IEEE Open Journal of the
Computer Society

Received 4 February 2024; accepted 15 March 2024. Date of publication 22 March 2024;
date of current version 10 April 2024. The review of this article was arranged by Associate Editor Peng Li.

*Digital Object Identifier 10.1109/OJCS.2024.3380828*

# Training and Serving System of Foundation Models: A Comprehensive Survey

**JIAHANG ZHOU** [1], **YANYU CHEN** [2], **ZICONG HONG** [4], **WUHUI CHEN** [3,5] **(Member, IEEE)**, **YUE YU** [5] **(Member, IEEE)**, **TAO ZHANG** [1], **HUI WANG** [5], **CHUANFU ZHANG** [1], **AND ZIBIN ZHENG** [3] **(Senior Member, IEEE)**

[1]School of Systems Science and Engineering, Sun Yat-sen University, Guangzhou 528406, China
[2]School of Informatics, Xiamen University, Xiamen 361005, China
[3]School of Software Engineering, Sun Yat-sen University, Guangzhou 510275, China
[4]Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR 999077, China
[5]Peng Cheng Laboratory, Shenzhen 518000, China

CORRESPONDING AUTHOR: WUHUI CHEN (e-mail: chenwuh@mail.sysu.edu.cn).

This work was supported in part by the National Key Research and Development Plan under Grant 2022ZD0115301, in part by the National Natural Science Foundation of China under Grant 62172453, in part by the National Natural Science Foundation of Guangdong province under Grant 2022A1515010154, in part by the Major Key Project of PCL under Grant PCL2023AS7-1 sponsored by CAAI-MindSpore Open Fund, developed on OpenI Community, and in part by the Pearl River Talent Recruitment Program under Grant 2019QN01X130.

**ABSTRACT** Foundation models (e.g., ChatGPT, DALL-E, PengCheng Mind, PanGu-$\Sigma$) have demonstrated extraordinary performance in key technological areas, such as natural language processing and visual recognition, and have become the mainstream trend of artificial general intelligence. This has led more and more major technology giants to dedicate significant human and financial resources to actively develop their foundation model systems, which drives continuous growth of these models' parameters. As a result, the training and serving of these models have posed significant challenges, including substantial computing power, memory consumption, bandwidth demands, etc. Therefore, employing efficient training and serving strategies becomes particularly crucial. Many researchers have actively explored and proposed effective methods. So, a comprehensive survey of them is essential for system developers and researchers. This paper extensively explores the methods employed in training and serving foundation models from various perspectives. It provides a detailed categorization of these state-of-the-art methods, including finer aspects such as network, computing, and storage. Additionally, the paper summarizes the challenges and presents a perspective on the future development direction of foundation model systems. Through comprehensive discussion and analysis, it hopes to provide a solid theoretical basis and practical guidance for future research and applications, promoting continuous innovation and development in foundation model systems.

**INDEX TERMS** Foundation model system, training, serving, network, computing, storage.

## I. INTRODUCTION

The combination of deep learning techniques and powerful computational capabilities continuously drives the development of artificial general intelligence, ushering us into the era of foundation models. However, achieving successful applications of foundation models is inseparable from comprehensive support at the system level. A foundation model system is built upon extensive training data, state-of-the-art models, high-performance computing resources, and meticulously optimized training and serving algorithms. The primary purpose of this system is to handle complex tasks with heightened precision, such as GPT3 [1], LLaMA [2], PanGu-$\Sigma$ [3], PengCheng Mind [4] etc.

Foundation models have demonstrated extraordinary performance in many tasks. This has led more and more major technology giants to dedicate significant human and financial resources to actively develop their foundation model systems, which increases the parameter size (Fig. 1). However, as the parameter size of foundational model systems continues to grow, challenges are posed throughout the lifecycle of foundation models, particularly during the training and serving phases. In the training phase, the substantial

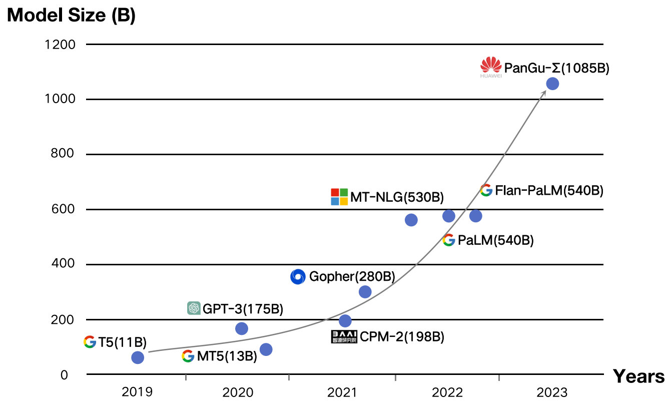

**FIGURE 1.** Evolutionary Chart of Model Sizes Over Time.

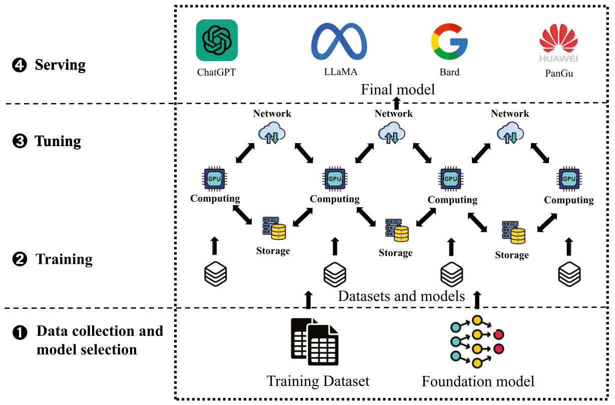

**FIGURE 2.** Lifecycle of the foundation model system.

parameter size results in significant demands for computation and storage, creating immense pressure on hardware resources and computational efficiency. Consequently, training these models usually takes a long time and requires efficient utilization of computational resources. In the serving phase, with the widespread application of foundation models, the significant increase in workload has become an unavoidable challenge. This heightened demand may lead to issues for serving systems, such as latency, performance decline, or resource bottlenecks. Therefore, employing highly efficient training and serving strategies becomes particularly crucial. Many researchers have actively explored and proposed effective methods for training and serving. However, different approaches have different application scenarios. So, it poses a challenge for system developers who struggle to identify the most suitable method for their problems. This challenge is precisely why this paper was proposed.

Although there have been some surveys on foundation models, Most surveys [5], [6], [7], [8], [9], [10], [11] predominantly focus on model design and downstream task adaptation, with only a minority delving into foundation model training. However, there are two notable shortcomings in these training-centric surveys [12]: firstly, they lack in-depth exploration from the perspective of updates in network, computing, and storage; secondly, their primary emphasis is on the training phase, neglecting considerations for the serving phase. Therefore, a comprehensive survey of foundation model training and serving methods is essential for system developers and researchers. Accordingly, this paper presents an in-depth analysis of the state-of-the-art methods in this domain. This paper provides systems developers and researchers valuable information through comprehensive analysis and comparison. It assists them in making the right decisions when confronted with the challenges associated with foundation model systems.

## II. BASIC CONCEPTS

This section comprehensively explains the fundamental concepts in foundation model systems.

### A. THE LIFECYCLE OF THE FOUNDATION MODEL SYSTEM

The lifecycle of the foundation model system (Fig. 2) encompasses several crucial stages. ❶ Initially, the collection and preprocessing of data ensure the quality and availability required for model training. Subsequently, choosing an appropriate model. ❷ Transitioning to the training phase, the model undergoes adjustments through the backpropagation algorithm, demanding substantial computational resources to enhance its fitting capability to the training data. ❸ Model evaluation and fine-tuning involve assessing performance with test data and adjusting for improved generalization. Once the model performs satisfactorily, it can be deployed into practical applications. ❹ In the serving stage, effective deployment and integration are crucial to ensuring harmonious collaboration with existing systems. The primary focus in this phase centers on performance optimization, aiming to enhance serving speed and reduce latency through strategies such as model quantization and hardware acceleration.

### B. TRANSFORMER FOR FOUNDATION MODELS

Transformer [13] is a deep learning model architecture comprised of encoders and decoders. Its core innovation lies in the self-attention mechanism, an important component widely utilized in foundational models. The main idea is to enable the model to focus on dynamic associations between different positions, thereby better capturing long-distance interdependent features in a sentence. In the current field of deep learning, the Transformer architecture has become the preferred choice for numerous foundational models. This architecture stands out for its outstanding performance and flexibility, particularly in excelling at natural language processing tasks. Many pivotal foundational models, such as GPT, LLaMA, and PengCheng Mind, have adopted the design of the Transformer. The successful applications of the Transformer architecture demonstrate its universality in foundational models, providing powerful modeling tools for various tasks.

## III. MODEL TRAINING

In foundation model training, the most significant challenges are the high demands for memory and computational power.

