IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 10 June 2024, accepted 2 July 2024, date of publication 8 July 2024, date of current version 19 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3424945

SURVEY

Achieving Peak Performance for Large Language Models: A Systematic Review

ZHYAR RZGAR K. ROSTAM[1], (Member, IEEE), SÁNDOR SZÉNÁSI[2,3], (Member, IEEE), AND GÁBOR KERTÉSZ[2,4], (Senior Member, IEEE)

[1]Doctoral School of Applied Informatics and Applied Mathematics, Óbuda University, 1034 Budapest, Hungary
[2]John von Neumann Faculty of Informatics, Óbuda University, 1034 Budapest, Hungary
[3]Faculty of Economics and Informatics, J. Selye University, 945 01 Komárno, Slovakia
[4]Laboratory of Parallel and Distributed Systems, Institute for Computer Science and Control (SZTAKI), Hungarian Research Network (HUN-REN), 1111 Budapest, Hungary

Corresponding author: Zhyar Rzgar K. Rostam (zhyar.rostam@stud.uni-obuda.hu)

ABSTRACT In recent years, large language models (LLMs) have achieved remarkable success in natural language processing (NLP). LLMs require an extreme amount of parameters to attain high performance. As models grow into the trillion-parameter range, computational and memory costs increase significantly. This makes it difficult for many researchers to access the resources needed to train or apply these models. Optimizing LLM performance involves two main approaches: fine-tuning pre-trained models for specific tasks to achieve state-of-the-art performance, and reducing costs or improving training time while maintaining similar performance. This paper presents a systematic literature review (SLR) following the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) statement. We reviewed 65 publications out of 983 from 2017 to December 2023, retrieved from 5 databases. The study presents methods to optimize and accelerate LLMs while achieving cutting-edge results without sacrificing accuracy. We begin with an overview of the development of language modeling, followed by a detailed explanation of commonly used frameworks and libraries, and a taxonomy for improving and speeding up LLMs based on three classes: LLM training, LLM inference, and system serving. We then delve into recent optimization and acceleration strategies such as training optimization, hardware optimization, scalability and reliability, accompanied by the taxonomy and categorization of these strategies. Finally, we provide an in-depth comparison of each class and strategy, with two case studies on optimizing model training and enhancing inference efficiency. These case studies showcase practical approaches to address LLM resource limitations while maintaining performance.

INDEX TERMS Distributed training, GPU acceleration, large language model, LLM, LLM acceleration, LLM frameworks, LLM optimization.

I. INTRODUCTION

In recent years, dense deep learning models have seen an extraordinary growth in the number of parameters [1], [2], [3]. Transformer as an effective deep learning architecture has been widely used over the recent years, and transformer-based models have achieved notable success and recognition in various fields including language modeling compared to the existing models [4], [5], [6], [7], [8], [9], [10], [11], [12], [13].

To achieve significant accuracy in deep learning, large models with billions to trillions of parameters are essential. Therefore, deep learning models continue to grow in complexity with an array of large-scale models ranging from Bidirectional Encoder Representations from Transformers ($BERT_{large}$, 340 million parameters) [8], Generative Pre-trained Transformer-3 (GPT-3, 175 billion parameters) [14], to General Language Model (GLM-3, 1.75 trillion parameters) [15]. With models now reaching trillions of

The associate editor coordinating the review of this manuscript and approving it for publication was Agostino Forestiero.

parameters, even the most powerful GPUs are struggling to keep up [1]. This resource-intensive requirement is making it difficult for many researchers to access the computational resources they need to train these models [1], [4], [16]. Also, handling, managing, and fitting these models into device memory is a daunting challenge due to memory limitations, and this tremendous size of data brings complexity, and requires high-end computing resources with significant memory requirements to process [5], [17], [18], [19]. Training large-scale models effectively require significant adjustments [20], [21], [22], [23], [24], especially in terms of increasing training throughput and loading these kinds of large models into GPU memory [18].

As a result, developing frameworks, libraries and proposing new techniques to overcome the mentioned challenges has become an essential task. There are many studies that have worked on possibilities for optimization and acceleration with large models and using various techniques to achieve state-of-the-art (SOTA) results without sacrificing accuracy. These remarkable advancements in the field of language models (LMs) required a systematic review of recent LM optimization and acceleration techniques. To address these challenges and guide future research, this SLR paper aims to:

- Analyze recent optimization and acceleration techniques for LLMs.
- Identify challenges associated with training, inference, and system serving for LLMs (billions/trillions of parameters).
- Develop a structured taxonomy to categorize LLM optimization techniques.
- Review and evaluate recent libraries and frameworks designed for LLM optimization.
- Identify promising areas for future research in LLM development, focusing on efficiency, scalability, and flexibility.

In this SLR we are making the following contributions:

- Comprehensive overview: We offer a comprehensive overview of the development of language modeling (Section II), detailing commonly used frameworks and libraries (Section IV), and recently used techniques and strategies (Sections V, VI, VII). This serves as a valuable resource for understanding the current landscape of LLM optimization.
- Taxonomy of optimization strategies: We categorize optimization strategies into three classes: training optimization, hardware optimization, and scalability and reliability. This taxonomy helps clarify the various approaches and their specific applications (presented in Fig 4, Sections V, VI, VII).
- Detailed analysis of techniques: Our analysis explores recent optimization and acceleration strategies, we provide two comparative analyses regarding performance, cost, and scalability for reviewed strategies (presented in Tables 6 and 7) and their core categories: training optimization, hardware optimization, and scalability and reliability (presented in Table 5). In the latter analysis, we also consider the focus of classes.
- Case studies: We include two in-depth case studies that demonstrate practical approaches to optimizing model training and enhancing inference efficiency. These case studies highlight how resource limitations can be addressed while maintaining performance (Sections VIII-A, VIII-B).
- Future direction: We explore a range of promising future directions for LLM development. These areas, detailed in specific sections, focus on enhancing efficiency, scalability, and flexibility for LLMs (Section X).

This review paper is organized as follows: an overview of language modeling development (Section II), followed by an in-depth explanation of the most commonly utilized frameworks and libraries specifically designed for optimizing and accelerating large language models (LLMs) (Section IV, Tables 3 and 4), accompanied by taxonomy and categorization. Additionally, it delves into recent optimization and acceleration strategies employed within LLMs, including the taxonomy and categorization of these strategies (presented in Fig. 1) (Section V, VI, VII), Table 8 summarizes the reviewed papers, excluding those already covered in Tables 3 and 4 or the main text. Moreover, we present an individual comparison in terms of performance, cost, and scalability for reviewed strategies discussed in Tables 6, and 7, and the classes (training optimization, hardware optimization, scalability and reliability) presented in Table 5. In addition to the mentioned factors, we consider the classes' focus in this comparison. Finally, we illustrate these concepts with two real-world examples: optimizing model training and improving inference efficiency through case studies (Section VIII).

A. RELATED WORKS

In this section, we will present the related studies that investigate optimization and acceleration with dense deep learning models and LLMs. Jahan et al., in [25] present a systematic literature review (SLR) by comparing 31 language models inspired by BERT, published between 2018 and 2020, to help researchers choose the best model based on their requirements. By analyzing each model's performance against RoBERTa, the study identified seven models that performed better, and the rest of the studies investigated with different parameter settings. The outperforming models varied in dataset size, suggesting that both large and small datasets can be effective depending on the model's architecture. Ultimately, this research provides valuable insights for researchers seeking the optimal language model for their specific tasks. Yu et al. [26] conduct a survey that explores the growing challenges and opportunities for optimizing large-scale deep learning systems. By highlighting recent advances in optimization techniques, it proposes a new way to categorize and explain the different computing approaches used. Zhao et al. [24] carry out a survey that focuses on the recent advancements in LLMs. The study concentrates

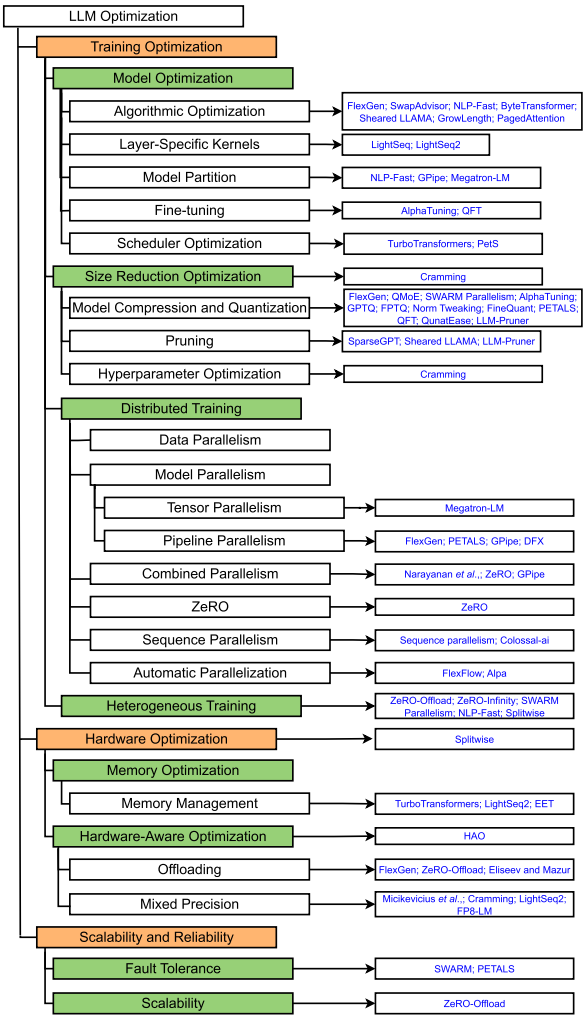

FIGURE 1. LLM optimization techniques and taxonomy.

on four major dimensions of LLMs: pre-training, adaptation tuning, utilization, and capacity evaluation. The survey emphasizes the techniques or discoveries essential for LLMs' success. Additionally, it provides an overview of available development resources and offers valuable guidelines for successful LLM implementation, drawing from the latest research. Bai et al.., in [27] provide a systematic survey that provides an overview of LLM resource efficiency. It focuses on LLM significant resource consumption in computational, memory, energy, and financial aspects. It categorizes techniques aimed at improving LLMs' resource efficiency. Standardized evaluation metrics and datasets are also proposed to facilitate fair comparisons. The survey offers insights into current advancements and guides future developments toward more sustainable and efficient LLMs. Wang et al. [28] explore new methods to achieve comparable accuracy while reducing training costs. They highlight optimized algorithms for quicker learning, distributed architectures leveraging widespread computing resources, and hardware acceleration with communication optimization for collaborative training. While challenges remain, these advancements pave the way for more affordable and accessible AI in the future. Min et al. [29] present a survey that explored recent studies for using powerful pre-trained language models (PLMs) in natural language processing (NLP) tasks through the analysis of three popular approaches. The first approach trains on a massive dataset for general language understanding, then specializes it for a specific task with focused training. The second approach prompts the PLM to treat the desired task as similar to its pre-training tasks, allowing for efficient "few-shot" learning with just a few examples. The third approach modifies NLP tasks as text generation to maximize the utilization of knowledge embedded within a generative language model. Qiu et al. [2] provide a comprehensive overview of pre-trained models (PTMs), from fundamental knowledge and model architectures to diverse pre-training tasks, extensions, and real-world applications. It proposes a clear taxonomy for easy navigation and provides abundant resources like code, tools, corpora, and reading lists. Recognizing current limitations, the survey also presents a discussion on promising future directions to shape the NLP landscape. This survey [30] explores techniques for building efficient LLMs. It categorizes approaches into three groups: Model-centric, Data-centric, and LLM frameworks. In the model-centric method focuses on optimizing the LLMs through techniques including compression, efficient training, and specialized architectures. Data-centric focusing on improving data quality and using prompts to guide the model efficiently. LLM frameworks create specialized software to handle LLMs, the survey aims to provide a comprehensive understanding of how to make LLMs more efficient and accessible.

In this SLR, we examine research published between 2017 and December 2023, filling a gap in existing surveys by specifically focusing on optimizing and speeding up LLMs. Following the PRISMA approach, we reviewed 65 articles. The study starts with an overview of language modeling's development and then dives deep into the most popular frameworks and libraries for optimizing and accelerating LLMs. It organizes these models with a clear taxonomy and categorizes them effectively. The research also investigates recent approaches for optimizing and accelerating LLMs, offering a classification system along with a summary and comparison of the reviewed papers containing the latest optimization techniques. Moreover, resource limitations and the impact of various optimization techniques in LLMs were addressed through two in-depth case studies. These studies delve into practical approaches for optimizing training and enhancing inference efficiency, demonstrating how these techniques can be applied effectively without excessive resources.

