TPOT
5: T: Adjust interval
6: t_{last}, t_{cur}: Last record and current timestamps
7: i_{last}, i_{next}: Last and predicted input size
8: o_{last}, o_{next}: Last and predicted output size
9: r_{last}, r_{next}: Last and predicted request count
10: TP_p: Best tensor parallel size of P-instance
11: TP_d: Best tensor parallel size of D-instance
12: $ttft_{next}, tpot_{next}$: Interpolated performance metrics
13: R_{opt}: Optimal P/D (prefill/decoding) ratio
14: n_p, n_d: Next number of P-instance and D-instance
15: adjustPD: Instance adjustment action
16: **Initialize** predictor, interpolator, t_{last}
17: **loop**
18:     update $r_{wait}, load_{kv}, t_{cur}$
19:     **if** checkOverload($r_{wait}, load_{kv}$) **then**
20:         $n_p \leftarrow \lceil r_{wait}/SIZE\_QUEUE \rceil$
21:         $n_d \leftarrow \lceil load_{kv}/KV\_CAP \rceil$
22:         adjustPD(n_d, TP_p, n_p, TP_d)
23:     **end if**
24:     **if** $t_{cur} - t_{last} > T$ **then**
25:         update $i_{last}, o_{last}, r_{last}$
26:         $t_{last} \leftarrow t_{cur}$
27:         predictor.add($i_{last}, o_{last}, r_{last}$)
     ARIMA one-step forecast (where $p, q, \mu, \varphi, \theta, \hat{\varepsilon}$ denote ARIMA parameters and estimated residuals):

$$\hat{X}_{t+1|t} = \mu + \sum_{j=1}^{p} \varphi_j X_{t+1-j} + \sum_{k=1}^{q} \theta_k \hat{\varepsilon}_{t+1-k}$$

28:         $i_{next} = \hat{I}_{t+1|t}, o_{next} = \hat{O}_{t+1|t}, r_{next} = \hat{R}_{t+1|t}$
29:         $ttft_{next} \leftarrow$ interpolator(i_{next})
30:         $tpot_{next} \leftarrow$ interpolator($r_{next}, \frac{i_{next}+o_{next}}{2}$)
31:         $R_{opt} \leftarrow \dfrac{r_{next}}{(o_{next} \times tpot_{next})/ttft_{next}}$
32:         $n_d \leftarrow \lceil r_{next}/CC_{max} \rceil$
33:         $n_p \leftarrow R_{opt} \times n_d$
34:         adjustPD(n_p, TP_p, n_d, TP_d)
35:     **end if**
36: **end loop**

---

is under 0.8s, a negligible overhead compared to the hundred-second reconfiguration intervals. Under real-world workloads, ARIMA maintains over 87% accuracy (defined by errors within 10 requests for future concurrency and 50 tokens for ISL and OSL) failing only during rare bursty conditions ($< 5\%$). To handle such unpredictable spikes, the system adaptively monitors prefill queue depth (r_{queue}) and KV-cache occupancy ($load_{kv}$). Upon detecting imminent overload, the system bypasses the scheduled interval and immediately scales P-instances to $r_{queue}/SIZE\_QUEUE$ and D-instances to $load_{kv}/KV\_CAP$, where the $SIZE\_QUEUE$ denotes the size of prefill queue, and KV denotes the maximum adjust-
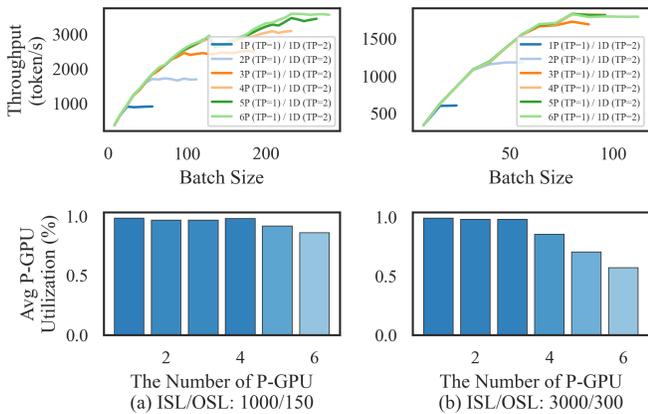
Figure 6: Optimal P/D ratio validation experiment results for varying numbers of P-instances and a single D-instance with $TP = 2$ under (a) fixed input/output sequence length of $1000/150$ (optimal P/D ratio is approximately 4.58). (b) fixed input/output sequence length of $3000/300$ (optimal P/D ratio is approximately 3.18).