**TABLE 1. Overview of Network, Computing, and Storage Optimization Strategies in Foundation Model Training**

| Parallel Computing | | | | | |
|---|---|---|---|---|---|
| **Parallelism Type** | **Specific Strategy** | **Year** | **Main Features** | **Scales (M, B, 10B, 100B+)** | **Open Resource** |
| Data Parallelism | DDP [14] | 2020 | Bucketing gradients, Skipping gradient synchronization | M | ✓ |
| | Xu et al. [16] | 2020 | Automatic cross-replica sharding, Efficient weight-update sharding | M | × |
| | FSDP [15] | 2023 | Fully sharded data parallel | 100B+ | ✓ |
| Tensor Parallelism | Megatron-LM [17] | 2021 | Weight matrix partitioned | 100B+ | ✓ |
| | Optimus [18] | 2023 | 2D partition gradient computation, Systematic buffering technique | 100B+ | ✓ |
| | Tesseract [20] | 2023 | 2.5-Dimensional Matrix Multiplication, Reducing communication overhead | B | × |
| | 3D Tensor Parallelism [22] | 2021 | 3-dimensional model parallelism, Perfect load balance | M | × |
| Pipeline Parallelism | GPipe [23] | 2019 | Novel batchsplitting pipelining | 10B | ✓ |
| | PipeDream [24] | 2019 | 1F1B, Weight stashing, Vertical sync | M | × |
| | Megatron-LM [17] | 2021 | Schedule with Interleaved Stages | 100B+ | ✓ |
| | Chimera [28] | 2021 | Bidirectional pipelines, More balanced activation memory consumption | B | ✓ |
| | PipeDream-2BM [27] | 2021 | Memory-efficient pipeline parallelism, Weight gradient coalescing strategy | B | × |
| | FTPipe [35] | 2021 | Mixed-pipe partitioning, Better balance of the compute- and memory-load | 10B | ✓ |
| | DAPPLE [26] | 2022 | Synchronous training, Topology-aware device assignment | B | × |
| | Varuna [25] | 2022 | Recomputation, Correctness-preserving job morphing | 100B+ | ✓ |
| | Hanayo [29] | 2023 | Wave-like pipeline, Unified framework for pipeline parallelism | 100B+ | × |
| | Mixpipe [30] | 2023 | Mixed scheduling, Flexible bidirectional pipeline | 10B | × |
| | Avgpipe [31] | 2023 | Elastic averaging training method | M | × |
| | Bpipe [33] | 2023 | Memory-balanced pipeline parallelism | 100B+ | × |
| | Bamboo [34] | 2023 | Redundant computation, Use of preemptible instances | M | × |
| | Dynapipe [32] | 2024 | Dynamic micro-batching pipeline | 10B | ✓ |
| Expert Parallelism | GShard [36] | 2021 | Sparsely-Gated Mixture-of-Experts | 100B+ | × |
| | FastMoE [38] | 2021 | Distributed MoE training system, Open-source system based on PyTorch | 100B+ | ✓ |
| | FasterMoE [40] | 2022 | Dynamic shadowing method, Novel roofline-like model | B | ✓ |
| | Lina [44] | 2023 | Tensor partitioning, Two-phase scheduling | B | × |
| | Janus [45] | 2023 | Data-centric paradigm, Topology-aware priority strategy | M | × |
| | DeepSpeed-MoE [41] | 2022 | Pyramid-Residual MoE architecture | 10B | ✓ |
| | DeepSpeed-TED [42] | 2023 | 3D hybrid parallel algorithm | 10B | ✓ |
| | SmartMoE [43] | 2023 | Expert placement strateg, Two-stage adaptive auto-parallelization approach | 10B | × |
| Hybrid Parallelism | Alpa [48] | 2022 | Two-level parallel execution, ILP Formulation | 100B+ | ✓ |
| | Smith et al. [46] | 2022 | 3D parallelism methodology | 100B+ | × |
| | Galvatron [49] | 2022 | Decision tree approach, Dynamic programming search algorithm | 10B | ✓ |
| **GPU Memory Optimization** | | | | | |
| **Category** | **Specific Strategy** | **Year** | **Main Features** | **Scales (M, B, 10B, 100B+)** | **Open Resource** |
| Checkpointing and Recomputation | Chen et al. [50] | 2016 | Checkpointing, Recomputation | M | × |
| | Checkmate [51] | 2022 | Integer linear program, Off-the-shelf MILP solvers | M | ✓ |
| Mixed Precision Training | Mixed Precision Training [52] | 2018 | Mixed precision | M | × |
| | Jia et al. [53] | 2018 | LARS algorithm | M | × |
| Memory Swapping | SwapAdvisor [56] | 2020 | Custom-designed genetic algorithm, Memory allocation, Swap decisions | M | × |
| | Autotm [57] | 2020 | Integer linear programming | M | ✓ |
| | FlashNeuron [59] | 2021 | SSDs for data offloading and prefetching, The first buffering-on-SSD solution | B | ✓ |
| | Stronghold [58] | 2022 | Work window method, CPU-GPU offloading | 10B | ✓ |
| | Patrickstar [62] | 2022 | Chunk-based memory management, Dynamic memory management | 10B | ✓ |
| | G10 [61] | 2023 | Amalgamating GPU, host, flash memory into a unified memory space | B | ✓ |
| | DeepUM [60] | 2023 | Enhances Unified Memory by prefetching techniques | B | × |
| Zero Redundancy Optimization | ZeRO [63] | 2020 | Zero memory redundancy | 100B+ | ✓ |
| | ZeRO-Offload [64] | 2021 | Parameters is offloaded to CPU memory | 10B | ✓ |
| | ZeRO-Infinity [65] | 2021 | Parameters is offloaded to CPU, and NVMe memory | 100B+ | ✓ |
| **Communication Optimization** | | | | | |
| **Category** | **Specific Strategy** | **Year** | **Main Features** | **Scales (M, B, 10B, 100B+)** | **Open Resource** |
| Communication Optimization | Bagua [66] | 2021 | MPI-style communication library | M | ✓ |
| | Out-Of-Order BackProp [69] | 2022 | Out-Of-Order Computation | 100B+ | ✓ |
| | ZeRO++ [70] | 2023 | Block-quantization, Data remapping, Quantized gradient averaging paradigm | 100B+ | ✓ |
| | Wang et al. [67] | 2023 | Decomposing the original communication and computational operations | 100B+ | × |
| | Mobius [68] | 2023 | Cross-mapping strategy, Communication-efficient | B | × |
| | Optimus-CC [72] | 2023 | Inter-node communication compression, Selective stage compression | B | × |
| | COCKTAILSGD [71] | 2023 | Random sparsification, Top-K sparsification, Quantization | 10B | × |

Therefore, this section explores the implementation of optimization strategies in foundation model training from three perspectives, network, computing, and storage, to address these challenges, as shown in Table 1.

## A. ADVANCED TECHNIQUES IN PARALLEL COMPUTING

### 1) DATA PARALLELISM: ACCELERATING WORKLOADS EFFECTIVELY

In data parallelism, each computational node possesses a replica of the model and independently processes a subset of data assigned to it. As shown in Fig. 3(a), each node uses its model replica for forward and backward propagation and gradient calculation. So, it requires gradient aggregation and synchronization operations to update the global model parameters. This distributed approach significantly reduces the computational load on individual nodes and speeds up the training process by parallelizing the workload.

Distributed Data Parallel (DDP) [14] utilizes gradient bucketing, computation-communication overlap, and gradient synchronization skipping to enhance the efficiency of distributed data parallelism. In DDP, storing the entire model's parameters on each node simplifies training but significantly increases memory demand, especially for foundation models. To solve this problem, several solutions have been proposed. Facebook introduced a technique called Fully Sharded Data

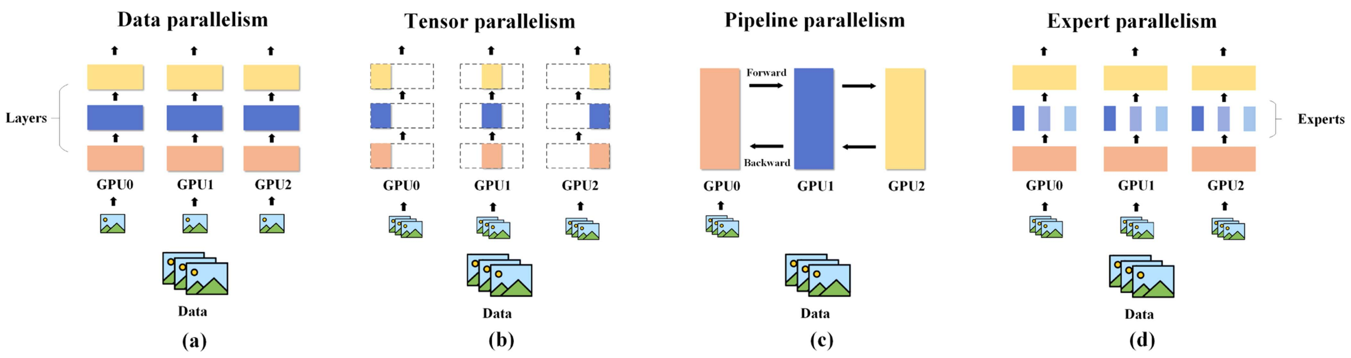

**FIGURE 3.** **Schematic diagram of parallelization strategies in foundation model systems. Different color blocks indicate different layers in the network.**

Parallel (FSDP) [15] to tackle this issue. It divides model parameters into smaller units, restoring the complete model parameters through communication before computation and discarding them immediately after the calculation. Similarly, Xu et al. [16] proposed an automatic cross-replica sharding technique for weight updates in data-parallel training to optimize memory and communication efficiency during model training.