B. RESEARCH METHODOLOGY

In this study, we have followed the PRISMA statement to ensure a systematic and transparent methodology. PRISMA provides a comprehensive set of guidelines for conducting

TABLE 1. Research queries executed.

No.	Research Query
RQ 1	LLM GPU Acceleration
RQ 2	LLM GPU Optimization
RQ 3	LLM Acceleration
RQ 4	LLM Optimization [1]
RQ 5	Large Language Model GPU Acceleration
RQ 6	Large Language Model GPU Optimization

systematic reviews. Our approach included a detailed search strategy across multiple databases, explicit inclusion and exclusion criteria, and a thorough study selection process. We documented each step meticulously, including the study selection and exclusion procedures. (presented in Fig. 2).

Eligibility Criteria: This review will focus on the optimization and acceleration of LLMs, examining the most recent and widely utilized libraries, frameworks, and techniques in this field. To ensure focused analysis, strict eligibility criteria are applied. Only studies published between 2017 and December 2023 are considered, excluding publications not written in English and retracted papers. Additionally, studies are excluded if they are irrelevant to our SLR, or do not explicitly address ''Optimization'' ''Acceleration,'' or ''Large Language Models'' in their titles, abstracts, or keywords.

Information Sources: To ensure a comprehensive search for authentic studies, a variety of sources, including databases, websites, and tools, were employed. Digital libraries like IEEE Xplore, Web of Science, and Scopus alongside open access libraries like arXiv and dedicated tools like Zotero facilitated the data collection and reference management. The last search was conducted on May 25th, 2024. Additionally, Rayyan and the researchrabbit.ai websites were utilized for data exploration and study selection.

Search Strategy: This systematic review leveraged two web-based AI tools, ResearchRabbit [31], and Rayyan [32], for both data collection and study selection. In all databases and websites, we were particularly interested in finding studies that focused on language modeling, particularly those that focused on LLM optimization and acceleration. We employed various queries in each source (see Table 1) and exported the retrieved studies for import into Rayyan. Rayyan's AI capabilities facilitated both the selection of desired studies and the exclusion of irrelevant ones.

Selection Process: The process of selecting which works to review in this study employed strict inclusion criteria. In this SLR we explore the techniques and methods that were primarily examined based on their focus on large-scale language modeling, including transformer-based models such as PLMs, LLMs, and even general NLP models. The Rayyan platform facilitated the selection process. Two stages were involved: initial screening using eligible and inclusion criteria, followed by author selection of the most relevant and impactful studies. Finally, the ''compute rating'' function in Rayyan was used, and the authors double-checked excluded studies for accuracy.

Data Extraction: In this stage, we focused on extracting relevant data from selected studies. Our aim was to collect information on two key aspects:

Outcomes: We were particularly interested in outcomes related to LLM optimization and acceleration. Specifically, we sought data on:

- Performance metrics: This could include metrics like perplexity [19], BLEU score, ROUGE score [33], or task-specific accuracy measures depending on the study's focus.
- Training time reduction: We looked for data on how different techniques impacted the time required to train LLMs.
- Resource usage: If studies reported resource (memory) usage changes with different optimization techniques, we collected that data.

We aimed to collect all relevant results within these outcome domains whenever possible. This included considering data from different measures, time points, and analyses reported by the study authors.

Additional Variables: In addition to the main outcomes, we also extracted data on the following aspects of the studies:

- LLM architecture: The specific type of LLM architecture used in the study.
- Optimization techniques: Detailed description of the optimization techniques employed in the study.
- Hardware/Software platforms: The hardware and software platforms used for training, inference, serving, and evaluation.

Data Collection Process: ResearchRabbit is a web-based tool powered by AI that helps and guides researchers to find relevant studies in a variety of digital libraries and allows researchers to export retrieved results in a collection to reference managers tools (similar to Zotero). ResearchRabbit's search is powered by SemanticScholar and shows only the top 50 search results for a single query, aiming to maintain the research focus effectively [31]. Initially, we applied our queries to the ResearchRabbit website and then added the most relevant retrieved results to our collection. Following that, we applied the same queries in digital libraries like IEEE Xplore, Web of Science, Scopus, and arXiv (see Table 2). The papers were reviewed on a case-by-case basis. Then, a precise summary of each paper was written. Finally, the interesting data that directly addressed the issues the papers attempted to address were extracted from the summaries.

Study Risk of Bias Assessment: In this SLR, we followed a meticulous process to assess the risk of bias in the included studies, adhering to best practices for ensuring the reliability and validity of our findings.

Automation Tools:

- We utilized Rayyan, an AI-powered tool, to facilitate the initial screening and selection process. Rayyan's AI

[1]The initial search query in arXiv with this RQ was broad, returning 440 studies, many irrelevant to our research. To refine the results and minimize the risk of bias, also ensure retrieval of high-quality, relevant papers, we employed the AND operator along with the title field within the search query specifically on arXiv.

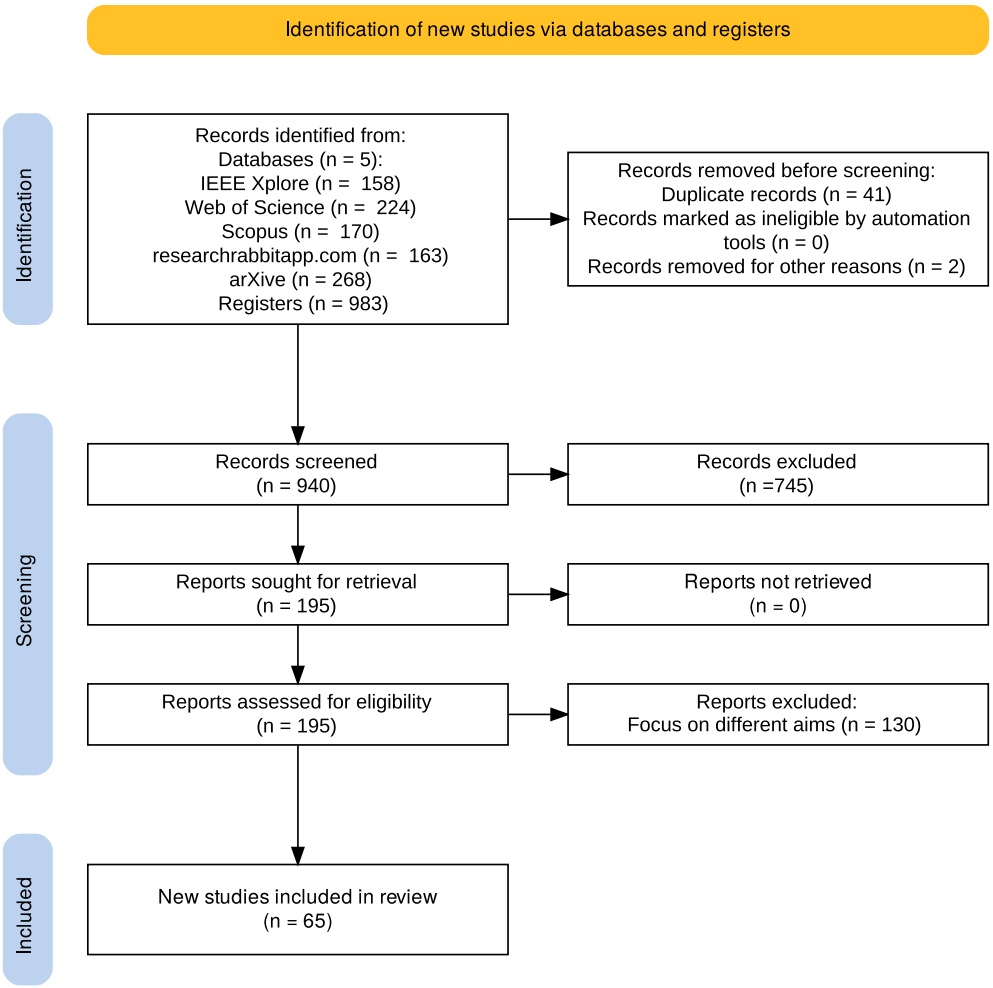

FIGURE 2. The PRISMA 2020 flow diagram of the performed search.

capabilities helped in identifying potential biases and categorizing studies based on relevance and quality.

- ResearchRabbit was used for gathering relevant studies, which provided a focused list of top search results, aiding in maintaining the research scope effectively.

Reviewer Process:

- Each study was assessed by three independent reviewers. This approach helps to minimize subjective bias and ensures a more balanced evaluation.
- The reviewers independently examined each study based on predefined criteria including selection bias, performance bias, detection bias, attrition bias, and reporting bias.

Independent Review and Consensus:

- The reviewers worked independently during the initial assessment phase to ensure unbiased evaluations.
- After the independent assessments, the reviewers compared their findings. Any discrepancies or disagreements were resolved through discussion and consensus.

We adhered to a rigorous and systematic approach to assess the risk of bias, which involved multiple independent reviewers and the use of validated tools. Automation tools such as Rayyan and ResearchRabbit played a crucial role in streamlining the screening and selection process, thereby enhancing the efficiency and accuracy of our assessments. By combining independent reviews, consensus discussions, and advanced AI tools, we ensured a robust and unbiased evaluation of the included studies.

Synthesis Methods: To enable a comprehensive and insightful analysis of LLM optimization techniques across diverse contexts, a three-tiered categorization scheme will be employed. The initial categorization will consist of group studies based on the utilized LLM libraries/frameworks and the optimization techniques investigated. Subgroups within these categories will be further established based on the specific type of LLM or the NLP task addressed by the studies. This method enables a highly detailed examination of how the effectiveness of optimization techniques varies across different LLM and NLP task configurations. Additionally, key findings from each individual study will be summarized in tables, including details like the optimization technique used, LLM type, NLP task addressed, achieved performance

TABLE 2. Studies retrieved per database / search engine.

Database / Search Engine	Total
IEEE Xplore	158
Web of Science	224
Scopus	170
ResearchRabbit	163
arXiv	268
Total	983

metrics, and the study's aims. Finally, a narrative synthesis will be conducted to analyze recurring themes across the studies. This thematic analysis will focus on the effectiveness of LLM libraries and optimization techniques in achieving performance improvements while considering resource constraints. It will also explore potential explanations for observed variations in effectiveness, with particular attention paid to factors like LLM size, resources used, and the NLP task addressed.

Reporting Bias Assessment and Certainty Assessment: To minimize the risk of bias in our systematic review, we implemented a multifaceted strategy. First, to address reporting bias, we utilized Rayyan and ResearchRabbit, as AI-powered tools, during the initial screening and selection process. These tools can categorize studies based on relevance and quality and can help flag studies with characteristics suggestive of reporting bias, such as those focusing solely on positive outcomes. Second, to strengthen the certainty of our findings and minimize subjective bias, we implemented a multi-reviewer approach. Each study underwent independent assessment by three reviewers based on predefined criteria. This approach ensures a more balanced evaluation and reduces the influence of individual reviewer bias.

II. LANGUAGE MODELING DEVELOPMENT

Language modeling is a fundamental approach to enhancing the ability of machines to understand and process human language. It is a computational model that can learn and predict the possibilities of incoming (or missing) tokens [24]. The development of language models can be classified as follows (see Fig. 3):

- *N-gram language models,* like bigrams and trigrams, are basic methods that learn from the frequency of word sequences in text [34], [35]. However, their limited context window restricts their ability to capture long-range dependencies and understand the deeper semantic relationships between words.
- *Markov assumption language models,* refers to those models that predict the next word based on the most recent in the context [24]. Both n-gram and Markov assumption language models are commonly used to improve task performance in NLP, and information retrieval (IR) [36].
- *Machine learning models,* these models investigate machine learning algorithms to enhance language comprehension. They are trained on extensive text corpora to discern patterns and relationships [37]. The adoption of machine learning in NLP introduced a more advanced methodology, enabling the creation of applications such as spam detection [38] and sentiment analysis [39].
- *Neural language models,* these models are developed based on NN for working with a sequence of data. They have a special ability of learning effective features for words or sentences. These studies [40], [41], [42] have initiated the use of language models for representation learning (beyond word sequence modeling), and show that these models have an important impact on the field of NLP [24], [43].
- *Transformer language models* refer to those models that leverage the capabilities of a deep learning architecture called Transformer to process and understand human language [44], [45]. These models achieved remarkable results by using *"special attention mechanism"* to understand the relationship between words and sentences. These models capture context-aware representation instead of learning fixed word representations, first pre-training then fine-tuning according to specific downstream tasks [2], [8], [24], [46]. Transformer architecture has been used to build PLMs such as BERT [8], GPT-2 [47], and BART [48]. These models underwent training using bidirectional language models and specifically designed pre-training tasks applied to extensive unlabeled datasets. The growth in model size and data size has revolutionized the way we approach downstream tasks, enabling large-sized PLMs to achieve remarkable performance gains. These models exhibit unique characteristics compared to smaller PLMs, such as 330M-BERT and 1.5B-GPT-2, demonstrating exceptional abilities in solving complex tasks. As a result, LLM is the term used to refer to large-sized PLMs [46], [49], [50].