Figure 7: The performance comparison in terms of (a) goodput, (b) end-to-end latency, (c) TTFT, and (d) TPOT of a static P/D ratio on real Azure workload experiments with LLaMa-3.3-70B-FP8 model, compared with aggregated methods.

ment threshold of KV-cache memory utilization, as detailed in Algorithm 2 (lines 18–23). To further mitigate resource wastage caused by occasional over-prediction, we pre-define a maximum adjustment threshold for the number of instances, constraining the scale of individual reconfiguration operations.

Algorithm 2 summarizes the procedure for forecasting future load, deriving the optimal P/D ratio, and executing instance adjustments. DOPD first jointly considers the factors listed above to determine the optimal tensor parallel size TP for P-instances and D-instances, and then performs instance adjustments according to the computed optimal P/D ratio. This design enables accurate forecasting of a future, workload-appropriate P/D ratio from historical telemetry and supports non-disruptive (zero-downtime) elastic resizing of P-instances and D-instances to promptly accommodate time-varying user request loads.

E. Request Scheduler

The Request Scheduler (denoted by ❺) is built into P-instances and D-instances to arrange requests reasonably, allowing the PD-Disaggregation system to better cope with mixed requests. To implement this, we group and batch requests by their ISL. Under high concurrency, short requests are batched according to their predicted future lengths to leverage kernel efficiency, reducing the mismatch between short-length load and the provisioned P/D ratio. Long requests, whose computational cost is inherently high, are not merged and are dispatched to P-instances for immediate execution.

For ultra-short requests we employ PD-aggregation: merging their prefill with other requests' decoding on D-instances avoids unnecessary KV-cache transfers and relieves prefill queue pressure (illustrated by the yellow blocks in step 1 and step 3 of the flow in Figure 4(b)). Using this request-scheduling scheme, our experiments on realistic workloads demonstrate substantial improvements in maximum concurrency, throughput, and TTFT without changing the global P/D ratio. Detailed results appear in Section VI.
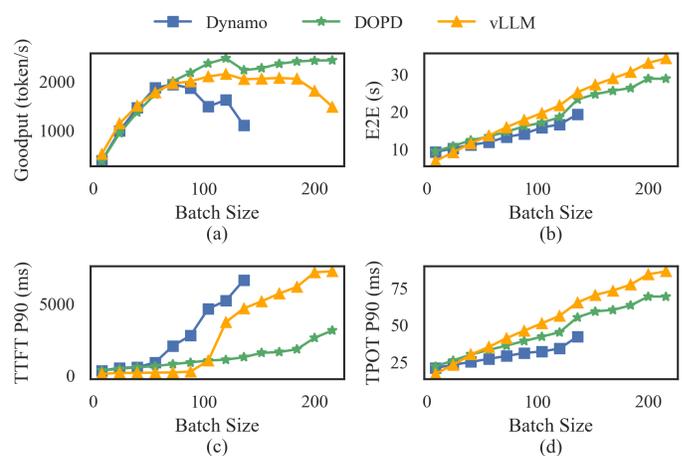
VI. PERFORMANCE EVALUATION

This section carries out experimental evaluation of the proposed DOPD system. It discusses configuration and analyses of our experiments and results along with scalability of DOPD and the directions of future research.

A. Evaluation Setup

1) Testbed: We conduct experiments on a local cluster composed of 8 NVIDIA H100-SXM-80GB [12] GPUs.

2) LLM models: For inference-performance evaluation we select three representative models used in prior works and industries test suites: OPT-30B [38], LLaMa-3.3-70B-FP8 [23] and QWen2.5-72B-FP8 [39].

3) Datasets: We employ three production workload traces collected from Microsoft (BurstGPT [40], Azure code Traces [41] and Azure conversation [41]), and we use the ShareGPT [42] dataset for request contents. These traces provide realistic workload characteristics.

4) Resource monitoring: To collect and aggregate telemetry we leverage NVIDIA's NVML [43] for low-level GPU metrics and Prometheus [37] for metrics aggregation and retention.