### 2) TENSOR PARALLELISM: SCALING FOUNDATION MODELS

Tensor parallelism is developed to address the challenges of training foundation models that exceed the memory capacity of a single device. In tensor parallelism (As shown in Fig. 3(b)), the parameters and computations of the model are divided (or sliced) across multiple computing devices, effectively reducing the memory load on each device.

Megatron-LM [17] introduced an efficient form of 1D tensor parallelism. For a given computational task, it involves two GEMM operations and a GeLU non-linearity: $Y = \text{GeLU}(XA), Z = YB$. $A$ can be partitioned into $[A_1, A_2\ ]$. So each processor can independently compute the $Y_i$:

$$[Y_1, Y_2] = [\text{GeLU}(XA_1), \text{GeLU}(XA_2)].$$

The second weight matrix $B$ can be split into $\begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$, so $Z$ is equal to $[Y_1, Y_2]\begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$. With this approach, $Y_iB_i$ can be computed separately on individual processors. In Transformer models, the method of 1D tensor parallelism is effectively applied to the computation of multi-head attention. This allows multiple processing units to simultaneously calculate different attention heads without waiting for the results of others.

Optimus [18] proposed an efficient and scalable 2D tensor parallelism. It is introduced based on the scalable universal matrix multiplication algorithm (SUMMA) [19]. Compared to 1D tensor parallelism, 2D parallelism distributes the computational load across more processing units, significantly enhancing overall computational efficiency. Although 2D parallelism offers a more fine-grained model partitioning approach, it can introduce higher communication overhead. To solve this problem, Tesseract [20] introduces the 2.5D tensor parallelism mechanism, which is based on the development of the 2.5D matrix multiplication [21]. Tesseract also develops different parallelization schemes for the matrix multiplication portion and the non-matrix multiplication portion, respectively, which minimize the communication by leveraging additional device requirements, thereby reducing the communication overhead and lowering the memory required per GPU.

To balance computation, memory, and communication loads effectively, 3D tensor parallelism [22] employs a 3-D Parallel Matrix Multiplication algorithm to accurately map and execute the computation process of the Transformer model. This algorithm optimizes the use of computational resources by intelligently distributing and computing different parts of the input and weight matrices on designated processors.

### 3) PIPELINE PARALLELISM: ENHANCING FOUNDATION MODEL SCALABILITY

In pipeline parallelism (Fig. 3(c)), the entire model is divided into several stages, with each part allocated to an independent GPU. However, a typical issue in pipeline parallel processing is the idle time created due to waiting for dependent data or processing results, commonly referred to as the bubble phenomenon. Therefore, effectively reducing these bubbles to enhance GPU utilization in pipeline parallelism becomes a critical issue.

GPipe [23] is one of the first significant works to apply the concept of pipeline parallelism to the training of foundation models. However, GPipe requires waiting for each micro-batch to complete forward propagation before starting backward propagation, as shown in Fig. 4(a). Therefore, the intermediate results (activations) produced during the forward computation of each micro-batch need to be cached in memory for subsequent backpropagation, resulting in increased memory usage. Meanwhile, this approach can also lead to the creation of a significant number of bubbles.

So PipeDream [24] utilizes a one-forward-one-backward (1F1B) strategy, as shown in Fig. 4(b), to solve these problems, in which the backward propagation process immediately follows the completion of forward propagation for a micro-batch. Varuna [25] improves upon PipeDream by performing recomputation earlier during the backward pass, effectively reducing bubbles and memory usage. However, the training mode based on PipeDream introduces two types of parameter

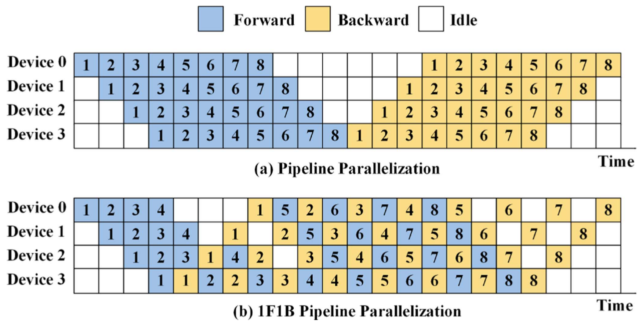
Forward
Backward
Idle
Device 0
Device 1
Device 2
Device 3
(a) Pipeline Parallelization
Time
(b) 1F1B Pipeline Parallelization

**FIGURE 4.** **Schematic diagram of pipeline parallelization and 1F1B pipeline parallelization.**

inconsistency. PipeDream utilizes weight stashing and vertical sync methods to address these issues. Instead, DAPPLE [26] performs synchronization after completing the forward and backward propagation for micro-batches. This synchronization ensures the consistency of model parameters across micro-batches, avoiding parameter inconsistency issues.

PipeDream utilizes a weight storage scheme to use the same weight version in forward and backward propagation for the same input. In the worst case, the number of stored weight versions equals the pipeline depth. Therefore, this can result in increased memory consumption. PipeDream-2BM [27] maintains only two versions of model weights within the pipeline. By storing only two versions, it significantly reduces memory usage. In Megatron-LM [17], pipeline parallelism is implemented using an Interleaved Schedule approach. To reduce pipeline bubbles, the Interleaved Schedule method assigns each device to two sets of model chunks, allowing each device to handle multiple stages. By utilizing the idle time of devices that would otherwise be waiting for backward computation, it can perform forward computation for the second set of model chunks, effectively reducing the size of bubbles in the pipeline. In contrast, Chimera [28] utilizes a bidirectional pipeline by deploying multiple stages of the model on a single GPU. Chimera minimizes idle time and maximizes GPU utilization by interleaving the computation of forward and backward propagation across different stages. Different from Chimera, Hanayo [29] avoids the strategy of model replication. Instead, it employed a wave-like pipeline scheme, further reducing bubble rates and enhancing performance. Similarly employing a bidirectional pipeline, MixPipe [30] achieves a better balance between pipeline and device utilization by adjusting the number of micro-batches. Additionally, MixPipe designs a hybrid scheduling strategy that combines 1F1B and 2F1B to achieve a more balanced memory usage, further decreasing bubble rates. To further reduce bubbles, AvgPipe [31] employs the approach of multiple parallel pipelines, with each pipeline handling a batch of data in each iteration. It trains parallel models using an elastic averaging training method. By processing more batches, AvgPipe can subdivide each batch into finer-grained micro-batches, effectively reducing the generation of bubbles.

In traditional pipelines, feeding a batch of samples with equal lengths into the GPU for training is common. In each batch, padding is applied to the input sequences to accommodate the length of the longest sequence, leading to evident memory wastage. Dynapipe [32] introduces a method of dynamic micro-batching, where the core idea is to ensure consistent sequence lengths among samples within each micro-batch without requiring uniformity across micro-batches. This approach reduces the padding overhead in micro-batch processing, effectively lowering memory consumption. To address memory balancing in the pipeline, Bpipe [33] uses an activation balancing approach that ensures that all GPUs can fully utilize comparable amounts of memory by transferring intermediate activations between different GPUs during training. This innovation solves the problem that some GPUs may face high memory pressure while others fail to fully utilize the performance.

The continuous growth of the foundation's scale has triggered a substantial demand for training resources. In addressing this issue, Bamboo [34] significantly reduces training costs by using preemptible instances optimally. When idle, these instances are available at a lower cost but may be preempted once users submit priority requests. Bamboo optimizes the training pipeline by introducing redundant computations to overcome this challenge. Specifically, each node performs computations not only on its layer but also on adjacent layers. Bamboo cleverly incorporates these additional computations into redundant layers, thus providing greater flexibility at a lower cost. The pipeline methods previously discussed primarily involve a simplistic partitioning of a model's adjacent layers. This approach can lead to imbalanced workload distribution across GPUs. As an improvement, FT-Pipe [35] introduces mixed-pipe partitioning technology. It employs a heuristic algorithm to allocate GPUs based on any computational blocks in the computation graph, not just adjacent layers.

#### 4) EXPERT PARALLELISM: ENHANCING SPECIALIZED COMPUTING CAPABILITIES

Expert parallelism, as depicted in Fig. 3(d), involves segmenting a specific part of a model into several specialized sub-models, referred to as experts, and distributing them across various computational devices. A gating network is used to determine how to allocate input data efficiently among these different experts.