III. MACHINE LEARNING MODELS

The process of building, deploying, and managing a machine learning model involves three distinct phases: *training, inference, and system serving*. Training is the foundation of machine learning, where a vast dataset of labeled data is used to develop a model that can identify patterns and relationships within the data. Inference is the application of the trained model, where new, unseen data is fed into the model to obtain predictions or classifications based on the learned patterns. System serving ensures the model's longevity and effectiveness in real-world applications, handling large volumes of requests, monitoring the model's performance, and providing continuous updates or modifications as needed [11], [19], [51]. In the section IV, we provide a categorization of the most recent frameworks and libraries utilized for LLMs optimization, structured into three primary classes: training, inference, and deployment and system serving (presented in Fig. 4). However, certain studies can

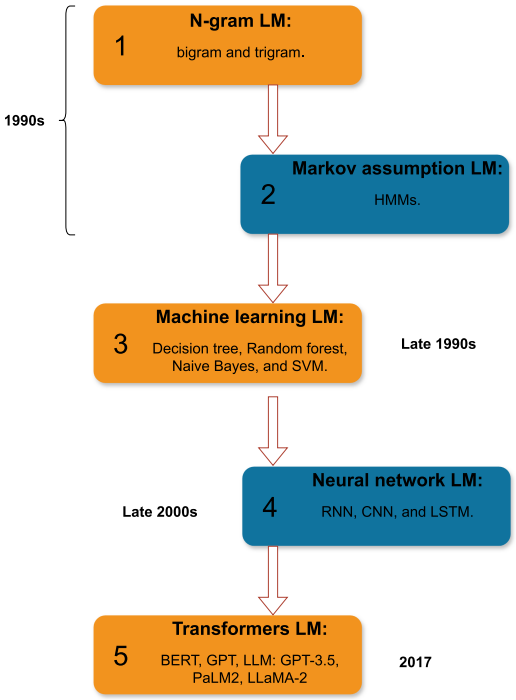

FIGURE 3. Language model development.

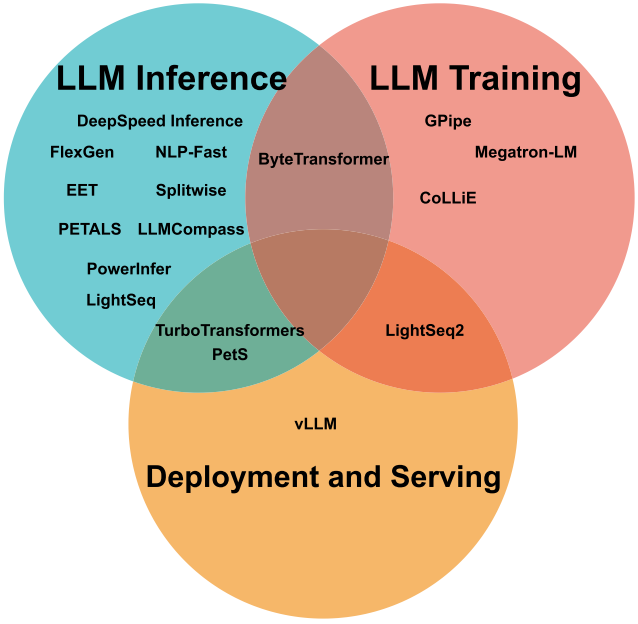

FIGURE 4. LLM frameworks and libraries.

be classified into two categories simultaneously, owing to their ability to handle multiple tasks, such as LightSeq2 (section IV-A4), TurboTransformers (section IV-B4), and PetS (section IV-B5).

IV. FRAMEWORKS AND LIBRARIES

As most LLMs are designed based on Transformers, these models are a powerful type of neural network that have achieved SOTA results on a wide range of applications. To achieve these results the models are required to have a huge model size with hundreds of billions, even trillion of parameters. Training LLMs requires distributed training algorithms, which employ parallel processing techniques to efficiently train these massive models. To streamline distributed training, various optimization frameworks have been developed, providing tools and infrastructure for implementing and deploying parallel algorithms [24], [52], [54]. In this section, we will provide the most recent frameworks and libraries designed to overcome those limitations.

A. LLM TRAINING FRAMEWORKS AND LIBRARIES

This section will delve into the objectives and outcomes of LLM frameworks and libraries employed in the training phase. Additionally, a summary of each framework/library will be provided individually (see Table 3).

1) GPIPE

GPipe [3] introduces a novel pipeline parallelism framework based on batch partitioning. It divides each mini-batch example into smaller micro-batches, which are subsequently executed in sequence across the cells. During training, it employs synchronous mini-batch gradient descent, where gradients from all micro-batches are aggregated and applied to the model at the end of the mini-batch. GPipe has been shown to train two large-scale models: a convolutional model for image classification and a transformer model for machine translation. The convolutional model, AmoebaNet, was trained on 480 $\times$ 480 input from the ImageNet 2012 dataset. To enhance its performance, the model width was expanded, and its parameters were scaled up to 557 million. The model achieved a top-1 validation accuracy of 84.4%. Meanwhile, the transformer model a single 128-layer, 6-billion-parameter multilingual model trained across 103 languages, was also evaluated. GPipe achieved superior performance compared to training 350-million-parameter bilingual transformer big models individually across 100 language pairs. The model presents its efficiency by boosting the performance on a variety of devices, with the support of flexibility on any deep network architectures, utilizing the synchronous gradient decent, and ensuring consistent training regardless of the number of partitions.

2) BYTETRANSFORMER

ByteTransformer [4] is a transformer framework for GPU acceleration with an efficient and high performance optimized for variable-length inputs in NLP problems. The framework uses an algorithm for overcoming the redundant computations on zero-padding tokens, and variable input length. Furthermore, the model proposed a fused Multi-Head Attention (MHA) to reduce the memory overhead of the intermediate matrix. This model manually optimizes the memory sizes of layer normalization by introducing bias and activation to maximize the overall system performance.

TABLE 3. Summary of LLM training frameworks and libraries.

Studies	Aims	Outcomes
GPipe [3]	A new **pipeline parallelism library** based on batch splitting. It is efficient, task independent, and works with different NN architectures.	Investigated single 6-billion-parameter, 128-layer Transformer model on a dataset with 103 languages achieved better results than individually training 350-million-parameter.
ByteTransformer [4]	A transformer **framework for GPU acceleration**, optimized for **variable-length inputs** in NLP problems.	Evaluated on an NVIDIA A100 GPU, for BERT-like transformers. Boosted fused MHA by 6.13× compared to PyTorch attention. Also, outperformed PyTorch, TensorFlow, Tencent TurboTransformer, Microsoft DeepSpeed, and NVIDIA FasterTransformer by 87%, 131%, 138%, 74%, and 55%, respectively.
Megatron-LM [19]	A deep learning **library for training LLMs with billions of parameters.** Offers a set of optimization methods for distributed training.	With 8.3 billion parameters trained on 512 NVIDIA V100 GPUs achieved 15.1 PetaFLOPs throughput. Also, achieving a perplexity of 10.8 on the WikiText103 benchmark and an accuracy of 66.5% on the LAMBADA dataset.
LightSeq2 [52]	Software **library that accelerates the training of transformer-based models within GPUs**.	Achieves significant speedups on a variety of NLP tasks. speedup of 308% on the WMT14 English-German machine translation task compared to PyTorch.
CoLLiE [53]	A **library for collaborative training of massive LMs**. It explores memory usage and throughput under different optimization methods, and investigates training techniques to improve the ability of a LM (LLaMA-65B) to follow user instructions.	Improves training efficiency for LLMs. Techniques like LoRA and AdaLomo specifically helped a large model (LLaMA-65B) follow instructions better, with an average score of 56.9, all without sacrificing overall performance.

Bold text in the "Aims" column indicates the framework's primary area of specialization or the range of tasks it is designed to address.

It has been used by some famous applications including TikTok and Douying of ByteDance. The model was evaluated on an NVIDIA A100, focusing on the forward pass of BERT-like transformers, including BERT [8], ALBERT [55], DistilBERT, and DeBERTa. It showcased a significant improvement, enhancing the fused MHA mechanism by 6.13× compared to PyTorch attention. Additionally, ByteTransformer outperformed PyTorch, TensorFlow, Tencent TurboTransformer [11], Microsoft DeepSpeed [5], and NVIDIA FasterTransformer by 87%, 131%, 138%, 74%, and 55%, respectively, in terms of the end-to-end performance of a standard BERT transformer.

3) MEGATRON-LM

Megatron-LM [19] is a deep learning library for training LLMs efficiently and effectively. It enables the library for the training of very large transformer models with billions of parameters. It offers a set of optimization methods for distributed training, it includes strategies like intra-layer model parallelism, and mixed-precision training. These optimization techniques significantly enhance training efficiency and speed, facilitating effective distributed training across multiple GPUs. Megatron-LM operates independently without requiring new compiler or library changes. This makes it orthogonal and complementary to pipeline model parallelism, allowing for seamless integration and flexibility within existing NLP frameworks.

The library has been shown to be highly effective for training LLMs. A Megatron-LM model with 8.3 billion parameters was trained on 512 NVIDIA V100 GPUs using 8-way model parallelism and achieved sustained performance of up to 15.1 PetaFLOPs across the entire application. This is significantly faster than previous approaches to training LLMs. Additionally, it has been shown to achieve SOTA results on several NLP benchmarks. A Megatron-LM model with 8.3 billion parameters achieved a perplexity of 10.8 on the WikiText103 benchmark. Also, it achieved an accuracy of 66.5% on the LAMBADA dataset, which outperforms the previous SOTA of 63.2%.

4) LIGHTSEQ2

LightSeq2 [52] proposes a software library that accelerates the training of transformer-based models within GPUs. It is a system-level optimization while maintaining accuracy and training behavior. The system works with BERT (encoder), GPT (decoder), Transformer (encoder-decoder), and vision transformer. The system uses three techniques for improving training speed and efficiency. First (layer-specific kernels), after analyzing Transformer-specific layers in detail, rewriting the kernels with dependencies and other techniques to improve parallelism, and using small kernels to improve GPU utilization. Second (mixed-precision trainer), instead of applying batch updates to many individual full-precision updates, it applies batch updates to reduced-precision parameters. Finally, introduced an efficient memory management technique to minimize the need for frequent allocation and release calls. This strategy involves recycling the memory space of tensors that remain unused during the backward pass.

The system accelerates the entire training process for transformer models. LightSeq2 achieves significant performance improvement on a variety of NLP tasks, including machine translation, on the WMT14 English-German machine translation task, it achieved a 308% speedup on the WMT14 English-German machine translation task compared to PyTorch.

5) COLLIE

CoLLiE [53] introduces a library designed to efficiently facilitate the collaborative training of LLMs using 3D

parallelism [24] (Sections V-C4, V-C1, V-C2), parameter-efficient fine-tuning (PEFT) methods, and optimizers. The library demonstrated significantly improved training efficiency compared to existing solutions. The study empirically evaluates the correlation between model size and GPU memory consumption under different optimization methods and analyzes throughput. Additionally, the study investigates training methods to improve the abilities of the LLaMA-65B model, specifically focusing on following user instructions. Techniques like LoRA [33], LOMO [56] (Section V-A4), AdaLomo [57], and AdamW demonstrated success in boosting the model's instruction following capabilities without sacrificing its overall performance. Notably, LoRA and AdaLomo achieved impressive results, enabling the model to achieve an average score of 56.9.

6) LLM TRAINING FRAMEWORKS AND LIBRARIES: CHALLENGES AND KEY FINDINGS

This section explores five prominent frameworks and libraries: GPipe [3], ByteTransformer [4], Megatron-LM [19], LightSeq2 [52], and CoLLiE [53]. Each offers unique functionalities to overcome limitations in LLM training.