5) Baselines: We divide the experiment into static experiments and dynamic experiments.

a) Static experiments: Under a fixed P/D configuration we compare the following systems:

- **DistServe** [19]: a representative PD-Disaggregation framework with strong performance characteristics.
- **Dynamo** [34]: the most feature-complete open-source PD-Disaggregation framework.
- **vLLM** [26]: a state-of-the-art open-source LLM inference system with broad adoption.

b) Dynamic experiments.: Because mature dynamic PD-Disaggregation approaches are not open-source and widely available apart from Dynamo, we compare DOPD against two Dynamo scheduling strategies:

- **DYN-LOAD**: a load-based scheduler that scales based on runtime utilization thresholds.
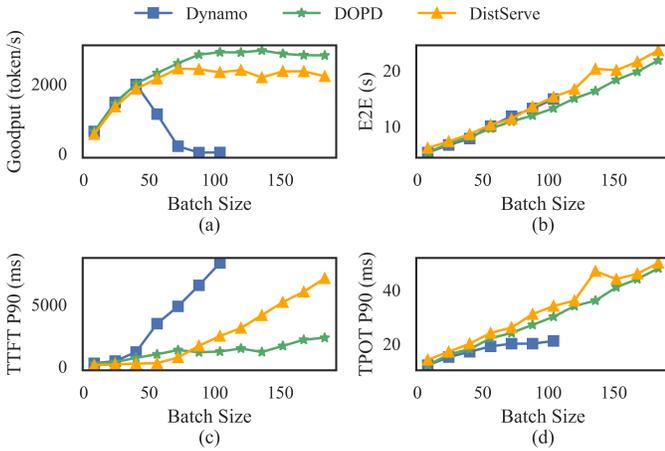
Figure 8: The performance comparison in terms of (a) goodput, (b) end-to-end latency, (c) TTFT, and (d) TPOT of a static P/D ratio on real Azure workload experiments with OPT-30B model, compared with disaggregated methods.

Figure 9: The performance comparison in terms of (a) end-to-end latency, (b) throughput and goodput, (c) TTFT, and (d) TPOT of dynamic scheduling experiments under real-world BurstGPT workloads.

- **DYN-SLA**: an SLA-aware scheduler that derives scaling decisions from real-time inference monitor metrics.

B. Evaluation Results

1) Validation of the Optimal P/D Ratio: To validate the correctness of the computed Optimal P/D Ratio, we first deploy Dynamo with the LLaMa-3.3-70B-FP8 model, varying the number of P-instances (each on 1 H100 GPU) while fixing the number of D-instances to one (with $TP = 2$ across 2 H100 GPUs). Prior to service launch, our computation yields an optimal P/D ratio of approximately 4.58, indicating that roughly 4.58 P-instances suffice to balance production and consumption.

During each experiment we issue inference requests with fixed input length $ISL = 1000$ and output length $OSL = 150$ at varying concurrency levels (concurrency is defined as the number of in-flight requests in system). Figure 6(a) illustrates throughput under the fixed $1000/150$ workload. It shows a near-linear increase in maximum throughput as the number of P-instances increases from 1 to 4. Beyond 4, throughput growth from 4 to 5 slows markedly, and from 5 to 6 is negligible. Concurrently, the average GPU utilization of P-instances (as shown in the lower figure in Figure 6(a)) indicates idle GPU time only when the number of P-instances exceeds 4. This proves that our pre-computed optimal number of P-instances is correct (between 4 and 5).

We further compute that for input/output sequence length of $3000/300$, the corresponding optimal number of P-instances is 3.18. Repeating the experiment under the $3000/300$ workload produces analogous results (as shown in Figure 6(b)), further confirming the accuracy of our calculation method.