Google's GShard [36] introduces the MoE [37] structure for the first time in training foundation Transformer-based models, aiming to address scalability issues in foundation model training. To optimize the performance of the MoE model, FastMoE [38] was proposed. It is the first high-performance MoE open-source system that supports the PyTorch [39] framework. FastMoE pairs the FFN layer in the Transformer and adopts a finer parallelization strategy. This strategy significantly speeds up the computation of the FFN part of the Transformer model. During the training process of MoE systems, challenges such as dynamic load imbalance and congested end-to-end communication need to be addressed. To tackle these challenges, FasterMoE [40] proposes a dynamic shadowing method to handle load imbalances. By dynamically adjusting task allocation and scheduling, system resources are utilized more evenly, improving overall efficiency. DeepSpeed-MoE [41] achieves significant breakthroughs in model parameter efficiency and performance cost optimization through the novel Pyramid-Residual MoE architecture and model compression techniques. Concurrently, DeepSpeed-TED [42] presents a novel 3D hybrid parallel algorithm that integrates data, tensor, and expert parallelism to further scale the training of MoE models. In addition, it incorporates memory optimization during the optimizer phase and enhances communication efficiency to reduce redundant data transfers.

On the other hand, to address the dynamic computational workload of MoE models, the SmartMoE [43] system introduces a unique expert placement strategy. Building upon the classic combination of parallel strategies, this strategy achieves dynamic load balancing. By intelligently adjusting the deployment positions of various experts in the model, the SmartMoE system effectively balances the computational workload and improves overall system efficiency. In distributed data parallelism, contention may occur between the all-to-all communication among MoEs and the all-reduce operations, leading to prolonged training times. Therefore, Lina [44] integrates tensor partitioning and pipelining to perform micro-operation scheduling, reducing blocking periods in distributed training. All of the above approaches use an expert-centric paradigm, keeping the expert in place and providing information to the expert through an all-to-all process. However, Janus [45] proposes a new data-centric paradigm: maintaining the data in place and moving the experts between GPUs. Janus hides the communication time by scheduling the requests of fetching experts in a fine-grained manner, thus reducing cross-node traffic. Moreover, Janus develops a topology-aware priority strategy, ensuring smooth intra-node expert exchanges without resource contention.

#### 5) HYBRID PARALLELISM: COMBINING THE POWER OF DIFFERENT PARALLEL COMPUTING APPROACHES

Although various parallel technologies have shown significant effects in theoretical and experimental research, a single parallel strategy often fails to meet the growing computational demands and complexity in actual deep learning model training. Therefore, hybrid parallelism becomes critical to addressing this challenge. The core of hybrid parallelism lies in its ability to make customized strategy choices based on the specific requirements of the task and available hardware resources, thereby maximizing training efficiency while ensuring model performance.

Combining multiple parallelization techniques for enhanced efficiency is common when conducting pre-training of foundation models with parameter scales in tens to hundreds of billions. Smith et al. [46] utilized a combination of pipeline and tensor parallelism techniques to parallelize the Transformer block in Megatron-Turing NLG during their training using DeepSpeed [47] and Megatron-LM. They expanded the training scale by incorporating data parallelism, allowing for training on more GPUs. To simplify the application and enhance the efficiency of parallelization strategies, Alpa [48] integrates all parallelization strategies into a single framework, establishing a compiler that automatically generates optimal parallelization strategies. Similarly, Galvatron [49] introduces a decision tree approach that leverages logical intuition for pruning, thereby significantly cutting down the search space. In addition, Galvatron employs a dynamic programming search algorithm to determine the most effective hybrid parallelization strategy.

### *B. GPU MEMORY OPTIMIZATION IN TRAINING*

As the model size increases, the demand for GPU memory grows exponentially. However, limited by hardware resources, insufficient GPU memory often becomes a bottleneck, restricting the scale and performance of the training. Therefore, developing effective GPU memory optimization techniques is essential to reduce memory consumption. Subsequent sections will explore various innovative GPU memory optimization techniques targeting these overheads.

#### 1) CHECKPOINTING AND RECOMPUTATION FOR MEMORY EFFICIENCY

In foundation model training, activation checkpointing technology reduces memory consumption by only saving key activation values and uses recomputation technology to regenerate these values during backpropagation.

It was Chen et al. [50] who first proposed the concept of activation checkpointing to tackle the high memory consumption in foundation model training. By selectively removing unneeded intermediate activations in the forward propagation process and reconstructing them during backward propagation through additional computations, this method significantly reduces GPU memory usage while allowing for the training of more extensive networks. However, recomputation imposes an additional time overhead. Therefore, it requires a trade-off between training time and memory requirements. To address this problem, Jain et al. proposed the Checkmate [51], which models the problem to minimize computation time while ensuring that task scheduling does not exceed the memory limit of the device. The Checkmate effectively manages memory usage by dynamically determining when to store activations and recompute them. This enables the training of larger-scale networks within the constraints of limited memory resources, providing an effective solution to address memory limitations in foundation model training.

#### 2) OPTIMIZING WITH MIXED PRECISION TRAINING

Mixed Precision Training [52] is a technique used in foundation models that simultaneously employs both low-precision and high-precision data types. Representing the training data types as 16-bit floating-point numbers can reduce the amount of computation while lowering the memory requirement. However, the 16-bit floating-point representation will inevitably impact model convergence. Jia et al. [53] utilized the LARS algorithm [54] to solve this problem. The algorithm works by using different learning rates for different layers. However, the test found that applying the LARS algorithm to the training of half-precision models directly caused a great loss of accuracy. This is because after multiplying by the LARS coefficients, many parameters directly go to zero due to the small range of the half-precision values, so Jia et al. converted the half-precision parameters into single-precision and then combined them with the LARS.

#### 3) MEMORY SWAPPING TECHNIQUES IN OPTIMIZATION

The basic idea of memory swapping technology is to offload the computational burden from the GPU to other devices, such as CPUs or NVMe. It migrates some model parameters and computational tasks from the GPU to other devices. This relieves the GPU's workload and enables it to handle the remaining computational tasks more efficiently.

This idea was first introduced in vDNN [55], which aims to reduce the pressure on the GPU memory by moving data that does not require immediate access from the GPU to the CPU memory. The implementation of vDNN represents an initial application of swapping technology. However, with technological advancements, more sophisticated methods have emerged. SwapAdvisor [56] employs genetic algorithms to automatically search for the best data transfer strategy as an alternative to manual judgment-based approaches. The benefit of this automated approach is that it reduces the need for human intervention, thereby increasing efficiency. In contrast, Autotm [57] uses an integer linear programming approach to search for suitable transfer strategies.

Stronghold [58] introduces a work window method, which keeps only part of the model's layers and parameters in the GPU. Under this mechanism, the GPU processes only the model layers within the work window, transferring the rest to the CPU. The corresponding resources are only moved from the CPU to the GPU when the work window shifts. Additionally, Stronghold models the window size, and leverages computation and communication overlap to hide the communication costs between the CPU and GPU effectively. Meanwhile, FlashNeuron [59] considers that offloading data directly to the CPU might interfere with other tasks running on the CPU and thus uses SSDs for data offloading and prefetching. DeepUM [60] enhances Unified Memory (UM) by incorporating prefetching techniques, effectively reducing the additional overhead caused by address translations and page faults. Similarly, G10 [61] innovatively extends the Unified Memory of GPUs, amalgamating GPU memory, host memory, and flash memory into a unified memory space. This fusion is achieved by storing flash memory page addresses in the UM page table. Consequently, a unified page table can point to host, GPU, or flash memory addresses. By preemptively analyzing the lifecycle of tensors, G10 enables efficient tensor swapping when needed, maximizing the overlap between GPU computation and tensor migration. Furthermore, Patrickstar [62] proposes a memory management method based on chunks, a series of consecutive tensors of the same size. This method is similar to storing files in fixed-sized disk blocks in a distributed file system. During training, chunks with different lifecycles can share memory, reducing memory usage. Additionally, Patrickstar collects memory usage information during the warm-up iteration phase to optimize memory management.

Beyond the methods above, other works like ZeRO-Offload and ZeRO-Infinity have also employed Memory Swapping Techniques. To comprehensively introduce the ZeRO series of research, this paper includes these additional works in the next section.

#### 4) ZERO REDUNDANCY OPTIMIZERS

Microsoft has developed a technology called Zero Redundancy Optimization (ZeRO) [63] as the core of the DeepSpeed distributed training framework. The core idea of ZeRO is to reduce the GPU memory by sacrificing some of the communication overhead. ZeRO divides the model parameters, gradients, and optimizer states into multiple parts, with each GPU maintaining only a portion of them during training and obtaining the rest when needed through an All-Gather operation. Building upon the foundation laid by ZeRO, ZeRO-Offload [64] leverages the idea of Heterogeneous DL training [56] to alleviate the pressure on GPU memory by effectively utilizing CPU memory. It divides the model parameters into two parts. One part of the parameters is kept in GPU memory for efficient computation during forward and backward propagation. The other part of the parameters is offloaded to CPU memory and accessed when needed. Further advancing these concepts, ZeRO-Infinity [65], similar to ZeRO-Offload, leverages GPU, CPU, and NVMe memory to enable the training of foundation models on limited resources without the need for code refactoring. With ZeRO-Infinity, the model parameters and gradients are still computed on the GPU, while the optimizer state and activations are offloaded to more suitable NVMe memory and CPU, respectively.