Addressing Training Challenges:

- Distributed training: As LLMs grow complex, training them on a single device becomes impractical. Frameworks like Megatron-LM [19] and CoLLiE [53] employ distributed training algorithms that split the model across multiple GPUs, enabling parallel processing and faster training.
- Efficiency and speed: LightSeq2 [52] tackles training speed through system-level optimizations. It utilizes techniques like layer-specific kernels and mixed-precision training to enhance GPU utilization and reduce memory usage. Similarly, ByteTransformer [4] accelerates transformer models for variable-length inputs in NLP tasks.
- Memory management: Efficient memory allocation is crucial for LLM training. CoLLiE [53] overcomes memory constraints in LLM training by utilizing 3D parallelism to efficiently distribute memory across training machines and GPUs, enabling the training of large models even in resource limited environments.
- Fine-tuning and performance: CoLLiE [53] investigates methods to improve specific LLM capabilities, such as following user instructions. It explores parameter-efficient fine-tuning methods that enhance model performance in targeted areas without compromising overall functionality.

Key Findings:

- GPipe [3] demonstrates successful training of a large multilingual transformer model, achieving superior results compared to training individual smaller models.
- ByteTransformer [4] significantly outperforms existing frameworks in terms of performance for BERT-like transformers on various benchmarks.
- Megatron-LM [19] facilitates training of LLMs with billions of parameters, achieving SOTA on NLP tasks while offering high throughput.
- LightSeq2 [52] accelerates transformer model training by up to 308%, showcasing substantial performance improvements.
- CoLLiE [53] introduces a library for collaborative LLM training, demonstrating improved efficiency and effectiveness in training large models like LLaMA-65B. It explores methods to enhance specific functionalities without impacting overall performance.

B. LLM INFERENCE FRAMEWORKS AND LIBRARIES

This section will introduce the LLM frameworks and libraries designed particularly for inference tasks, followed by a summary of each one (see Table 4).

1) DEEPSPEED INFERENCE

DeepSpeed Inference [5] presents a comprehensive system solution for efficient transformer inference. It has the potential to enable new and innovative applications of transformer models in cloud datacenters and other resource-constrained environments. The system consists of two main parts: DeepSpeed Transformer and ZeRO-Inference [1]. The model is a GPU-only solution that leverages a variety of optimizations to achieve SOTA (minimize) latency and (maximize) throughput for transformer models of all sizes. Specifically, in the first phase DeepSpeed Transformer uses tensor-slicing and inference-optimized pipeline parallelism to scale dense transformer models across GPUs.

For sparse transformer models, it has developed a massive-GPU sparse transformer layer that can extend the scalability of Mixture-of-Experts (MoE) transformer layers to hundreds of GPUs. This is achieved through a combination of parallelism techniques and optimization strategies for communication. Then, DeepSpeed Transformer employs optimized sparse kernels to reduce the computational burden on a single GPU. ZeRO-Inference [1] is a heterogeneous solution that leverages GPU, CPU, and NVMe memory to enable massive transformer inference with limited GPU resources.

It is particularly useful for inferring models that are too large to fit in GPU memory. It works by partitioning the model weights across multiple GPUs and offloading unused weights to CPU and NVMe memory. This allows ZeRO-Inference to infer models that are much larger than would be possible with GPU-only solutions. As a result, DeepSpeed Inference boosts throughput by more than 1.5× for throughput-oriented scenarios and minimizes the latency by more than 7.3× compared to existing solutions for latency orientation scenarios. It facilitates real-time inference at a trillion-parameter scale by utilizing hundreds of GPUs, marking an unparalleled achievement in terms of inference scale. This technology allows for the inference of models that are 25 times larger than what GPU-only solutions can handle,

TABLE 4. Summary of LLM inference frameworks and libraries.

Studies	Aims	Outcomes
DeepSpeed Inference [5]	GPU-only solution, powerful and versatile system for **efficient transformer inference** at scale.	Boosts throughput by 1.5×, minimizes latency by 7.3×, empowers inference of 25×-larger models at 84 TFLOPS.
FlexGen [17]	An offloading engine model designed for high-throughput LLM inference by efficiently **utilizing limited resources** from GPU, CPU, and disk, employing various techniques to enhance efficiency.	**1)** Achieved 40× throughput speedup with 5000-second latency for batch size 64 or 2048 tokens. **2)** Achieved 79× throughput speedup with 12000-second latency for batch size 256 or 8192 tokens. **3)** Achieved 100× throughput speedup with 4000-second latency for batch size 144 or 4608 tokens using 4-bit quantization compression.
NLP-Fast [58]	Accelerates the performance of large-scale **heterogeneous** NLP models.	Evaluated a variety of NLP models and hardware platforms, including CPU, GPU, and FPGA. The throughput improved by up to 2.92×, 1.59×, and 4.47× over the baseline performance.
TurboTransformers [11]	A lightweight, easy-to-use system that enables efficient deployment of transformer models for **online services**.	It introduces three innovative features:**1)** Efficient GPU-based batch reduction kernels for Softmax and LayerNorm. **2)** Sequence-length-aware memory allocation algorithm. **3)** New batch scheduler employing dynamic programming for optimal throughput on variable-length requests.
PetS [59]	A **unified framework** for multitask PET serving in a single system.	Enables 26× more concurrent tasks and enhances serving throughput by 1.53× on Desktop GPUs and 1.63× on Server GPUs.
PETALS [60]	A collaborative platform for **distributed inference** and fine-tuning of LLMs **over the internet**.	With an optimal hardware setup involving CPU RAM offloading via PCIe 4.0 and GPU pairs connected through PCIe switches, offloading 176B parameters takes 5.5 seconds in a regular setup and 11 seconds in a multi-GPU setup, with each GPU having 1 GB of memory per billion parameters and PCIe 4.0 throughput of 256 Gbit/s (or 128 Gbit/s behind a PCIe switch for two GPUs).
LightSeq [61]	Inference library, addresses the need for efficient and convenient deployment of Transformer models in **online services**.	In machine translation benchmarks, consistently outperforms TensorFlow and FasterTransformer (FT), achieving up to 14× and 1.4× speedups, respectively.
Easy and Efficient Transformer (EET) [62]	Library to **accelerate transformer inference**.	Compared against Fairseq, LightSeq, and Faster Transformer (FT) on 2080Ti and A100, EET achieves speedups of 4.48-20.27× and 4.30-27.43×, respectively, over Fairseq. On 2080Ti, EET achieves a speedup of 0.82-2.46× over LightSeq for model sizes of 768 and 1024. Additionally, EET achieves speedups of 1.21-6.30× and 1.62-812× over FT v3.1 on 2080Ti and A100, respectively, and a speedup of 1.40-4.20× over FT v4.0 on A100.
Splitwise [63]	**Improve LLM inference efficiency by separating compute-intensive and memory-intensive phases** onto different machines with hardware specialization.	Achieve significant outcomes like up to 1.4× higher throughput at 20% lower cost or 2.35× higher throughput with same cost and power consumption.
Zhang *et al.*, [64] LLMCompass	**Evaluate hardware design** for LLMs using LLMCompass library for recommending cost-effective hardware designs.	Achieve significant outcomes like accurate (average error 10.4% for task time, 4.1% for LLM tasks), simulates GPT-3 175B on 4× A100 GPUs in 16 minutes, identifies more affordable hardware designs.
PowerInfer [65]	Build a **faster LLM inference engine** for consumer-grade GPUs.	Achieve significant speedups (up to 11.69× faster) by optimizing for hot neurons on GPU and cold neurons on CPU, while maintaining accuracy and approaching performance of high-end GPUs.

Bold text in the "Aims" column indicates the framework's primary area of specialization or the range of tasks it is designed to address.

achieving a substantial throughput of 84 TFLOPS, which is over 50% of the A6000 peak performance.

2) FLEXGEN

Accelerating LLM inference is achievable by using multiple high-end accelerator technologies, due to their high computational and memory requirements. FlexGen [17] study proposes an offloading engine model which focuses on using (resource-constrained devices) limited resources to reach high-throughput LLM inference. The engine is flexible for configuration using different hardware resources by aggregating memory and computation from the GPU, CPU, and disk. To optimize throughput within the search space, researchers developed a linear programming-based search algorithm, through it the model can find efficient patterns for saving and accessing tensors. It has a larger space of batch size options to choose from without sacrificing accuracy through using 4-bit to compress weights and attention cache without the need for retraining or calibration. The model's efficiency has been experimented by using NVIDIA T4 (16 GB) GPUs for running OPT-175B. It significantly outperforms DeepSpeed Zero-Inference [1], [5] and Hugging Face Accelerate by enabling significantly larger batch sizes, often reaching orders of magnitude higher than its competitors. As a result, it can achieve significant speedups in throughput. On a single T4 GPU equipped with 208 GB CPU DRAM and a 1.5 TB SSD, input sequence length 512, and output sequence length 32: With a latency of 5,000 seconds, it (effective batch size 64) surpasses DeepSpeed Zero-Inference (batch size 1) by over 40×, whereas Hugging Face Accelerate fails to complete a single batch. Furthermore, it can reach 69× higher throughput with a higher latency of 12000 seconds compared to baselines (effective batch size 256, or 8192 tokens in total). Finally, the model can achieve 100× higher maximum throughput (effective batch size 144, or 4608 tokens in total) with 4-bit quantization to compression and 4000 seconds by holding all weights in the CPU and getting rid of disk offloading. The model achieved these results by aggregating memory and computation from the GPU, CPU, and disk, and by using a number of techniques to improve efficiency, such as I/O schedule tasks, possible compression techniques, and distributed pipeline parallelism.

FlexGen is a significant advancement in LLM inference, as it enables high-throughput generation on resource-constrained devices. This opens new possibilities for deploying and using LLMs in a wider range of applications.

3) NLP-FAST

NLP-Fast [58] is a system that accelerates the performance of large-scale heterogeneous NLP models by identifying performance-critical operations and applying holistic model partitioning, cross-operation zero skipping, and model/config adaptive hardware reconfiguration. NLP-Perf, a performance analysis tool, collects performance data for NLP models and identifies performance-critical operations. Holistic model partitioning is a comprehensive optimization technique, which integrates three model partitioning approaches: partial-head update, column-based algorithm, and feed-forward splitting, to facilitate end-to-end model partitioning. Cross-operation zero skipping, skips zero or near-zero values across multiple operations, which can significantly reduce the amount of computation required, these two optimization can be executed on different hardware platforms. Model/config adaptive hardware reconfiguration, reconfigures the model architecture for the specific hardware platform that it is running on, which can further improve performance. NLP-Fast has been evaluated on a variety of NLP models and hardware platforms, including CPU, GPU, and FPGA. The evaluation results show that NLP-Fast can improve throughput by up to 2.92×, 1.59×, and 4.47× over the baseline performance on each platform.

4) TURBOTRANSFORMERS

TurboTransformers [11] is a lightweight, easy-to-use system that enables efficient deployment of transformer models for online services. It achieves SOTA performance on GPU platforms by proposing three innovative features that distinguish it from other similar models: Firstly, proposes an efficient and parallel GPU-based batch reduction kernels algorithm for Softmax and LayerNorm. Secondly, proposes a sequence-length-aware algorithm for memory allocation to efficiently balance memory allocation and deallocation, this algorithm overcomes the problem of variability of the input sentences. Finally, applying the framework involves utilizing a novel batch scheduler that leverages dynamic programming to achieve the optimal throughput on variable-length requests. It is a lightweight and easy-to-use system that can be integrated into PyTorch code with just a few lines of code. This makes the model a very accessible option for researchers and practitioners who want to use transformer models for online services.

5) PETS

The existing large-scale transformer modes follow the pre-train-then-fine-tune paradigm, copying the entire model for each downstream task consumes a lot of storage. This approach is unsuited for multi-purpose serving. Parameter Efficient Transformers (PET) reduce the resource overhead. They share the pre-trained model among tasks and only fine-tune a specific portion based on the task parameters. Prior to PetS [59], the serving systems did not have any mechanism to provide flexibility for PET task management, and also there is no available efficient method to serve queries to different task batches. It is the first unified framework for multi-task PET serving in a single system. As a class of transformer models PETs have been designed to be more efficient in terms of both parameters and computation. Therefore, PETs are well-suited for deployment in resource-constrained environments. Conventional serving frameworks move data between the GPU and CPU memory when the GPU cannot hold all of the data for the tasks that are being processed. This reduces the throughput of the system. It has the potential to revolutionize the way that LLMs are served, making it possible to deploy and run LLMs on a wider range of devices and with lower resource requirements. This could make LLMs more accessible to a wider range of users and businesses. Pets framework is a flexible PET tasks management mechanism and a specialized PET Inference Engine (PIE) that allows both inter-task and inter-algorithm query-batching. It enables 26× more concurrent tasks and enhances serving throughput by 1.53× on Desktop GPUs and 1.63× on Server GPUs.