2) Comparison of Static P/D Inference-Performance: To compare the performance of DOPD with representative PD-aggregated and PD-disaggregated approaches under a static P/D configuration, we extract the first 1024 entries from Microsoft Azure traces and use ShareGPT as the source of request content, generate a request set with average input/output
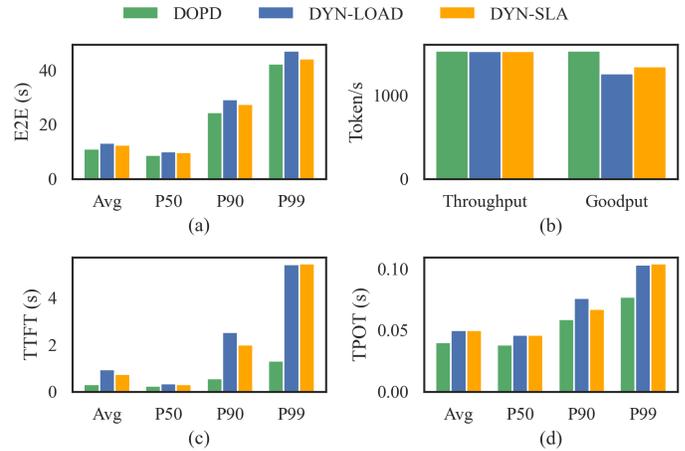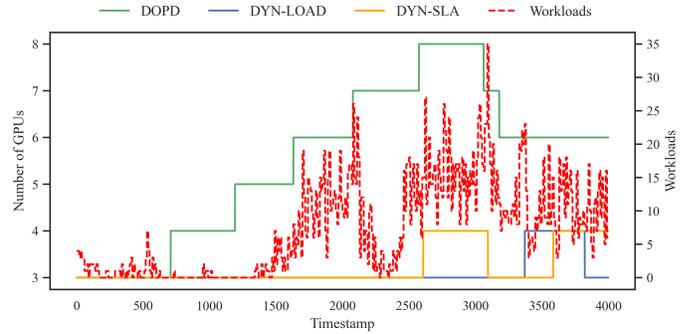

Figure 10: GPU provisioning and workload variation over time. Solid lines (left axis) denote the GPU counts for DOPD, DYN-LOAD, and DYN-SLA, while the dashed line (right axis) represents the instantaneous request rate.

sequence length of approximately $1024/245$. We then show two sets of experiments.

First, we compare DOPD with a representative PD-aggregation system and a PD-Disaggregation baseline using the LLaMa-3.3-70B-FP8 model. The aggregation baseline (vLLM) is deployed on 4 H100 GPUs, whereas both disaggregation baselines (Dynamo and DOPD) are configured as 2P (TP=1) / 1D (TP=2). We generate inference workload from the Microsoft Azure traces and replay it to each deployment. Results are reported in Figure 7, which shows goodput, TTFT, TPOT, and end-to-end latency under the same workload. Compared to vLLM, DOPD reduces P90 TPOT from 0.097s up to 0.079s and increases goodput by up to $1.5\times$. Under reasonable SLO constraints (violation rate $< 5\%$), DOPD also supports a substantially higher maximum concurrency, rising from 72 to 112.

Second, we evaluate three PD-Disaggregation systems using the facebook/opt-30b model. In these experiments each system are configured as 1P (TP=1) / 1D (TP=4). Results appear in Figure 8. As concurrency increases, DOPD's benefits become pronounced: relative to a representative disaggregation system (DistServe), DOPD reduces P90 TTFT from 7.10s to 2.31s and improves goodput by up to 27%. Under the SLO constraints
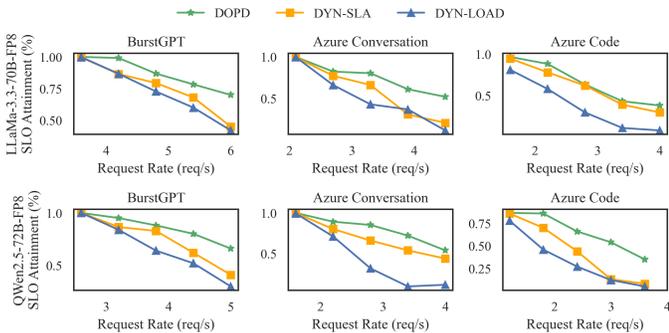
Figure 11: SLO attachment performance of different workload datasets under different requests rate and different scheduling policies for different models.

(violation rate $< 5\%$), the maximum supported concurrency increases from 88 to 104. Through DOPD's length-aware scheduling, it significantly outperforms Dynamo across all evaluation metrics.