### C. COMMUNICATION OPTIMIZATION

As demonstrated in the previous sections, communication overhead is a significant bottleneck in the distributed training of foundation deep learning models. This issue is especially pronounced when synchronizing model parameters, gradients, and optimizer states across multiple GPUs or nodes. The mainstream solutions focus on reducing the amount of communication, optimizing communication patterns, and enhancing the overlap between computation and communication.

For instance, Gan et al. developed an MPI-style communication library called Bagua [66]. The library provides a series of flexible and modular primitives to support state-of-the-art system relaxation techniques of distributed training. Bagua achieves efficient implementation and scalability through this design for various cutting-edge distributed learning algorithms. The method proposed by Wang et al. [67] involves decomposing the original communication and computational operations into more fine-grained tasks, thereby achieving an overlap between communication and computation that effectively reduces data communication overhead. Mobius [68] introduces a pipeline strategy for heterogeneous memory, which overlaps communication with computation by prefetching data from the CPU to the GPU memory for the next stage. Additionally, it employs a Cross-mapping strategy to reduce communication contention, further optimizing overall performance. Simultaneously, Out-Of-Order BackProp [69] maximizes the overlap between communication and computation by optimizing the sequence of computing output gradients, weight gradients, and parameter updates.

ZeRO achieves parallel computation by distributing model weights, gradients, and optimizer states across multiple GPUs, increasing communication volume and frequency. As an improvement, ZeRO++ [70] employs weight quantization, meaning model parameters are compressed into smaller data types (such as INT8) in real-time before communication, reducing the required communication bandwidth and time. Moreover, ZeRO++ maintains a complete model copy on each machine, enhancing intra-machine communication bandwidth. COCKTAILSGD [71] integrates various communication compression techniques, cleverly overlapping communication with local gradient computation. During the communication steps, it combines three different compression techniques (random sparsification, top-K sparsification, and quantization) to achieve more excellent compression than each method individually. Lastly, Optimus-CC [72] utilizes three techniques: compression of back-propagation gradients, merging of embedding layer synchronization operations, and selective phase compression to reduce inter-node communication volume. Optimus-CC selectively compresses based on the communication needs of different training stages, thus minimizing unnecessary communication overhead and enhancing overall training efficiency.

## IV. MODEL SERVING

This section discusses five principal areas of optimization in foundation model serving systems: batch processing optimization, sparse acceleration techniques, resource scheduling optimization, GPU memory optimization, and multi-model inference (As shown in Table 2). It presents various innovative techniques and technologies designed to enhance processing efficiency, minimize latency, and improve memory usage. These strategies are categorized within the “network-computing-storage” optimization framework.

- *Network optimization* is accomplished through efficient batch processing and resource scheduling, optimizing data flow and task execution.

**TABLE 2. Summary of Optimization Techniques in Foundation Model Serving**

| Method/Framework | Main Features | Year |
|---|---|---|
| ***Batch Processing Optimization*** | | |
| DVABatch [73] | Dynamic Batching | 2022 |
| Orca [74] | Selective Batching | 2022 |
| ***Sparse Acceleration Techniques*** | | |
| SparseAttention [75] | Sparse attention masks | 2023 |
| Deja Vu [76] | Use MLP to predict sparsity | 2023 |
| H2O [77] | Sparse KV Cache | 2023 |
| STI [78] | Sharded Models | 2023 |
| OliVe [79] | Outlier Quantization | 2023 |
| ***Resource Scheduling Optimization*** | | |
| Clockwork [80] | Latency Predict | 2020 |
| DeepSpeed Inference [81] | Integrated Scheduling | 2022 |
| REEF [82] | Real-Time Scheduling | 2022 |
| AlphaServe [83] | Automated Model Parallelism | 2023 |
| FastServe [84] | Preemptive Scheduling | 2023 |
| SHEPHERD [87] | Predictive Load Management | 2023 |
| OSML [88] | Predictive Resource Allocation | 2023 |
| ***GPU Memory Optimization In Inference*** | | |
| Gpulet [89] | Virtual GPU Partitioning | 2022 |
| FlexGen [90] | Zig-Zag Block Scheduling | 2023 |
| DHA [91] | Direct GPU Access | 2023 |
| vLLM [92] | PageAttention Mechanism | 2023 |
| ***Multi-Model Inference*** | | |
| PetS [93] | Selective Model Sharing | 2022 |
| Tabi [95] | Multi-Level Model Inference | 2023 |
| Speculative Decoding [96] | Speculative decoding | 2023 |
| LLMCad [97] | Model collaboration | 2023 |

- *Computing optimization* is characterized by multi-model inference, enabling the efficient utilization of computational resources.
- *Storage optimization* involves GPU memory management and the application of sparse acceleration techniques, collectively reducing memory footprint and computational overhead.

Integrating these “network-computing-storage” principles ensures a comprehensive optimization approach, which is crucial for the performance of foundation model serving systems.

### A. BATCH PROCESSING OPTIMIZATION

Batch processing allows models to handle multiple requests efficiently by grouping input data into batches. This method allows for more efficient use of computational resources by leveraging parallel processing capabilities, significantly improving the throughput and reducing the latency of model inferences. DVABatch [73] proposes a multi-entry and multi-exit strategy that employs operations like *new*, *split*, and *stretch* to dynamically customize batch sizes for different stages of the model, thereby optimizing efficiency, throughput, and reducing latency. In addition to this approach, Orca [74] introduces a selective batching mechanism that strategically applies batch processing and padding to fully connected layers, maximizing efficiency. Simultaneously, it refrains from applying this method to attention layers, minimizing memory overhead. Furthermore, Orca presents an iterative-level scheduling strategy that offers adaptability by enabling batch size adjustments after each processing iteration. The evaluation of Orca on Azure's high-end hardware may not necessarily apply to typical environments with limited resources. On the other hand, DVABatch's testing on NVIDIA Titan RTX GPUs, despite being more accessible, failed to consider multi-GPU configurations.

### B. SPARSE ACCELERATION TECHNIQUES

Sparse acceleration techniques play a crucial role in optimizing the performance of Transformer-based foundation models when faced with limited computational and memory resources. These methods leverage the inherent sparsity in model parameters, attention mechanisms, and KV Cache to prioritize computation and storage. By focusing on the most influential components of the model and reducing the overhead on less critical areas, sparse acceleration approaches enable high model performance while accommodating deployment in resource-constrained environments.

Addressing the computational intensity of self-attention mechanisms in Transformers, Dai et al. [75] introduce a method that leverages the intrinsic sparsity within self-attention matrices. The method applies structured sparse attention masks, refined through entropy-aware fine-tuning, to concentrate computation on crucial attention parameters extracted from the model's attention distribution. These attention masks shape the parameters into patterns such as blocks or stripes, enhancing computational efficiency. Another notable work is Deja Vu [76], which offers a strategic solution by leveraging the concept of contextual sparsity. The proposed approach effectively identifies and activates selective attention heads and FFN parameters that are crucial in processing the given input. The Deja Vu utilizes an MLP to predict the critical attention heads and FFN parameters precisely. In cases where the KV Cache retrieval fails, it triggers a recomputation process. Another relevant work is that the H2O [77] method suggests that in attention blocks, the cumulative attention scores of tokens follow a power-law distribution, with only a few tokens playing a pivotal role in generation. To conserve memory, H2O retains only the KV Cache for these pivotal tokens, which are identified as highly contributive KV Cache based on their elevated cumulative attention scores. This approach substantially reduces the memory requirement while retaining the essential KV Cache of the attention mechanism.

In low-resource scenarios on edge devices, both computation and memory are constrained. STI [78] partitions the model into manageable shards. To balance accuracy and latency, a central scheduler dynamically allocates shards at optimal precision levels, taking into account their importance and resource availability. In the realm of model quantization acceleration, OliVe [79] presents the concept of Outlier-Victim Pair (OVP) quantization. This approach recognizes the importance of outliers for model accuracy, identifying adjacent normal values, termed 'victim values', that can be pruned without significant performance degradation. OliVe strategically retains outliers while selectively pruning adjacent normal values, aligning this approach with hardware design principles. Studies such as Deja Vu demonstrate that models can maintain high accuracy even when subjected to 75% sparsity, indicating that performance is not directly proportional to sparsity levels. Furthermore, the strategic implementation of sparsity, as seen in the STI study, can decrease computational requirements while preserving or even improving model performance. This emphasizes the potential of sparsity in optimizing performance within resource limitations without significantly impacting task-specific results.