6) PETALS

PETALS [60] emerges as a collaborative platform specifically designed for the distributed inference and fine-tuning of LLMs over the internet. It aims to overcome the limitations associated with existing approaches, offering a range of advantages. The platform focuses on achieving high performance by leveraging pipeline parallelism, effectively enhancing the efficiency of LLM inference and fine-tuning processes. Furthermore, it showcases scalability, demonstrating its capability to support a substantial number of users and accommodate large-scale LLMs. This adaptability is complemented by the provision of a flexible API, allowing users to tailor the inference and fine-tuning processes according to their specific requirements. The PETALS key feature is its emphasis on collaboration, providing a framework that enables multiple participants to actively engage in LLM inference and fine-tuning tasks collectively. The collaborative nature of PETALS contributes to its potential in democratizing access to LLMs, making them more accessible and valuable across a diverse range of applications. In summary, PETALS emerges as a promising platform with the potential to enhance the accessibility and utility of LLMs. It can offload a 176B parameter model in 5.5 seconds for a regular setup and 11 seconds for a multi-GPU setup. These results demonstrate PETALS's superior performance for running large models with limited resources.

7) LIGHTSEQ

LightSeq [61] is a lightweight inference library, addresses the need for efficient and convenient deployment of Transformer

models in online services. It utilizes a combination of GPU optimization techniques, including coarse-grained fused kernel functions, hierarchical auto-regressive search, and dynamic GPU memory reuse strategy, to achieve significant performance gains compared to TensorFlow and FasterTransformer (FT). It supports a wide range of models and search algorithms, encompassing BERT, GPT, Transformer, and Variational Autoencoders (VAEs), whereas seamlessly integrating with popular models like BERT [8], RoBERTa [66], GPT, VAEs, MT Transformer, and Speech Transformer. The library is user-friendly, with a serving system and CUDA implementations, enabling easy deployment of popular models online without code modification. It addresses the deployment challenges of resource-intensive sequence models, narrowing the performance gap between large models and the demands of online services. In machine translation benchmarks, it consistently outperforms TensorFlow and FasterTransformer (FT), achieving up to 14× and 1.4× speedups, respectively.

8) EET

Easy and Efficient Transformer (EET) [62] offers a library designed to accelerate transformer inference. It encompasses a range of optimizations for transformer inference, spanning both algorithmic and implementation aspects. To address the inefficiencies of explicit matrix addition and masked attention, the study implements custom CUDA kernels. Also, to extend all kernels to support a larger model size up to 12288 and a longer sequence above 4096 the research proposes a new method called thread block folding. Furthermore, the study introduced a CUDA memory management mechanism aimed at minimizing memory usage for models of the size. EET evaluated against Fairseq, LightSeq, and Faster Transformer (FT), On both a 2080Ti and A100, EET achieves a speedup of 4.48-20.27× and 4.30-27.43×, respectively, compared to Fairseq. On a 2080Ti, EET outperforms LightSeq [61] with a speedup of 0.82-2.46× for model sizes of 768 and 1024. EET attains a speedup of 1.21-6.30× and 1.62-812× over FT v3.1 on a 2080Ti and A100, respectively, and a speedup of 1.40-4.20× over FT v4.0 on an A100.

9) SPLITWISE

Splitwise [63] investigates inefficiencies in LLM inference, which relies on expensive GPUs. The analysis reveals two distinct phases in LLM inference: a compute-intensive prompt computation and a memory-intensive token generation phase. While existing methods optimize batching and scheduling, they underutilize compute resources during token generation. To address this, the study proposes separating these phases across different machines. This allows for hardware optimized for each phase: powerful machines for prompt computation and potentially older, more cost-effective machines for token generation. Splitwise facilitates communication between these machines using fast interconnects. This approach enables the design of clusters optimized for throughput, cost, or power consumption. The model achieves up to 1.4× higher throughput at 20% lower cost or 2.35× higher throughput with the same cost and power consumption. This approach improves LLM inference efficiency by leveraging hardware specialization, leading to more cost-effective and power-efficient deployments.

10) LLMCOMPASS

Zhang et al. [64] propose LLMCompass, a library that efficiently evaluates hardware design for LLMs. LLMCompass considers various hardware options and identifies the optimal configuration for a specific task. The study also uses a cost model to recommend the most economical design. The library demonstrates high accuracy, with an average error of 10.4% for predicting task execution time and 4.1% for LLM tasks, compared to real hardware. Notably, the model can simulate running a massive LLM like GPT-3 175B on a powerful computer setup with 4× A100 GPUs in just 16 minutes. Leveraging LLMCompass, the study identified hardware designs that are more affordable than current options (e.g., using less powerful components or cheaper memory) while still offering good performance. These designs could make LLMs more accessible to a wider range of users.

11) POWERINFER

PowerInfer [65] is a high performance inference engine designed to run LLMs efficiently on consumer-grade GPUs. It leverages the power-law distribution of neuron activation in LLMs, assigning frequently activated (hot) neurons to the GPU and input-specific (cold) neurons to the CPU. This hybrid approach significantly reduces the pressure on GPU memory and minimizes data transfers between CPU and GPU. Furthermore, PowerInfer incorporates adaptive predictors and neuron-aware sparse operators to optimize performance and maintain model accuracy. Evaluations demonstrate that PowerInfer on an NVIDIA RTX 4090 GPU achieves inference speeds up to 11.69× faster inference than systems like llama.cpp.It delivers an average token generation rate of 13.20 tokens per second, rivaling the performance of top-tier server-grade GPUs.

12) LLM INFERENCE FRAMEWORKS AND LIBRARIES: CHALLENGES AND KEY FINDINGS

This section presents various frameworks and libraries designed to improve the efficiency of running LLMs. The following paragraphs discuss the challenges and key findings of the reviewed studies.

Challenges of LLM Inference:

- LLMs are computationally expensive due to their massive size and complex architecture.
- Traditional inference methods struggle to handle large models on resource-constrained devices.
- Balancing speed, accuracy, and resource utilization is crucial for deploying LLMs in real-world applications.

Key Findings:

- Hardware specialization: Splitwise [63] proposes separating compute-intensive and memory-intensive phases onto different machines with specialized hardware.
- Resource optimization: FlexGen [17] utilizes various techniques like I/O scheduling, compression, and distributed processing to efficiently use resources from CPU, GPU, and disk.
- Algorithmic optimizations: Libraries like EET [62] and LightSeq [61] implement custom algorithms and memory management techniques to accelerate inference on GPUs.
- Heterogeneous platforms: NLP-Fast [58] leverages different hardware platforms (CPU, GPU, FPGA) by identifying performance-critical operations and applying targeted optimizations.
- Distributed inference: PETALS [60] facilitates collaborative inference and fine-tuning of LLMs across a network, enabling scalability and efficient resource utilization.
- Efficiency gains: Several frameworks achieve significant performance improvements. DeepSpeed Inference [5] boasts throughput boosts of 1.5× and latency reductions of 7.3×. FlexGen demonstrates even greater throughput gains, particularly on resource-constrained devices. Other frameworks like NLP-Fast [58], TurboTransformers [11], LightSeq [61], and EET [62] show promising results in accelerating inference.

C. LLM DEPLOYMENT AND SERVING LIBRARIES

As mentioned in section IV, some of the frameworks and libraries are utilized for multiple purposes. Besides vLLM [67] (Section IV-C1), the models used for deployment and serving purposes are mentioned in these sections LightSeq2 IV-A4, TurboTransformer IV-B4, PetS IV-B5.

1) VLLM

vLLM [67] is a high performance system that efficiently handles LLMs at a large scale. The model tackles the memory limitations of existing LLM serving systems through a novel algorithm called PagedAttention (Section V-A1). PagedAttention splits the KV cache into manageable blocks, minimizing wasted memory and enabling efficient sharing across requests. vLLM is a distributed system that supports popular LLMs and even models exceeding single GPU memory. Evaluations present vLLM significantly improves throughput by 2-4× faster compared to existing systems, especially for complex tasks involving long sequences, large models, and intricate decoding algorithms. This makes vLLM a significant advancement for efficient LLM processing, enabling faster and more scalable LLM applications.

2) LLM DEPLOYMENT AND SERVING LIBRARIES: CHALLENGES AND KEY FINDINGS

As explored in previous sections (Sections IV-A6 and IV-B12) a variety of LLM frameworks exist that hold promise for deployment and serving applications. This section will discuss the key challenges and findings associated with LLM deployment and serving.

Challenges of LLM Deployment and Serving:

- Memory limitations: Large LLMs can easily overwhelm the memory capacity of a single GPU. This limits their deployment and serving for real-world applications.
- Scalability: Effectively handling multiple user requests simultaneously with large LLMs requires efficient scaling solutions.
- Variability of input: LLM performance can suffer when dealing with input sequences of varying lengths, requiring dynamic memory allocation strategies.
- Ease of deployment: Integrating complex LLM serving systems into existing workflows can be challenging for researchers and practitioners.

Key Findings:

- PagedAttention: This algorithm (introduced by vLLM [67]) breaks down the KV cache into manageable blocks, minimizing wasted memory and enabling efficient sharing across requests. This is a significant improvement for processing large LLMs.
- Efficient GPU utilization: TurboTransformers [11] utilize techniques like parallel GPU kernels and dynamic batch scheduling to optimize performance on GPUs. This translates to faster inference for transformer-based models.
- System-level optimizations: LightSeq2 [52] demonstrates how system-level optimizations within the training process can significantly improve training speed and efficiency for transformer models. This translates to faster deployment of LLMs in general.

These findings from vLLM [67], TurboTransformers [11], and LightSeq2 [52] offer promising solutions for overcoming challenges in LLM deployment and serving. By focusing on memory management, efficient GPU utilization, user-friendly tools, and co-optimization.

V. TRAINING OPTIMIZATION

Training optimization in LLMs involves improving the efficiency and effectiveness of the training process. This encompasses a range of techniques and strategies aimed at improving factors such as convergence speed, model generalization, and resource utilization. The goal of training optimization is to achieve the desired model performance with faster training times, reduced resource requirements, and improved overall training effectiveness. In this section, we will focus on model optimization, size reduction, distributed training, and heterogeneous training (Fig. 5).

A. MODEL OPTIMIZATION

Model optimization in LLMs refers to the process of improving the model's architecture, structure, or parameters to enhance its overall performance. We stated various techniques aimed at achieving better accuracy, efficiency, or both. Common model optimization strategies for LLMs include

FIGURE 5. Training optimization techniques.

TABLE 5. Comparative analysis between different categories.

Optimization	Focus	Performance	Cost	Scalability
Training Optimization	Focuses on accelerating the training process and minimizing resource usage within LLMs. This is accomplished through various techniques that enhance the efficiency of the training workflow. The aim is to achieve the same level of model performance in a reduced timeframe and with less computational power.	Techniques like SwapAdvisor [68] enables the training of models up to 12× larger than the standard GPU memory capacity while preserving substantial performance, ZeRO [18] achieves 10× speedup, train trillion parameter models (8× larger than existing models), Cramming [16] enables single-GPU LLM training in a one day, and Megatron-LM [69] outperforms ZeRO-3, achieving a remarkable 70% improvement in performance for models with both 175 billion and 530 billion parameters.	Lower training costs due to faster training times, reduced memory usage, or enabling training on less powerful hardware.	Techniques like ZeRO [18] can scale to trillionn parameters, SparseGPT [70] processes very large models (OPT-175B, BLOOM-176B) efficiently, FlexFlow [71] improves parallelism efficiency by 2.5-10×, ZeRO-Offload [20] enables training 10× larger models on the same hardware, and ZeRO-Infinity [1] Highly scalable for training models with trillions of parameters.
Hardware Optimization	Systematically enhances the performance, efficiency, and functionality of computer hardware by addressing bottlenecks in hardware architecture, software, and operating systems. This approach can increase overall speed, reduce power consumption, and improve hardware reliability. Additionally, it enables more efficient use of hardware resources and allows for deployment on less powerful devices.	Techniques like FlexGen [17], LightSeq2 [52], TurboTransformers [11] improve performance (throughput, latency) for inference, potentially reducing operational costs.	Lower deployment costs by enabling inference on resource-constrained devices (CPUs, FPGAs) or requiring fewer servers for the same workload.	Techniques like TurboTransformers [11] and Splitwise [63] can potentially scale well on different hardware configurations.
Scalability and Reliability	Improve the ability to train and run large models on distributed systems and handle potential hardware issues.	Techniques like PETALS [60] achieves faster inference for large language models, with an optimal setup inferring a 176 billion parameter model in 5.5 seconds.	Techniques like SWARM Parallelism [10] trains large models on unreliable, heterogeneous devices with low network bandwidth.	Techniques like ZeRO-Offload [20] enables large model training on single GPUs and scales to larger systems using model parallelism.

algorithmic optimization (section V-A1), layer-specific kernels (section V-A2), model partition (section V-A3), fine-tuning (section V-A4), and scheduler optimization (section V-A5).