3) Dynamic P/D Instance-Scheduling Performance Comparison: To test the performance of DOPD and other dynamic instance-scheduling methods, DOPD significantly outperforms Dynamo's built-in load-based and SLA-aware scheduling strategies under complex and highly variable inference workloads. We combine the workload of the BurstGPT dataset with the request content of the ShareGPT dataset to generate a simulated LLM inference workload. According to the timestamps in the BurstGPT trace, we replay requests to three service deployments. All three systems (DOPD, DYN-LOAD, DYN-SLA) are initially configured with one P-instance ($TP = 1$) and one D-instance ($TP = 2$) running the LLaMa-3.3-70B-FP8 model. Figure 10 shows the number of requests versus time (sampled every 0.1s), while the other curves plot the total number of GPUs provisioned over time under each scheduling strategy. Although DYN-SLA and DYN-LOAD adjust resource allocation in response to load changes, their scaling amplitudes and timeliness are insufficient. In contrast, DOPD's GPU count closely tracks the RPS curve. It scales out rapidly during traffic spikes and scales in promptly when load subsides. Figure 9 presents the results. From the four plots it is clear that DOPD's end-to-end (E2E) latency, TTFT, TPOT, and goodput all outperform the other baselines. As a result, DOPD achieves excellent SLO attainment (99.4%), whereas DYN-SLA and DYN-LOAD incur worse SLO attainment (87.3% and 80.8% respectively). These results demonstrate that DOPD can judiciously scale P-instances and D-instances in response to load fluctuations, substantially increasing the SLO attainment.

To further demonstrate the versatility and robustness of our approach across diverse workloads and model architectures, we conducted extensive evaluations using traces derived from the BurstGPT, Azure Conversation, and Azure Code datasets. As illustrated in Figure 11, we evaluated the end-to-end performance under varying request rates for both LLaMa-3.3-70B-FP8 and QWen2.5-72B-FP8 models. For the LLaMa-3.3-70B-FP8 model, although all three scheduling strategies exhibit a decline in SLO attainment at high request rates due

to limited hardware resources, DOPD consistently achieves the most graceful degradation, maintaining the highest SLO attainment across all datasets. In contrast, the DYN-LOAD baseline performs poorly, particularly under high concurrency, as its reactive adjustment strategy only triggers reconfiguration after the system has already reached an overloaded state. This latency prevents the system from proactively adapting to traffic surges, resulting in severe SLO violations. Similarly, for the QWen2.5-72B-FP8 model, DOPD maintains its superior performance with the slowest rate of SLO decline. Due to the lack of guidance for computing an optimal P/D ratio, DYN-SLA exhibits highly unstable performance and struggles to balance P-instances and D-instances, which in turn leads to a high rate of SLO violations. Notably, all methods face challenges when processing workloads with a high proportion of long-context requests. This performance bottleneck stems from both hardware quantity constraints and the inherent computational intensity of long-sequence prefill, highlighting a critical area for future research focused on optimizing tail latency in long-context inference scenarios.

VII. DISCUSSION

The cost of scheduling and scalability. DOPD demonstrates high efficiency and scalability with minimal scheduling overhead. The PD Manager's execution, dominated by ARIMA forecasting, completes within 1s. As this process is asynchronous and occurs at infrequent intervals (hundreds of seconds), the resulting CPU overhead is negligible. Instance reconfiguration is equally efficient: graceful shutdown typically finishes within 5s, while scaling up a 70B model takes ≤ 30s for a P-instance (and $\approx 1.5\times$ longer for a D-instance due to CUDA Graph loading). Crucially, accurate workload prediction enables proactive scaling, effectively masking these initialization latencies. Furthermore, the Request Scheduler incurs only microsecond-level latency, and the instance-decoupled architecture inherently facilitates seamless scaling to large-scale clusters.

Limitation. Despite DOPD's robustness on realistic production traces, occasional ultra-long prompts can transiently degrade overall system performance. How to rapidly and efficiently process ultra-long context requests has remained a formidable challenge since the emergence of LLMs. Furthermore, in most current LLM inference scenarios, ultra-long context requests are rare and therefore difficult for the predictor to anticipate. Scheduling decisions that rely on average-length signals may perform poorly when extremely long contexts suddenly appear. A potential and straightforward solution is to deploy an auxiliary specialized node dedicated to prioritizing ultra-long requests. Another limitation of our work is that the current approach does not yet support the dynamic selection of different TP degrees for varying workloads, as this would introduce significantly more complex P/D ratio calculations and more intricate logic for instance reconfiguration. However, the system inherently supports the scaling or removal of instances with different TP configurations. In the future, we can further refine our algorithm by modeling these more complex scenarios.