### C. RESOURCE SCHEDULING OPTIMIZATION

Effective resource scheduling is essential in optimizing service delivery. To effectively manage variable workloads in GPU-based inference services, several systems incorporate techniques such as multi-GPU optimization, dynamic task distribution, and advanced scheduling algorithms. These methods ensure adherence to Quality of Service (QoS) by minimizing latency through operator fusion, adapting to demand surges through model parallelism, and employing preemptive and adaptive queuing to maintain throughput. DeepSpeed Inference [81] offers a multi-GPU inference solution designed to handle foundation models while adhering to limited GPU memory constraints. DeepSpeed Inference presents a tailored pipeline-parallel schedule for autoregressive decoders, which effectively reduces latency while harnessing the combined power of GPU, CPU, and NVMe storage resources. Additionally, it incorporates operator fusion within Transformer modules to optimize memory utilization and improve overall throughput. AlphaServe [83] utilizes model parallelism in order to distribute the inference of foundation models across multiple GPUs. This approach effectively mitigates memory limitations and reduces latency. Additionally, AlphaServe implements a two-layer algorithm that facilitates efficient cluster distribution and ensures compliance with service-level objective (SLO) requirements. These processes are automated within the Alpha framework. Taking optimization further, FastServe [84] employs preemptive scheduling with a skip-join Multi-Level Feedback Queue to reduce job times and minimize request wait times. This approach efficiently directs jobs to the most suitable queue, avoiding unnecessary transitions and delays. However, despite these sophisticated approaches, Current methods [85], [86] that make decisions for each request individually often result in GPU over-provisioning during short-term high loads, leading to low resource utilization. To address this issue, Shepherd [87] improves predictability by batching individual requests and employs a two-stage algorithm. This algorithm utilizes load data to partition the GPU cluster for service groups and incorporates preemptive scheduling to prioritize large batches that meet the service-level objective (SLO) for optimized throughput.

In scenarios with strong latency requirements, some works have addressed the issue. Clockwork [80] ensures predictable DNN inference times, combating tail latency from diverse tasks, hardware, and inputs to satisfy SLOs and minimize delays. By limiting options at each computational layer for uniform execution times and deploying a central controller that assigns tasks with known durations, Clockwork maintains strict timing assurances. REEF [82] is a system designed for efficient and timely DNN inference on GPUs. It schedules tasks in a way that prioritizes real-time tasks, quickly interrupts other tasks if needed, and allocates computing units first to real-time kernels, then distributes the remaining units to best-effort kernels. And OSML [88] predicts QoS fluctuations by analyzing architectural metrics. It consists of three models: Model A is used to allocate resources and detect scenarios where resources are running low, referred to as "resource cliff" situations. Model B reallocates resources to prioritize QoS-sensitive services, while Model C makes real-time adjustments to allocation to ensure sustainable service performance.

### *D. GPU MEMORY OPTIMIZATION IN INFERENCE*

During the process of inference, the weight parameters of a model significantly consume GPU memory. Various studies have concentrated on optimizing these model parameters. For instance, FlexGen [90] is a throughput-oriented generative inference system that optimizes offloading strategies. A standout characteristic of FlexGen is its zig-zag block scheduling strategy. The zig-zag block scheduling strategy explores the computation graph by advancing column-by-column and reusing weights within each column to minimize loading times. When the memory limits for activations are reached, the process transitions to the next column, optimizing GPU memory utilization and efficiently processing the model through a zig-zag pattern. Additionally, it dynamically loads and unloads activation values and KV Cache as required. In another study, Jeong et al. [91] utilized Direct-Host-Access (DHA) for direct GPU memory access, reducing latency for layers like the embedding layer. They also applied Parallel Model Transmission, dividing the model per GPU for parallel loading via PCIe. The sections are then quickly transferred to the primary GPU using NVLink, optimizing layer execution.

Inference of foundation models faces challenges when it comes to GPU memory limits, particularly due to the increased size of the KV Cache as token counts grow, which can lead to potential memory overflows. To address this issue, frameworks often limit iteration lengths and pre-allocate memory for the KV Cache. However, these measures can result in memory fragmentation and hinder the efficiency of inference. Several techniques have been proposed to optimize this aspect. The PageAttention mechanism proposed by vLLM [92] addresses the issues of GPU memory over-allocation and fragmentation. It accomplishes this by emulating OS page table mapping and segmenting GPU memory into blocks. A block mapping table is then used to ensure logically sequential but physically discrete storage. This dynamic approach effectively meets the demand of the KV Cache, reducing memory fragmentation and improving inference throughput. Drawing inspiration from the virtual nature of operating systems, The gpulet [89] concept introduces an abstraction for partitioning GPUs, creating virtual GPUs that possess a fraction of the physical GPU resources. The proposed multidimensional search-based scheduling framework optimizes GPU tasks by considering data batch sizes along with the temporal and spatial sharing of resources.

### *E. MULTI-MODEL INFERENCE*

Multi-model inference involves utilizing multiple models for serving. An important research question in this context is how to effectively combine these diverse models and optimize resource allocation to achieve optimal performance. PetS [93] introduces a multi-task Parameter Efficient Transformers (PET) framework that fine-tunes a shared core model with task-specific Adapters [94], enabling unified task processing while saving memory and simplifying deployment. In the context of hierarchical models ranging from small to large, one approach is presented by Tabi [95]. It employs well-calibrated confidence scores using temperature scaling to determine whether a query can be promptly resolved using the smaller model or if it should be escalated to the larger model. For escalated queries, Tabi reduces system overhead by employing attention-based word pruning and a weighted ensemble approach. Another technique introduced by Google Research is Speculative Decoding [96], which utilizes a smaller model to generate tokens sequentially while a larger model concurrently verifies the correctness of each token in parallel. This approach allows for the generation of multiple tokens in a single iteration of the larger model. LLMCad [97] differs from Google's Speculative Decoding by employing a tree-based token generation approach that facilitates the concurrent evaluation of multiple tokens. To accomplish this, LLMCad utilizes a smaller language model to construct a comprehensive vocabulary tree comprising various word paths. The larger LLM then efficiently and concurrently evaluates these paths.

In a resource-constrained environment, maintaining performance in multi-model inference necessitates strategic resource management. One approach to achieve this is through the deployment of resource isolation. Another strategy is dynamic allocation, which adjusts resources in real-time based on usage to optimize system efficiency. A critical facet of this strategy is priority scheduling, which entails the dynamic adjustment of resource allocation based on task urgency, business priorities, and other indicators. This ensures that crucial tasks have sufficient resources to meet performance requirements. Additionally, load balancing is employed as an intelligent request distribution mechanism to evenly assign workload across various model instances. This not only prevents individual models from becoming overloaded but also enhances overall resource utilization.

## V. CHALLENGE AND FUTURE DIRECTIONS

❶ *Privacy protection:* Regarding privacy protection, the key challenge for foundation models lies in the potential unauthorized collection, usage, and inadvertent disclosure of personal information. Future efforts should focus on incorporating privacy protection mechanisms into the design and application of models to ensure robust safeguards for user data, preventing unauthorized use and disclosure threats.

❷ *Security:* Foundation models exhibit a relatively weak ability to defend against malicious attacks, making them susceptible to activities such as command injection and prompt injection. Particularly in critical domains such as politics, military, finance, and healthcare, any form of malicious attack could severely affect the stability of national society and the safety of people's lives and property. Therefore, future efforts must focus on enhancing security measures for foundation models to ensure their reliable protection in critical domains.

❸ *Energy sustainability:* Foundation systems face a significant challenge in terms of energy sustainability during both training and serving. This entails a high demand for substantial computational resources, which may result in adverse environmental impacts. The key to future efforts lies in enhancing the energy efficiency of models and adopting more energy-efficient hardware innovations. Through innovative green computing and sustainable development, these efforts aim to make foundation model systems more environmentally friendly and efficient, reducing energy dependence and mitigating environmental impact.

## VI. CONCLUSION

The primary contribution of this paper is that it offers a comprehensive view on the training and serving of foundational model systems, especially through its detailed analysis in the aspects of networking, storage, and computing. In the training section, it discusses various parallel computing strategies. Each strategy has unique advantages and application scenarios. Additionally, it explores GPU memory optimization and communication optimization techniques. The serving section discusses key technologies such as batch processing, sparse acceleration, resource scheduling, GPU memory optimization, and multi-model inference. These strategies are essential for ensuring the efficiency and practicality of the foundation model system in real-world scenarios. In summary, the training and serving of foundation model systems is an evolving field. With the emergence of new technologies, it anticipates solving more challenges and further advancing the field of artificial general intelligence.

## REFERENCES

[1] T. Brown et al., "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 1877–1901.

[2] H. Touvron et al., "Llama: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.

[3] X. Ren et al., "PanGu-$\Sigma$: Towards trillion parameter language model with sparse heterogeneous computing," 2023, *arXiv:2303.10845*.

[4] "PengCheng Mind," *Peng Cheng Laboratory*. [Online]. Available: http://cloudbrain.pcl.ac.cn/

[5] Y. Chang et al., "A survey on evaluation of large language models," *ACM Trans. Intell. Syst. Technol.*, pp. 1–45, 2023.