1) ALGORITHMIC OPTIMIZATION

FlexGen [17] devised a linear programming-based search algorithm to optimize throughput within the search space. This model can identify efficient patterns for tensor saving and access.

Building on techniques for efficient model execution, SwapAdvisor [68], proposes a novel approach to deep learning memory management, that enables the training and serving of large models despite limited GPU memory. Through smart swapping between CPU and GPU memory, it optimizes scheduling, memory allocation, and swap planning to maximize computational efficiency. This approach allows training models up to 12× beyond the usual GPU memory limit while maintaining significant performance. It stands as an innovative solution for deep learning with limited GPU resources.

NLP-Fast [58] employs algorithmic optimization techniques to enhance the performance of large-scale heterogeneous NLP models. One of the techniques is cross-operation zero skipping, which eliminates unnecessary computations by skipping zero or near-zero values across multiple operations. By leveraging these techniques, NLP-Fast can significantly improve the overall performance of NLP models on various hardware platforms.

ByteTransformer [4] was developed to address the challenges of redundant computations and memory overhead in transformer models, it employs a combination of algorithmic optimizations and memory-efficient techniques, including a padding-free algorithm, fused MHA, and manually optimized memory sizes of layer normalization. These techniques effectively eliminate unnecessary computations, minimize memory footprint, and reduce the cost of accessing GPU global memory, leading to significant performance gains compared to other transformer frameworks.

Sheared LLAMA [72] model introduces a dynamic batch loading. This innovative algorithm efficiently adjusts the composition of sampled data within each training batch based on varying losses observed across different domains. The primary objective is to dynamically update the batch loading process to maximize learning efficiency, ensuring that the model achieves the reference loss approximately simultaneously across all domains. All training experiments have been done on a maximum of 16 Nvidia A100 GPUs (80 GB).

GrowLength [73] enhances the pre-training of LLMs by gradually increasing the training length, it introduces an innovative method inspired by the principles of extending context windows during training. This innovative approach accelerates the pre-training phase of LLMs by dynamically and gradually extending the length of the training sentence. The primary benefit of this method lies in its adaptability and efficient resource utilization. It optimizes computational resources effectively, allowing models to process more tokens within a restricted time frame. Throughout the training process, the model incrementally increases the training length, resulting in reduced computational expenses and enhanced training efficiency. The experiments have been done in three different setups: 1) LLM128, in this setup the sentence length fixed of 128 tokens, totaling 0.36B. 2) LLM1024, sentence length was set to 1024 tokens, the same total tokens as LLM128, allowing direct runtime comparison. 3) GrowLength, in this experiment the method progressively grew from 128 to 1024 tokens, saving time with shorter lengths and enhancing performance at 1024 tokens. As a result, with equivalent tokens, LLM1024 required longer pre-training than LLM128. Using GrowLength led to a significant decrease in loss, and emphasized its computational efficiency and practical value in resource-constrained configurations.

PagedAttention [67] introduces another innovative approach to improve learning efficiency. This novel attention algorithm is inspired by virtual memory used in operating systems. The algorithm splits the KV cache into fixed-size blocks, similar to memory pages, reducing fragmentation and enabling efficient sharing across requests. This approach significantly improves memory utilization and allows for larger batch sizes.

2) LAYER-SPECIFIC KERNELS

LightSeq [61] (Section IV-B7) is a lightweight inference library instead of using a straightforward combination of the fine-grained GPU kernel functions in TensorFlow or PyTorch implementations, it utilizes a method known as coarse-grained fusion. This strategy mitigates the significant time costs associated with numerous kernel function launches and GPU memory I/O operations for intermediate results. Therefore, it achieves a significant reduction in the number of atomic kernel functions, leading to a remarkable performance boost compared to conventional TensorFlow approaches.

LightSeq2 [52] (Section IV-A4) proposed a software library that accelerates the training of transformer-based models within GPUs. It is a system-level optimization while maintaining accuracy and training behavior. The library works with BERT (encoder), GPT (decoder), Transformer (encoder-decoder), and vision transformer. LightSeq2 uses three techniques for improving training speed and efficiency. The first technique used for increasing GPU utilization is layer-specific kernels technique. After analyzing Transformer-specific layers in detail, rewriting the kernels with dependencies and other techniques to improve parallelism, and using small kernels to improve GPU utilization.

3) MODEL PARTITION

NLP-Fast [58] (Section IV-B3) accelerates the performance of large-scale heterogeneous NLP models by applying several techniques. It proposed holistic model partitioning as a solution for optimizing every operation in NLP models. This technique breaks down the model into smaller, more efficient submodels can be tailored for different hardware platforms.

GPipe [3] (Section IV-A1) is an efficient, task-independent, and supports any deep neural network architecture that can be expressed as a sequence of layers.

It can use different accelerators, each of which supports re-materialization. GPipe partitions the model across the accelerators, with each accelerator responsible for a sequence of layers (called a cell).

Megatron-LM [19] introduces a new method for training LLMs, which empowers the training of exceptionally large transformer models with billions of parameters within GPUs. Megatron-LM uses intra-layer model parallelism, a strategy that subdivides the model into smaller submodels capable of being trained separately.

4) FINE-TUNING

AlphaTuning [74] is a novel method specifically designed for large-scale pre-trained language models (PLMs). It combines the quantization of PLMs with fine-tuning, only a subset of quantized parameters is fine-tuned for the target task. This selective approach significantly decreases the overall memory footprint and the number of parameters to be trained. Despite these reductions, it maintains performance levels comparable to full fine-tuning across a diverse range of downstream tasks.

QFT [75] proposes a novel framework designed for memory-efficient fine-tuning of LLMs. The model utilizes quantization techniques to significantly reduce memory usage during fine-tuning while preserving model performance. The framework adopts the Lion optimizer, known for its memory efficiency and compatibility with quantization, and the conversion of all model states into integers to minimize memory footprint. The study also features a specialized gradient flow and parameter update scheme tailored for quantized weights. Extensive evaluations show the framework's effectiveness, allowing fine-tuning of large LLaMA-7B models with less than 30 GB of memory on a single A6000 GPU a substantial reduction compared to standard methods while maintaining similar performance across various benchmarks.

LOMO [56] is a novel technique for training LLMs on machines with limited GPU capacity. LOMO proposed a memory-efficient update method that greatly lowers memory consumption compared to traditional methods. This enables fine-tuning large models, such as those with 65 billion parameters, on consumer-grade GPUs like the RTX 3090. The study validates LOMO's efficiency through analyses of memory usage, performance testing, and benchmark task evaluations. Existing techniques like LOMO reduce memory usage but compromise performance.

AdaLomo [57] offers a better solution. It incorporates a key feature from the powerful AdamW optimizer (adaptive learning rate) but uses clever techniques to stay memory-friendly. This allows AdaLomo to match AdamW's performance on various tasks, making LLM training more accessible with less memory needed. On average, AdaLomo achieved scores of 30.8, 39.7, 51.0, and 56.9 on the LLaMA benchmark for models with 7B, 13B, 30B, and 65B parameters, respectively.

LoRA [33] is a method designed to adapt LLMs, such as GPT-3, for specific tasks, addressing the challenges of traditional fine-tuning. Instead of adjusting all pre-trained model weights, LoRA introduces trainable rank decomposition matrices into each layer of the Transformer architecture, significantly reducing the number of trainable parameters needed for downstream tasks. This approach reduces the number of trainable parameters by 10,000× and reduces GPU memory requirements by 3× compared to GPT-3 175B fine-tuned with Adam, while maintaining or improving model quality on benchmarks like RoBERTa, DeBERTa, GPT-2, and GPT-3. LoRA achieves higher training throughput with no added inference latency and facilitates efficient task-switching by sharing the pre-trained model and only optimizing the small low-rank matrices, thereby reducing storage and hardware costs. It is versatile and can be combined with other methods, applicable to any neural networks with dense layers. For GPT-3 175B, LoRA with 4.7M parameters achieves 73.4% accuracy on WikiSQL, 91.7% on MNLI-m, and Rouge-1/2/L scores of 53.8/29.8/45.9 on SAMSum, demonstrating its superior performance and efficiency.

5) SCHEDULER OPTIMIZATION

TurboTransformers [11] (Section IV-B4) introduces a novel sequence-length-aware batch scheduler that utilizes dynamic programming (DP) to optimize response throughput. This approach overcomes the limitations of traditional batch schedulers that struggle with varying input lengths. The model considers sequence length in batching decisions. The scheduler's core algorithm operates in $O(n^2)$ time complexity, making it efficient for real-time applications.

PetS [59] (Section IV-B5) introduces a unified framework aimed at enhancing multi-task PET serving efficiency. It comprises two main components: a flexible PET task management mechanism and a specialized PIE. Together, these components facilitate both inter-task and inter-algorithm query-batching, streamlining the processing of PET tasks. This approach optimizes resource utilization and enhances the efficiency of PET serving. The PET task scheduler efficiently schedules PET operations to run in parallel on the GPU, maximizing hardware utilization and performance. It dynamically assigns PET tasks to CUDA streams, considering both PET operator characteristics and system resource constraints. This lightweight online scheduling strategy effectively balances computational and memory-intensive tasks, leading to improved throughput and reduced latency in multi-task PET serving scenarios.

B. SIZE REDUCTION OPTIMIZATION

Minimizing the size or complexity of LLMs is a crucial optimization technique known as size reduction optimization. This approach is essential for addressing challenges associated with memory demands, computational efficiency, and storage limitations. Size reduction optimization encompasses various techniques, including model compression and quantization (Section V-B1), pruning (Section V-B2), and hyperparameter optimization (Section V-B3).

Cramming [16] investigates the trade-offs involved in scaling down language model training, and investigates different parts of the training pipeline to identify the modifications that have the biggest impact on performance in a scaled-down setting. As a result, the research figured out that even under customized and constrained settings, the scaling laws [76] were almost true as it was observed for performance in large-compute settings. As a predictable outcome of these laws, it is a challenging task to perform downscaling. However, a smaller model architecture requires less computation power and allows to boost up the gradient computations, as a result the rates of the improved model within the time remain nearly unchanged. The study shared that doing modifications to the training methodology leverages scaling law to bring about enhancements by increasing the effective rate of gradient computations without sacrificing the model size. Two model setups have been analyzed: one utilizing a classical rtx2080ti GPU, and the other employing a modern rtxa4000 or rtxa6000 GPUs. Each setup was configured with 4 CPU cores and 32 GB of RAM. The paper proposes several modifications to the standard training pipeline to make it possible to train a language model on a single GPU in one day. As a result, each of these modifications has a direct impact on the model size reduction such as smaller model architecture, shorter training schedule, lower learning rate, mixed precision training, and specialized training library.

1) MODEL COMPRESSION AND QUANTIZATION

FlexGen [17] (Sections IV-B, and V-A1) through a linear programming-based search algorithm identifies optimal patterns for tensor storage and retrieval. Furthermore, it employs 4-bit quantization to compress weights and attention cache without compromising accuracy, significantly reducing model size and memory footprint. These optimizations enable it to achieve impressive throughput gains compared to existing LLM inference systems.

SWARM parallelism [10], proposes a model for training a large model with unreliable heterogeneous devices with low network bandwidth by using dynamically generated, randomized pipelines instead of static pipelines dynamically instead of statically. The study incorporates 8-bit compression to minimize model size and facilitate training on resource-constrained devices with limited network bandwidth. This compression technique significantly reduces the amount of data that needs to be transferred between nodes during training, leading to improved efficiency and throughput.

QMoE [77] is a compression and execution framework that reduces memory usage significantly. This is achieved through a scalable compression algorithm that shrinks trillion parameter MoEs down to less than 1 bit per parameter. This impressive compression is facilitated by a custom format specifically designed to work with bespoke GPU kernels, enabling efficient processing with minimal slowdowns. QMoE can compress the SwitchTransformer-c2048 model to under 160 GB (20× compression, 0.8 bits per parameter) with minimal impact on accuracy, achievable within a day on a single GPU. This enables the execution of trillion-parameter models on affordable commodity hardware, such as a single server with 4× NVIDIA A6000 or 8× NVIDIA 3090 GPUs, with less than 5% runtime overhead compared to uncompressed inference. The framework reduces the model size from 3.2TB in bfloat16 to less than 160 GB, allowing efficient execution on commodity hardware and enhancing the practical adoption and research of MoE architectures.