Future work and extension. Future work can explore targeted mitigations for ultra-long prompt interference and investigate integrations with sparse attention mechanisms or speculative-decoding architectures to enhance generality and tail robustness. Meanwhile, as user demand for Multimodal Large Language Models (MLLMs) increases, we also consider further extending our methodology by integrating it with emerging techniques (such as those described in [44] and [45]) to optimize the inference performance of MLLMs. Nevertheless, DOPD provides a principled foundation for dynamic resource management in disaggregated LLM inference, striking a balance between scalability and operational practicality.

VIII. CONCLUSIONS

In this paper, we proposed DOPD, a dynamic framework that combines PD-Disaggregation with dynamic adjustment of instances to improve goodput, reduce inference latency, achieve more stringent SLOs, and conserve GPU resources. We introduce a method for calculating the optimal P/D ratio and a complementary request scheduling strategy that enhances the adaptation of disaggregated systems to mixed-length workloads. The proposed P/D ratio calculation together with workload-aware scheduling enables DOPD to achieve excellent performance under complex, time-varying workloads. Through extensive experiments on realistic production traces and representative LLMs, we demonstrate that DOPD achieves up to $1.5\times$ improvement in goodput while simultaneously reducing inference latency. DOPD also achieves a near-zero SLO violation rate under real-world workloads and reliably detects load fluctuations to trigger timely elastic scaling. Our results indicate that DOPD substantially improves the inference efficiency of PD-Disaggregation architectures for LLMs, thereby facilitating their deployment in industrial environments with demanding availability and performance requirements.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar *et al.*, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NY, USA: Curran Associates Inc., 2017, p. 6000–6010.

[2] openai. (2024) Introducing apis for gpt-3.5 turbo and whisper. Accessed: 2025-07-01. [Online]. Available: https://openai.com/index/introducing-chatgpt-and-whisper-apis

[3] H. Touvron, L. Martin, K. Stone *et al.*, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: https://arxiv.org/abs/2307.09288

[4] B. Hui, J. Yang, Z. Cui *et al.*, "Qwen2. 5-coder technical report," *arXiv preprint arXiv:2409.12186*, 2024.

[5] openai. (2025) Introducing gpt-5. Accessed: 2025-07-01. [Online]. Available: https://openai.com/index/introducing-gpt-5

[6] meta. (2025) The llama 4 herd: The beginning of a new era of natively multimodal ai innovation. Accessed: 2025-07-01. [Online]. Available: https://ai.meta.com/blog/llama-4-multimodal-intelligence

[7] J. Ribas. (2023) New bing. Accessed: 2025-07-01. [Online]. Available: https://blogs.bing.com/search-quality-insights/february-2023/Building-the-New-Bing

[8] Google. (2025) Google gemini. Accessed: 2025-07-01. [Online]. Available: https://blog.google/products/gemini

[9] Cursor. (2025) Cursor. Accessed: 2025-07-01. [Online]. Available: https://cursor.com

[10] Y. Sun, S. Ma, T. Xiao *et al.*, " Accurate and Interpretable Log-Based Fault Diagnosis using Large Language Models ," *IEEE Transactions on Services Computing*, no. 01, pp. 1–14, Aug. 2025.

[11] K. Team, Y. Bai, Y. Bao *et al.*, "Kimi k2: Open agentic intelligence," 2025. [Online]. Available: https://arxiv.org/abs/2507.20534

[12] Nvidia. (2022) Nvidia h100 tensor core gpu. Accessed: 2025-07-01. [Online]. Available: https://www.nvidia.com/en-us/data-center/h100

[13] Z. Yao, Z. Tang, W. Yang *et al.*, " Enhancing LLM QoS Through Cloud-Edge Collaboration: A Diffusion-Based Multi-Agent Reinforcement Learning Approach ," *IEEE Transactions on Services Computing*, vol. 18, no. 03, pp. 1412–1427, May 2025.

[14] Z. Li, W. Feng, M. Guizani *et al.*, "Tpi-llm: Serving 70b-scale llms efficiently on low-resource mobile devices," *IEEE Transactions on Services Computing*, pp. 1–13, 2025.