[6] Hadi et al., "Large language models: A comprehensive survey of its applications, challenges, limitations, and future prospects," *Authorea Preprints*, 2023.

[7] H. Zhao et al., "Explainability for Large Language Models: A Survey," *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 2, pp. 1–38, 2023.

[8] X. Wang et al., "Large-scale multi-modal pre-trained models: A comprehensive survey," *Mach. Intell. Res.*, vol. 20, no. 4, pp. 447–482, 2023.

[9] S. Yin et al., "A survey on multimodal large language models," 2023, *arXiv:2306.13549*.

[10] Y. Zhou et al., "Vision Language Applications: A Survey," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 826–842.

[11] C. Zhou et al., "A comprehensive survey on pretrained foundation models: A history from bert to chatgpt," 2023, *arXiv:2302.09419*.

[12] W. X. Zhao et al., "A survey of large language models," 2023, *arXiv:2303.18223*.

[13] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.

[14] S. Li et al., "PyTorch distributed: Experiences on accelerating data parallel training," 2020, *arXiv:2006.15704*.

[15] Y. Zhao et al., "PyTorch FSDP: Experiences on scaling fully sharded data parallel," in *Proc. VLDB*, 2023, pp. 3848–3860, *arXiv:2304.11277*.

[16] Y. Xu et al., "Automatic cross-replica sharding of weight update in data-parallel training," 2020, *arXiv:2004.13336*.

[17] D. Narayanan et al., "Efficient large-scale language model training on GPU clusters using megatron-Lm," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2021, pp. 1–15.

[18] Q. Xu and Y. You, "An efficient 2D method for training super-large deep learning models," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2023, pp. 222–232.

[19] R. A. Van De Geijn et al., "SUMMA: Scalable universal matrix multiplication algorithm," *Concurrency: Pract. Experience*, vol. 9, no. 4, pp. 255–274, 1997.

[20] B. Wang et al., "Tesseract: Parallelize the tensor parallelism efficiently," in *Proc. Int. Conf. Parallel Process.*, 2022, pp. 1–11.

[21] E. Solomonik et al., "Communication-optimal parallel 2.5 D matrix multiplication and LU factorization algorithms," in *Proc. Eur. Conf. Parallel Process.*, 2011, pp. 90–109.

[22] Z. Bian et al., "Maximizing parallelism in distributed training for huge neural networks," 2021, *arXiv:2105.14450*.

[23] Y. Huang et al., "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019.

[24] D. Narayanan et al., "PipeDream: Generalized pipeline parallelism for DNN training," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, 2019, pp. 1–15.

[25] S. Athlur et al., "Varuna: Scalable, low-cost training of massive deep learning models," in *Proc. 17th Eur. Conf. Comput. Syst.*, 2022, pp. 472–487.

[26] S. Fan et al., "DAPPLE: A pipelined data parallel approach for training large models," in *Proc. 26th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, 2021, pp. 431–445.

[27] D. Narayanan et al., "Memory-efficient pipeline-parallel DNN training," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 7937–7947.

[28] S. Li et al., "Chimera: Efficiently training large-scale neural networks with bidirectional pipelines," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2021, pp. 1–14.

[29] Z. Liu et al., "Hanayo: Harnessing wave-like pipeline parallelism for enhanced large model training efficiency," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2023, pp. 1–13.

[30] W. Zhang, B. Zhou, X. Tang, Z. Wang, and S. Hu, "MixPipe: Efficient bidirectional pipeline parallelism for training large-scale models," in *Proc. 60th ACM/IEEE Des. Automat. Conf.*, 2023, pp. 1–6.

[31] Z. Chen et al., "Elastic Averaging for Efficient Pipelined DNN Training," in *Proc. 28th ACM SIGPLAN Annu. Symp. Princ. Pract. Parallel Program.*, 2023, pp. 380–391.

[32] C. Jiang et al., "DynaPipe: Optimizing multi-task training through dynamic pipelines," in *Proc. EuroSys*, 2024, *arXiv:2311.10418*.

[33] T. Kim et al., "BPIPE: Memory-balanced pipeline parallelism for training large language models," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 16639–16653.

[34] J. Thorpe et al., “Bamboo: Making preemptible instances resilient for affordable training of large DNNs,” in *Proc. 20th USENIX Symp. Netw. Syst. Des. Implementation* 2023, pp. 497–513.

[35] S. Eliad et al., “Fine-tuning giant neural networks on commodity hardware with automatic pipeline model parallelism,” in *Proc. USENIX Annu. Tech. Conf.*, 2021, pp. 381–396.

[36] D. Lepikhin et al., “Gshard: Scaling giant models with conditional computation and automatic sharding,” in *Proc. Int. Conf. Learn. Representations*, 2021, *arXiv:2006.16668*.

[37] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, “Adaptive mixtures of local experts,” *Neural Computation*, vol. 3, no. 1, pp. 79–87, Mar. 1991.

[38] J. He et al., “Fastmoe: A fast mixture-of-expert training system,” 2021,*arXiv:2103.13262*.

[39] A. Paszke et al., “Pytorch: An imperative style, high-performance deep learning library,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2019.

[40] J. He et al., “FasterMoE: Modeling and optimizing training of large-scale dynamic pre-trained models,” in *Proc. 27th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, 2022, pp. 120–134.

[41] Rajbhandari et al., “Deepspeed-Moe: Advancing mixture-of-experts inference and training to power next-generation AI scale,” in *Proc. Int. Conf. Mach. Learn.* 2022, pp. 18332–18346.

[42] S. Singh et al., “A hybrid tensor-expert-data parallelism approach to optimize mixture-of-experts training,” in *Proc. 37th Int. Conf. Supercomputing*, 2023, pp. 203–214.

[43] M. Zhai et al., “SmartMoE: Efficiently training sparsely-activated models through combining offline and online parallelization,” in *Proc. USENIX Annu. Tech. Conf.*, 2023, pp. 961–975.

[44] J. Li et al., “Accelerating distributed MoE training and inference with Lina,” in *Proc. USENIX Annu. Tech. Conf.*, 2023, pp. 945–959.

[45] J. Liu et al., “Janus: A unified distributed training framework for sparse mixture-of-experts models,” in *Proc. ACM SIGCOMM Conf.*, 2023, pp. 486–498.

[46] S. Smith et al., “Using deepspeed and megatron to train megatron-turing NLG 530b, a large-scale generative language model,” 2022, *arXiv:2201.11990*.

[47] J. Rasley et al., “Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters,” in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 3505–3506.

[48] L. Zheng et al., “Alpa: Automating inter-and intra-operator parallelism for distributed deep learning,” in *Proc. 16th USENIX Symp. Operating Syst. Des. Implementation*, 2022, pp. 559–578.

[49] X. Miao et al., “Galvatron: Efficient transformer training over multiple GPUS using automatic parallelism,” in *Proc. VLDB*, 2022, *arXiv:2211.13878*.

[50] T. Chen et al., “Training deep nets with sublinear memory cost,” 2016, *arXiv:1604.06174*.

[51] P. Jain et al., “Checkmate: Breaking the memory wall with optimal tensor rematerialization,” in *Proc. Mach. Learn. Syst.*, 2022, vol. 2, pp. 497–511.

[52] P. Micikevicius et al., “Mixed precision training,” *Proc. Int. Conf. Learn. Representations*, 2018, *arXiv:1710.03740*.

[53] X. Jia et al., “Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes,” 2018, *arXiv:1807.11205*.

[54] Y. You et al., “Imagenet training in minutes,” in *Proc. 47th Int. Conf. Parallel Process.*, 2018, pp. 1–10.

[55] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, “vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design,” in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2016, pp. 1–13.

[56] C. C. Huang et al., “Swapadvisor: Pushing deep learning beyond the GPU memory limit via smart swapping,” in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 1341–1355.

[57] M. Hildebrand et al., “AutoTM: Automatic tensor movement in heterogeneous memory systems using integer linear programming,” in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 875–890.

[58] X. Sun et al., “Stronghold: Fast and affordable billion-scale deep learning model training,” in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2022, pp. 1–17.

[59] J. Bae et al., “FlashNeuron:SSD-enabled large-batch training of very deep neural networks,” in *Proc. 19th USENIX Conf. File Storage Technol.*, 2021, pp. 387–401.

[60] J. Jung et al., “DeepUM: Tensor migration and prefetching in unified memory,” in *Proc. 28th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2023, pp. 207–221.

[61] H. Zhang et al., “G10: Enabling an efficient unified GPU memory and storage architecture with smart tensor migrations,” in *Proc. 56th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2023, pp. 395–410.

[62] J. Fang et al., “Parallel training of pre-trained models via chunk-based dynamic memory management,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 1, pp. 304–315, Jan. 2023.

[63] S. Rajbhandari et al., “Zero: Memory optimizations toward training trillion parameter models,” in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2020, pp. 1–16.