AlphaTuning [74] is a compression-aware parameter-efficient adaptation method for large-scale PLMs. It combines the quantization of PLMs with fine-tuning, but only a subset of quantized parameters are fine-tuned for the target task. This significantly reduces the total memory footprint and the number of trainable parameters, while still achieving comparable performance to full fine-tuning on a variety of downstream tasks. It relies on binary-coding quantization, a technique that decomposes full-precision parameters into binary parameters alongside a distinct set of scaling factors. The model is evaluated across various PLMs and downstream tasks, and achieves comparable performance to full fine-tuning, even at low bitwidths. While it was applied to GPT-2 and OPT, it achieved a compression ratio of over 10 times under 4-bit quantization and a reduction in the number of trainable parameters by over 1,000-fold, while still achieving competitive performance on a variety of downstream tasks.

GPTQ [78] proposes a new highly accurate and highly efficient post-training quantization method based on approximate second-order information which is called a new one-shot weight quantization. This model reaches a level that is considered acceptable to precisely quantize models to 3 or 4 bits per parameter, it requires a few hours at most to run on a model that has hundreds of billions of parameters. The model experimented on both OPT-175B and BLOOM-176B it took approximately 4 GPU hours by reducing the bitwidth down to 3 or 4 bits per weight, with minimal loss of accuracy compared to the uncompressed baseline. Compared to previous one-shot quantization methods the model achieves more than twice the compression without sacrificing accuracy. Also, within the method for first-time models with 175 billion parameters can execute inside a single GPU for generative inference. The results show that these enhancements can boost performance by up to 3.25× while using high-end GPUs (NVIDIA A100) over FP16 and reach 4.5× and up to 4.5× while using more cost-effective GPUs (NVIDIA A6000). This model can also achieve robust accuracy results even using an extreme quantization regime, while the weights are quantized to 2-bit or ternary quantization level.

FPTQ [79] also proposes a novel post-training quantization technique to address the deploying LLM challenge. This technique effectively compresses LLMs into a format using 4-bit weights and 8-bit activations (W4A8). This approach achieves SOTA performance on popular LLMs like BLOOM [80], LLaMA [14], and LLaMA-2 without requiring further fine-tuning. FPTQ offers a significant advantage by

optimizing both memory usage and computation efficiency during the inference stage without sacrificing accuracy. This technique simplifies the deployment process for LLMs and makes them more practical for real-world use. The model was validated on various datasets, including LAMBADA, MMLU, and a set of Common Sense QA tasks. The researchers compared the model's performance to an existing technique called LLM-QAT (LLM-Quantization-Aware Training). However, limited data availability for LLM-QAT restricted the comparison to the Common Sense QA dataset. On this task, FPTQ achieved results closer to the FP16 compared to LLM-QAT. While the analysis was only possible for 7B and 13B parameter LLaMA models due to data limitations, FPTQ consistently performed better across all subsets of the dataset. This is evidenced by the average scores: 73.38 and 76.81 for LLaMA-7B and 13B, respectively. These findings suggest that FPTQ is an effective approach for LLM quantization.

Norm Tweaking [54] method introduces a novel technique in quantization, specifically for LLMs. While existing quantization methods like GPTQ [78] achieve acceptable 4-bit weight-only quantization, attempts at lower bit quantization often lead to significant performance degradation. It introduces a strategy to rectify the quantized activation distribution, restoring accuracy for LLMs. The method involves generating calibration data and applying channel-wise distance constraints to normalization layer weights. Experiments show significant improvements in both weight-only quantization and joint quantization of weights and activations, achieving high accuracy even at 2-bit quantization. It offers a practical solution for reducing computational and storage costs in LLMs while maintaining performance.

FineQuant [81] introduces an innovative weight-only quantization technique that significantly decreases memory usage and speeds up LLM inference with minimal quality loss. Key features of this technique include utilizing pre-trained model weights without further fine-tuning, applying adaptive granularity quantization to minimize accuracy loss, and implementing an efficient GPU processing approach. Tested on large-scale models like OPT-175B, FineQunat demonstrates minimal accuracy loss, achieves up to 3.65× higher throughput with the same number of GPUs, and reduces resource demands.

PETALS [60] (Section IV-B6) is a collaborative platform for distributed inference and fine-tuning of LLMs over the internet. To enhance efficiency, quantization techniques have been employed to store a higher number of parameters per GPU, thereby decreasing the need for consecutive devices and communication rounds and use 8-bit precision to compress the weights, reducing the nodes required to store all layers. To achieve more efficient data transfer between pipeline stages, dynamic blockwise quantization is utilized. It utilize 8-bit mixed matrix decomposition for matrix multiplication allows the model to quantize the weights to 8-bit precision, significantly reducing the memory footprint compared to 16-bit weights.

QFT [75] (Section V-A4) addresses memory limitations during fine-tuning LLMs by introducing a novel quantization framework. It converts all model states into integers to minimize memory footprint and employs the Lion optimizer for its memory efficiency and compatibility with quantization. Additionally, the framework incorporates a specialized scheme for handling quantized weights during training.

QuantEase [82] is a framework for post-training quantization of LLMs that enhanced their deployment efficiency. The framework addresses the challenge of layer-wise quantization by optimizing each layer individually, utilizing Coordinate Descent (CD) to achieve high quality solutions efficiently without complex matrix operations. The framework includes an outlier-aware variant that maintains crucial ''outlier'' weights in full precision to enhance accuracy. Demonstrating SOTA performance, QuantEase significantly improves perplexity and zero-shot accuracy compared to existing methods like GPTQ [78], with up to 15% relative improvement. Efficient linear algebra optimizations allow for the quantization of large models such as Falcon-180B on a single GPU in under 3 hours. The outlier-aware variant supports near or sub-3-bit quantization with minimal accuracy loss, outperforming methods like SpQR by up to two times in perplexity reduction.

LLM-Pruner [83] (Section V-B2) compresses LLMs by removing non-essential parts based on gradient information while keeping their functionality. This significantly reduces model size with minimal accuracy loss, achieved through fine-tuning with a small amount of data.

2) PRUNING

SparseGPT [70] framework has developed an efficient and precise post-training pruning technique for significantly reducing the size of large-scale GPT-family models. This method achieves at least 50% sparsity in a single step, without requiring retraining. Remarkably, it enables the processing of the largest open-source models, such as OPT-175B and BLOOM-176B, in less than 4.5 hours. It makes the model achieve 50-60% unstructured sparsity with a negligible increase in perplexity and removes more than 100 billion weights with minimal impact on accuracy. The study demonstrates that the parameterization of massive GPT models enables pruning without relying on the gradient information. It highlights that sparse models with comparable accuracy to dense models can be identified within the ''close neighborhood'' of the dense models. The study's findings reveal that sparse models achieve performance very similar to the dense models. The study also shows that it is easier to prune larger models: for a fixed sparsity level, the accuracy drop for larger sparse models is smaller, to the point where there is practically no accuracy decrease when reaching 50% sparsity, to the point where reach 50% sparsity does not result in any noticeable accuracy decrease on the largest models.

Sheared LLAMA [72] is used to reduce the size of the LLaMA2-7B model to 1.3B and 2.7B parameters, and it performed better than other open-source models of the same

size on a variety of downstream and instruction tuning evaluations. LLM-shearing also requires only 3% of the computing resources to train as the same models trained from scratch. One of the main steps of Sheared LLAMA is a novel pruning algorithm that can prune a source model to any specified target architecture. The algorithm is an extended version of CoFiPruning that allows the source model to be pruned to any specified target architecture, based on the desired model size and performance requirements. Pre-trained models are typically well-optimized to balance expressivity and inference efficiency, therefore these configurations are used as the target architectures.

LLM-Pruner [83] introduces a framework for compressing LLMs in a task-agnostic way while minimizing the need for the original training corpus. The framework uses structural pruning to remove non-critical parts of the model based on gradient information. The pruned models' performance is recovered using LoRA [33] tuning, which takes just 3 hours and 50K data samples. Experiments on LLaMA [14], Vicuna [84], and ChatGLM [85] show that the compressed models maintain 94.97% of their original performance even after removing 20% of parameters. However, higher pruning rates lead to significant performance drops and incoherent sentence generation.

3) HYPERPARAMETER OPTIMIZATION

Selecting the right hyperparameters is essential for developing effective LLMs, as these parameters significantly influence the model's convergence speed, generalization ability, and overall performance in various language tasks. Whereas often an iterative and computationally demanding process, hyperparameter optimization is crucial for achieving optimal model performance. Cramming [16] employs a lower learning rate to stabilize the training process and prevent overfitting, enabling effective model training within limited computational resources.

C. DISTRIBUTED TRAINING

Distributed training refers to the process of training LLMs across multiple computing devices or processing units. This approach harnesses the power of parallelism to distribute the computational burden, enabling faster training of large models with millions or even billions of parameters. Distributed training is crucial for managing the massive datasets and computational demands associated with cutting-edge LLMs.

1) DATA PARALLELISM

Data parallelism is a parallel training technique that replicates the entire model across multiple GPUs or devices and distributes the training data among them. Each device handles a portion of the data, performs forward and backward propagation, and computes gradients independently. These gradients are then aggregated across all devices to update the global model parameters. It is a fundamental and widely used technique for improving the training throughput of deep learning models. Its simplicity, scalability, and effectiveness make it a valuable tool for researchers and practitioners in the field of machine learning [15], [24], [69], [80].

2) MODEL PARALLELISM

The model parallelism can be classified into two groups tensor parallelism (section V-C2a) and pipeline parallelism (section V-C2b).

a: TENSOR PARALLELISM

Tensor parallelism involves partitioning a tensor across an array of devices, necessitating a distributed matrix-matrix multiplication algorithm for mathematical computations. Using the tensor parallelism reduces the response time for individual queries [15], [17]. Megatron-LM introduced 1D tensor parallelism (Section IV-A3) which partitions the linear layer within the Transformer architecture along either the row or column dimensions. Within Megatron-LM, tensors are broken down into a single dimension [15], [19].

b: PIPELINE PARALLELISM

FlexGen [17] (Section IV-B2) utilizes pipeline parallelism to distribute an l-layer LLM evenly across m GPUs, enabling parallel execution of all layers. Each GPU executes the same sequence of operations, essentially reducing the problem to training an *n/m-layer* transformer on a single GPU. This approach leverages the existing policy search algorithm developed for single-GPU training. In order to implement micro-batch pipelining, a new repetition statement (for-loop) is used within the applied algorithm effectively merging the iteration-level pipeline parallel execution schedule with a single-device offloading runtime.

PETALS [60] (Section IV-B6) utilizes pipeline parallelism to efficiently distribute the computation of LLMs among multiple servers. Servers are organized into a chain, with each server responsible for executing a portion of the model pipeline. This approach enables efficient parallel processing, improving the overall performance of inference and fine-tuning tasks.

GPipe [3] (Section IV-A1) employs a novel pipeline parallelism algorithm based on batch splitting, where mini-batch examples are divided into smaller micro-batches and sequentially executed across cells during training. The model utilizes synchronous mini-batch gradient descent, accumulating gradients from all micro-batches and applying them to the model at the end of the mini-batch. The efficiency of the model is demonstrated through the successful training of large-scale models, including a convolutional model (AmoebaNet) for image classification and a transformer model for machine translation. The model showcases its flexibility across various deep network architectures, achieving superior results and consistent training performance on diverse devices.

DFX [86] is a low-latency multi-FPGA appliance for accelerating transformer-based text generation. It uses model parallelism to split the transformer model across multiple

FPGAs. This allows each FPGA to process a different part of the model in parallel, thereby accelerating the overall text generation process. Also, it uses an efficient network to interconnect the FPGAs and reduce communication overhead. The network uses a ring topology to minimize communication overhead. This model utilized four Xilinx Alveo U280 FPGAs and evaluated its performance on the GPT-2 language model. It demonstrated a 5.58× acceleration in speed and a 3.99× enhancement in energy efficiency compared to four NVIDIA V100 GPUs. In addition to its performance and energy efficiency benefits, this solution proves to be more cost-effective than GPU-based alternatives. Moreover, it offers an 8.21× cost advantage over a GPU appliance delivering similar performance levels.