[15] J. Hu, M. Xu, K. Ye *et al.*, " BrownoutServe: SLO-Aware Inference Serving under Bursty Workloads for MoE-based LLMs ," *IEEE Transactions on Computers*, no. 01, pp. 1–14, Jan. 2026. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TC.2026.3655019

[16] Y. Li, J. Guo, Z. Tang *et al.*, " Cloud-Edge System for Scheduling Unpredictable LLM Requests with Combinatorial Bandit ," *IEEE Transactions on Services Computing*, no. 01, pp. 1–15, Sep. 2025.

[17] T. Dao, D. Fu, S. Ermon *et al.*, "Flashattention: Fast and memory-efficient exact attention with io-awareness," in *Advances in Neural Information Processing Systems*, vol. 35. Curran Associates, Inc., 2022, pp. 16 344–16 359.

[18] R. A. Jacobs, M. I. Jordan, S. J. Nowlan *et al.*, "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, no. 1, pp. 79–87, 1991.

[19] Y. Zhong, S. Liu, J. Chen *et al.*, "Distserve: disaggregating prefill and decoding for goodput-optimized large language model serving," in *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*. USA: USENIX Association, 2024.

[20] P. Patel, E. Choukse, C. Zhang *et al.*, "Splitwise: Efficient generative llm inference using phase splitting," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture*, 2024, pp. 118–132.

[21] M. Wang, Y. Ye, and Z. Zhou, "Llm serving optimization with variable prefill and decode lengths," 2025. [Online]. Available: https://arxiv.org/abs/2508.06133

[22] C. Hu, H. Huang, L. Xu *et al.*, "Shuffleinfer: Disaggregate llm inference for mixed downstream workloads," *ACM Trans. Archit. Code Optim.*, vol. 22, no. 2, Jul. 2025. [Online]. Available: https://doi.org/10.1145/3732941

[23] RedHatAI. (2025) Redhatai/deepseek-r1-distill-llama-70b-fp8-dynamic. Accessed: 2025-07-01. [Online]. Available: https://huggingface.co/RedHatAI/DeepSeek-R1-Distill-Llama-70B-FP8-dynamic

[24] Nvidia. (2025) Nixl. Accessed: 2025-07-01. [Online]. Available: https://github.com/ai-dynamo/nixl

[25] G.-I. Yu, J. S. Jeong, G.-W. Kim *et al.*, "Orca: A distributed serving system for Transformer-Based generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 521–538.

[26] W. Kwon, Z. Li, S. Zhuang *et al.*, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[27] A. Agrawal, N. Kedia, A. Panwar *et al.*, "Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve," in *18th USENIX Symposium on Operating Systems Design and Implementation*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 117–134.

[28] A. K. Kamath, R. Prabhu, J. Mohan *et al.*, *POD-Attention: Unlocking Full Prefill-Decode Overlap for Faster LLM Inference*. NY, USA: Association for Computing Machinery, 2025, p. 897–912.

[29] L. Zheng, L. Yin, Z. Xie *et al.*, "Sglang: efficient execution of structured language model programs," in *Proceedings of the 38th International Conference on Neural Information Processing Systems*. NY, USA: Curran Associates Inc., 2024.

[30] J. Yuan, H. Gao, D. Dai *et al.*, "Native sparse attention: Hardware-aligned and natively trainable sparse attention," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vienna, Austria: Association for Computational Linguistics, Jul. 2025, pp. 23 078–23 097.

[31] R. Qin, Z. Li, W. He *et al.*, "Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot," in *23rd USENIX Conference on File and Storage Technologies*. Santa Clara, CA: USENIX Association, Feb. 2025, pp. 155–170.

[32] Y. Jin, T. Wang, H. Lin *et al.*, "P/d-serve: Serving disaggregated large language model at scale," 2024. [Online]. Available: https://arxiv.org/abs/2408.08147

This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2026.3670011

IEEE TRANSACTIONS ON SERVICES COMPUTING

14

[33] Y. Wu, T. Liu, Y. Zeng *et al.*, "Arrow: Adaptive scheduling mechanisms for disaggregated llm inference architecture," 2025. [Online]. Available: https://arxiv.org/abs/2505.11916

[34] Nvidia. (2025) ai-dynamo. Accessed: 2025-07-01. [Online]. Available: https://github.com/ai-dynamo/dynamo

[35] G. E. P. Box and G. Jenkins, *Time Series Analysis, Forecasting and Control*. USA: Holden-Day, Inc., 1990.