[64] J. Ren et al., “ZeRO-Offload: Democratizing billion-scale model training,” in *Proc. USENIX Annu. Techn. Conf.*, 2021, pp. 551–564.

[65] S. Rajbhandari et al., “Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning,” in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2021, pp. 1–14.

[66] S. Gan et al., “Bagua: Scaling up distributed learning with system relaxations,” 2021, *arXiv:2107.01499*.

[67] S. Wang et al., “Overlap communication with dependent computation via decomposition in large deep learning models,” in *Proc. 28th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2023, pp. 93–106.

[68] Y. Feng et al., “Mobius: Fine tuning large-scale models on commodity GPU servers,” in *Proc. 28th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2023, pp. 489–501.

[69] H. Oh et al., “Out-of-order backprop: An effective scheduling technique for deep learning,” in *Proc. 17th Eur. Conf. Comput. Syst.*, 2022, pp. 435–452.

[70] G. Wang et al., “ZeRO : Extremely efficient collective communication for giant model training,” 2023, *arXiv:2306.10209*.

[71] J. Wang et al., “CocktailSGD: Fine-tuning foundation models over 500 Mbps networks,” in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 36058–36076.

[72] J. Song et al., “Optimus-CC: Efficient large NLP model training with 3D parallelism aware communication compression,” in *Proc. 28th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2023, pp. 560–573.

[73] W. Cui et al., “DVABatch: Diversity-aware multi-entry multi-exit batching for efficient processing of DNN services on GPUs,” in *Proc. USENIX Annu. Techn. Conf.*, 2022, pp. 183–198.

[74] G. I. Yu et al., “Orca: A distributed serving system for transformer-based generative models,” in *Proc. 16th USENIX Symp. Operating Syst. Des. Implementation*, 2022, pp. 521–538.

[75] S. Dai, H. Genc, R. Venkatesan, and B. Khailany, “Efficient transformer inference with statically structured sparse attention,” in *Proc. 60th ACM/IEEE Des. Autom. Conf.*, 2023, pp. 1–6.

[76] Z. Liu et al., “Deja Vu: Contextual sparsity for efficient llms at inference time,” in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 22137–22176.

[77] Z. Zhang et al., “H2O: Heavy-hitter oracle for efficient generative inference of large language models,” in *Proc. Int. Conf. Mach. Learn.*, 2023.

[78] L. Guo et al., “STI: Turbocharge NLP inference at the edge via elastic pipelining,” in *Proc. 28th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2023, pp. 791–803.

[79] C. Guo et al., “OliVe: Accelerating large language models via hardware-friendly outlier-victim pair quantization,” in *Proc. 50th Annu. Int. Symp. Comput. Architecture*, 2023, pp. 1–15.

[80] A. Gujarati et al., “Serving DNNs like clockwork: Performance predictability from the bottom up,” in *Proc. 14th USENIX Symp. Operating Syst. Des. Implementation*, 2020, pp. 443–462.

[81] R. Y. Aminabadi et al., “DeepSpeed-inference: Enabling efficient inference of transformer models at unprecedented scale,” in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2022, pp. 1–15.

[82] M. Han et al., “Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences,” in *Proc. 16th USENIX Symp. Operating Syst. Des. Implementation*, 2022, pp. 539–558.

[83] Z. Li et al., “AlpaServe: Statistical multiplexing with model parallelism for deep learning serving,” in *Proc. 17th USENIX Symp. Operating Syst. Des. Implementation*, 2023, pp. 663–679.

[84] B. Wu et al., “Fast distributed inference serving for large language models,” 2023, *arXiv:2305.05920*.

[85] D. Crankshaw et al., "InferLine: Latency-aware provisioning and scaling for prediction serving pipelines," in *Proc. 11th ACM Symp. Cloud Comput.*, 2020, pp. 477–491.
[86] F. Romero et al., "INFaaS: Automated model-less inference serving," in *Proc. USENIX Annu. Techn. Conf.*, 2021, pp. 397–411.
[87] H. Zhang et al., "SHEPHERD: Serving DNNs in the wild," in *Proc. 20th USENIX Symp. Netw. Syst. Des. Implementation*, 2023, pp. 787–808.
[88] L. Liu et al., "Intelligent resource scheduling for Co-located latency-critical services: A multi-model collaborative learning approach," in *Proc. 21st USENIX Conf. File Storage Technol.*, 2023, pp. 153–166.
[89] S. Choi et al., "Serving heterogeneous machine learning models on multi-GPU servers with spatio-temporal sharing," in *Proc. USENIX Annu. Techn. Conf.*, 2022, pp. 199–216.
[90] Y. Sheng et al., "FlexGen: High-throughput generative inference of large language models with a single GPU," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 31094–31116.
[91] J. Jeong et al., "Fast and efficient model serving using multi-GPUs with direct-host-access," in *Proc. 18th Eur. Conf. Comput. Syst.*, 2023, pp. 249–265.
[92] W. Kwon et al., "Efficient memory management for large language model serving with pagedattention," in *Proc. 29th Symp. Operating Syst. Princ.*, 2023, pp. 611–626.
[93] Z. Zhou et al., "PetS: A unified framework for parameter-efficient transformers serving," in *Proc. USENIX Annu. Techn. Conf.*, 2022, pp. 489–504.
[94] Z. Lin et al., "The adapter-bot: All-in-one controllable conversational model," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 16081–16083.
[95] Y. Wang et al., "Tabi: An efficient multi-level inference system for large language models," in *Proc. 18th Eur. Conf. Comput. Syst.*, 2023, pp. 233–248.
[96] Y. Leviathan et al., "Fast inference from transformers via speculative decoding," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 19274–19286.
[97] D. Xu et al., "LLMCad: Fast and scalable on-device large language model inference," 2023, *arXiv:2309.04255*.

**JIAHANG ZHOU** is currently working toward the Ph.D. degree with the School of Systems Science and Engineering, Sun Yat-Sen University, Guangzhou, China. His research interests include software engineering, artificial Intelligence, and foundation models system software.

**YANYU CHEN** is currently working toward the master's degree with the School of Informatics, Xiamen University, Xiamen, China. His research interests include large language models, knowledge graph, and graph neural networks.

**ZICONG HONG** received the B.Eng. degree in software engineering with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. He is currently working toward the Ph.D. degree with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. His research interests include blockchain, edge/cloud computing, and federated learning.

**WUHUI CHEN** (Member, IEEE) received the bachelor's degree from Northeast University, Shenyang, China, in 2008, and the master's and Ph.D. degrees from The University of Aizu, Aizuwakamatsu, Japan, in 2011 and 2014, respectively. From 2014 to 2016, he was a Research Fellow with the Japan Society for the Promotion of Science, Japan. From 2016 to 2017, he was a Researcher with The University of Aizu. He is currently an Associate Professor with Sun Yat-sen University, Guangzhou, China. His research interests include edge/cloud computing, cloud robotics, and blockchain.

**YUE YU** (Member, IEEE) is currently a Researcher with Pengcheng Lab and an Associate Professor with the College of Computer, National University of Defense Technology, Changsha, China. His research findings have been published on IEEE Transactions on Software Engineering, IEEE Transactions on Instrumentation and Measurement, CHI, CSCW, ICSE, FSE, and ACL. His research interests include cloud computing, software engineering, and artificial Intelligence. He is a Technical Committee Member of OpenI Community.

**TAO ZHANG** received the B.Eng. and M.S. degrees from the Nanjing Institute of Communication Engineering, Nanjing, China, in 1994 and 1997, respectively, and the Ph.D. degree from the PLA University of Science and Technology in 2001. He is currently a Professor with the School of System Science and Engineering, Sun Yat-sen University, Guangzhou, China. His main research interests include network security, system security, and CPS resilience.

**HUI WANG** received the Ph.D. degree in systems engineering from the National University of Defense Technology, Changsha, China, in 2005. He is currently a Researcher with Peng Cheng Laboratory, Shenzhen, China. His main research interests include distributed machine learning, federated learning, computing power networks, NLP, and application.

**CHUANFU ZHANG** received the M.Eng. and Ph.D. degrees in mechatronics engineering and automation from the National University of Defense Technology, Changsha, China, in 2002 and 2011, respectively. He is currently an Associate Professor with the School of Systems Science and Engineering, Sun Yat-sen University, Guangzhou, China. His research interests include modern cryptography application technology, information security modeling, and simulation theory.

**ZIBIN ZHENG** (Senior Member, IEEE) received the Ph.D. degree from the Chinese University of Hong Kong, Hong Kong in 2011. He is currently a Professor with the School of Computer Science and Engineering, Sun Yat-sen University, China. He has authored or coauthored more than 300 international journal and conference papers, including nine ESI highly cited papers. His research interests include blockchain, artificial intelligence, and software reliability. He was the recipient of several awards, including the Top 50 Influential Papers in Blockchain of 2018, ACM SIGSOFT Distinguished Paper Award at ICSE2010, and Best Student Paper Award at ICWS2010.