[36] J. Bai, S. Bai, Y. Chu *et al.*, "Qwen technical report," 2023. [Online]. Available: https://arxiv.org/abs/2309.16609

[37] J. Ribas. (2015) prometheus. Accessed: 2025-07-01. [Online]. Available: https://github.com/prometheus/prometheus

[38] facebook. (2022) facebook/opt-30b. Accessed: 2025-07-01. [Online]. Available: https://huggingface.co/facebook/opt-30b

[39] RedHatAI. (2024) Redhatai/qwen2.5-72b-instruct-fp8-dynamic. Accessed: 2026-01-05. [Online]. Available: https://huggingface.co/RedHatAI/Qwen2.5-72B-Instruct-FP8-dynamic

[40] Y. Wang, Y. Chen, Z. Li *et al.*, "Burstgpt: A real-world workload dataset to optimize llm serving systems," 2025. [Online]. Available: https://arxiv.org/abs/2401.17644

[41] Microsoft. (2023) Azurepublicdataset. Accessed: 2025-07-01. [Online]. Available: https://github.com/Azure/AzurePublicDataset

[42] anon8231489123. (2023) Sharegpt. Accessed: 2025-07-01. [Online]. Available: https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered

[43] Nvidia. (2025) Nvidia management library (nvml). Accessed: 2025-07-01. [Online]. Available: https://developer.nvidia.com/management-library-nvml

[44] G. Singh, X. Wang, Y. Hu *et al.*, "Efficiently serving large multimodal models using EPD disaggregation," in *Forty-second International Conference on Machine Learning*, 2025.

[45] H. Qiu, A. Biswas, Z. Zhao *et al.*, "Modserve: Modality- and stage-aware resource disaggregation for scalable multimodal model serving," in *Proceedings of the 2025 ACM Symposium on Cloud Computing*, New York, USA, 2026, p. 817–830.

Wanyi Zheng received her Bachelor's degree from Hebei University in 2025. She is currently pursuing a Master's degree at the Southern University of Science and Technology, and is also a joint-training student at the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. Her research interests include operating systems and the integration of large language model systems, with a primary focus on LLM inference optimization and system management.

Yan Wang (Member, IEEE) received the PhD degree in computer science from Inner Mongolia University, Hohhot, China, in 2015. She is currently an Associate Professor and a Master Supervisor of Computer Science and Technology with Inner Mongolia University. She has published more than 40 papers in international conferences and journals in her field. Her research interests include service computing, formal methods, and software technology.

Kejiang Ye (Senior Member, IEEE) received the BSc and PhD degrees from Zhejiang University, in 2008 and 2013, respectively. He was also a joint PhD student with the University of Sydney from 2012 to 2013. After graduation, he works as post-doc researcher with Carnegie Mellon University from 2014 to 2015 and Wayne State University from 2015 to 2016. He is currently a professor with the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. His research interests focus on the performance, energy, and reliability of cloud computing and network systems.

Junhan Liao received his bachelor's degree in computer science and technology from Hunan University of Technology. Currently, he is a master's student at Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. His main research interests include large language models inference optimization and system management.

Rajkumar Buyya (Fellow, IEEE) is currently a Redmond Barry distinguished professor and director with the Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, the University of Melbourne, Australia. He has authored more than 625 publications and seven textbooks including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets, respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=175, i10-index=837, 164,000+citations).

Minxian Xu (Senior Member, IEEE) is currently an Associate Professor at Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. He received his PhD degree from the University of Melbourne in 2019. His research interests include resource scheduling and optimization in cloud computing. He has co-authored 80+ peer-reviewed papers published in prominent international journals and conferences. His PhD thesis was awarded the 2019 IEEE TCSC Outstanding Ph.D. Dissertation Award. He was also awarded the 2023 IEEE TCSC Award for Excellence (Early Career Award).

Chengzhong Xu (Fellow, IEEE) received the Ph.D. degree in computer science and engineering from the University of Hong Kong in 1993. He is the Dean of Faculty of Science and Technology and the Interim Director of Institute of Collaborative Innovation, University of Macau. He published two research monographs and more than 300 peer-reviewed papers in journals and conference proceedings. His papers received about 17K citations with an H-index of 72. His main research interests lie in parallel and distributed computing and cloud computing.