3) COMBINED PARALLELISM

Narayanan et al.., in [69] proposed a new technique called PTD-P for training LLMs on GPU clusters. PTD-P combines pipeline parallelism, tensor parallelism, and data parallelism to achieve high computational performance and graceful scaling. Data parallelism divides the training data into smaller batches, which are then processed in parallel on all the GPU servers. This allows PTD-P to achieve faster training by leveraging the parallel computing capabilities of the GPU cluster. Also, GPipe [3], and ZeRO [18] (section IV-A1, and V-C4 respectively) are other examples of combined parallelism.

4) ZERO

ZeRO [18] proposed solutions to overcome the limitations of existing methods and efficiently train large models. While using existing systems the memory consumption can be classified into two main parts which are model states, and residual states. Most of the memory capacity is used by model states (such as momentum, variance in Adam, gradients, and parameters) while working with large models. The rest part of the memory is occupied by residual states (such as activation, temporary buffers, and unusable fragmented memory). For applying optimization in both model state memory and residual state memory, efficiently training models of such colossal sizes is crucial as they grow from billions to trillions of parameters. The study introduces a novel memory optimization technique aimed at substantially improving training speed, and with approach enables scaling the model size in proportion to the number of devices while maintaining high efficiency. Leveraging the latest hardware, this model can scale to over 1 trillion parameters by carefully evaluating communication volume and memory capacity requirements, this boosts memory efficiency for model states. For optimizing model state memory, which occupies most of the memory during training, the study introduces ZeRO-DP, ZeRO-powered data parallelism which has three main optimization stages: in the first stage, only the optimizer states are partitioned; in the second stage, both optimizer states and gradients are partitioned; and in the final stage, all three model states are partitioned. This results in a significant boost in memory efficiency. The rest of the memory consumed by residual states could become a secondary memory bottleneck. The study overcame this problem by three factors: Firstly using activation partition to optimize activation memory by locating and deleting activation replication in existing MP (model parallelism), and when appropriate offloads activations to CPU. Secondly, keeping the balance of memory and computation efficiency by introducing appropriate size temporary buffers to strike. Finally, during the training process, memory becomes fragmented because tensors have varying lifetimes. The lack of contiguous memory, resulting from this fragmentation, can lead to memory allocation failures, even when there is sufficient free memory space available. To address this problem, ZeRO-R takes a proactive approach by effectively handling memory based on the distinct lifetimes of tensors, thereby preventing memory fragmentation. Remarkably, this model achieves a throughput of 15 Petaflops when training models with over 100 billion parameters, demonstrating super-linear speedup on 400 GPUs. It indicates an 8× increase in model size and a 10× increase in performance compared to recent SOTA models.

5) SEQUENCE PARALLELISM

Sequence parallelism [15], [87], is a novel approach proposed to efficiently train Transformers with longer sequences on GPUs. It addresses the quadratic memory requirements of self-attention in Transformer models. Unlike traditional methods, it does not require a single device to handle the entire sequence. By splitting sequences into chunks and distributing them across devices, it achieves effective training with infinitely long sequences. It introduces Ring Self-Attention to enhance the process, demonstrating superior performance in batch size and sequence length compared to tensor parallelism, handling sequences over 27× longer than existing methods.

6) AUTOMATIC PARALLELISM

The automatic selection and parallelization strategies as the latest advances in parallel training demonstrated by FlexFlow [71] and Alpa [24], [88]. Alpa is an automated system that generates execution plans for distributed model-parallel training. It's an architecture that can automatically derive efficient parallel execution plans at each parallelism level. It is different from specialized systems as it can handle models with heterogeneous architectures and models without manually designed plans. However, it is not hardware-aware and does not consider network topology. Also, it does not search for activation checkpointing, which could lead to suboptimal results. Alpa has been evaluated on large models training with billions of parameters. The model's performance has been compared with the SOTA systems such as Megatron-LM [19] and DeepSpeed [5], on an Amazon EC2 cluster with 64 GPUs. It presents the similar training performance as Megatron-LM on GPT models and

outperforms DeepSpeed on GShared MoE models with up to 9.7× speedup. Moreover, it generalized well to models without manual strategies and demonstrated 80% liner scaling efficiency on Wide-ResNet. The results presented that Alpa's performance in training large models efficiently and its ability to generalize to diverse models.

D. HETEROGENEOUS TRAINING

ZeRO-Offload [20], aims to democratize large-scale model training, making it accessible to a wider audience. It achieves this by using a single GPU to train models with over 13 billion parameters, eliminating the need for data scientists to modify the models or sacrifice computational efficiency. The study introduces ZeRO-Offload, a novel heterogeneous deep learning (DL) training technology. The model leverages both CPU memory and compute for offloading and offers an efficient scaling path on multiple GPUs through collaboration with ZeRO-powered data parallelism [18]. Through first-principle analysis, the study asserts that the model provides an optimal solution, maximizing memory savings while minimizing communication and CPU compute overhead for large model training.

ZeRO-Infinity [1] introduces an innovative system technology that enables the model scaling on constrained resources. It achieves this without the need for extensive model code modifications by harnessing the power of GPU, CPU, and NVMe memory. The model made up of five innovative technologies: 1) *infinity offload engine*, this technique uses simultaneous exploitation of GPU, CPU, and NVMe memory, as well as GPU and CPU compute to fully leverage heterogeneous architecture on modern clusters, 2) *memory-centric tiling*, handle extensive operators without necessity of model parallelism, 3) *bandwidth-centric partitioning*, is employed to make the most of the aggregate memory bandwidth across all parallel devices, 4) *overlap-centric design*, is implemented to enable the simultaneous execution of compute and communication tasks, 5) *ease-inspired implementation*, to prevent the need for extensive model code refactoring. SWARM Parallelism [10] (section V-B1) introduced a model aimed at training large models efficiently, particularly on unreliable heterogeneous devices with limited network bandwidth. Instead of employing static pipelines, the model utilizes dynamically generated and randomized pipelines to adapt to varying conditions. This allows each device to share its results with any other device that is responsible for the next stage of the pipeline. This enables devices with high performance to process inputs from multiple predecessors, distribute their results across multiple weaker peers, and rebalance the workload in case of failure to improve utilization.

NLP-Fast [58] (Section IV-B3) is a system designed to enhance the performance of large-scale heterogeneous NLP models by pinpointing the most resource-intensive operations and employing a combination of techniques: holistic model partitioning, cross-operation zero skipping, and model/config adaptive hardware reconfiguration. Splitwise [63] (section IV-B9) improves LLM inference by separating workload onto different machines for high throughput, cost, or power efficiency. It allows for building both homogeneous and heterogeneous clusters depending on the optimization goal.

E. TRAINING OPTIMIZATION: CHALLENGES AND KEY FINDINGS

In the previous sections we have offered a comprehensive overview of training optimization (Section V) which includes model optimization (Section V-A), size reduction optimization (Section V-B), distributed training (Section V-C), and heterogeneous training (Section V-D). In this section and the following paragraphs, we will discuss training optimization's challenges and key findings.

Challenges of Model Optimization:

- Resource constraints: LMs demand significant memory and computational power, limiting training and deployment on single devices.
- Balancing efficiency and accuracy: Optimizing LLMs requires finding a balance between efficient resource utilization and maintaining model performance.
- Memory bottlenecks: Distributing LMs across devices introduces memory limitations on each device.
- Communication overhead: Data exchange between devices during training can become a bottleneck, slowing down the process.
- Hardware heterogeneity: Efficiently utilizing devices with varying memory capacities and processing speeds in a distributed setting is challenging.
- Scalability limitation: Traditional methods might not scale well with increasing device numbers due to memory and communication constraints.

Key Findings:

- Algorithmic: Techniques like FlexGen [17], LightSeq [61], and NLP-Fast [58] improve efficiency by optimizing computations, memory access, and utilizing specialized hardware kernels.
- Model partitioning: Techniques like GPipe [3] and Megatron-LM [19] partition models for efficient processing across multiple devices.
- Fine-tuning for efficiency: Techniques like AlphaTuning [74] and LoRA [33] enable fine-tuning large models on limited memory by reducing the number of parameters requiring adjustment.
- Scheduler optimization: Techniques like TurboTransformers [11] improve response throughput and task execution on GPUs.
- Size reduction optimization: This approach focuses on reducing model complexity through techniques like quantization (reducing storage bits) and pruning (removing non-essential parts).
- Parallelism strategies: 1) Data parallelism: Distributes training data across devices for faster training. 2) Model parallelism: Splits the model across devices for parallel computations (tensor, pipeline, sequence parallelism).

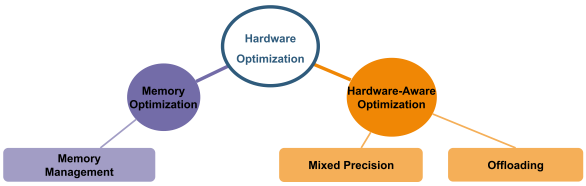

FIGURE 6. Hardware optimization.

3) Combined parallelism: Combines data and model parallelism for even faster training (PTD-P, ZeRO [18], GPipe [3]).

- Memory optimization: ZeRO [18] optimizes memory for trillions of parameters, Activation Partitioning deals with activation memory efficiently, and ZeRO-Offload [20] and ZeRO-Infinity [1], which allow training on single GPUs or limited resources by utilizing CPU and NVMe memory.
- Heterogeneous optimization: SWARM Parallelism [10] tackles unreliable devices with limited bandwidth by adapting workloads, NLP-Fast [58] optimizes execution on mixed platforms by pinpointing resource-heavy operations, and Splitwise [63] distributes work across heterogeneous machines considering different goals like throughput, cost, and power consumption.
- Automatic parallelism: Alpa [88] automatically generates execution plans for distributed model parallel training, applicable to diverse models.

Overcoming these challenges and leveraging these techniques, model training can be made more efficient, scalable, and accessible, paving the way for even more powerful and versatile LLMs.

VI. HARDWARE OPTIMIZATION

Hardware optimization is a systematic approach to improving the performance, efficiency, and functionality of computer hardware. By identifying and addressing bottlenecks in hardware architecture [18], software, and the operating system, hardware optimization can enhance overall speed, reduce power consumption, and improve the reliability of hardware components (Fig. 6). Splitwise [63] (Section IV-B9) is a technique to optimize hardware utilization by separating the prompt computation and token generation phases onto different machines. This approach allows designing clusters optimized for cost, throughput, and power consumption. The model achieves up to 1.4× higher throughput at 20% lower cost or 2.35× higher throughput with the same cost and power.

A. MEMORY OPTIMIZATION

In the process of training deep learning models, memory usage is primarily attributed to various factors, including model parameters, layer activations, gradients, and optimizer states, such as momentum and variances in the Adam algorithm [15], [18]. The terms ''model states'' [18] or ''model data'' [15] encompass model parameters, gradients, and optimizer states collectively, while ''residual states'' [18] or ''non-model data'' [15] refer to layer activations, temporary buffers, and unusable fragmented memory collectively.

In this section, we will explain the common and recent approaches that have been used for increasing training throughput and loading larger models into GPU memory while training deep learning models.

1) MEMORY MANAGEMENT

TurboTransformers [11] (section IV-B4), proposed a sequence length aware algorithm for memory allocation to efficiently balance memory allocation and deallocation, this algorithm overcomes the problem of variability of input sentence. LightSeq2 [52] introduces an innovative memory management approach, specifically designed for the Transformer structure. This strategy efficiently reduces peak memory consumption and minimizes the need for frequent allocation and release calls. Notably, LightSeq2 stands out as the pioneer in accelerating the entire training process of Transformers. In real-time applications where response time is crucial, model parallelism and pipeline parallelism can introduce significant delays due to the extra communication overhead caused by splitting tensors or layers, even with technologies like NVLink and GPUDirect. EET [62] (section IV-B8) focuses on minimizing memory usage for loading large models in online services. The proposed solution involves dynamic memory management, specifically targeting the reduction of memory consumption for activation caches and operation result buffers, as weights and certain pre-allocated caches are inherently difficult to compress. They introduce a dynamic CUDA memory management mechanism specifically designed to reduce CUDA memory usage for the same model size, unlike the manual memory allocation required by FT.

B. HARDWARE-AWARE OPTIMIZATION

Hardware-aware optimization (HAO) is the process of optimizing the hardware utilization of deep learning models to achieve maximum performance on specific hardware platforms [91]. In this section, we will explain offloading and mixed precision optimization.

1) OFFLOADING

FlexGen [17] (SectionIV-B2) presents an offloading framework for LLMs that optimizes I/O efficiency and throughput by considering computation schedule, tensor placement, and computation delegation. It utilizes a linear programming-based search algorithm and unifies the placement of weights, activations, and the KV cache, enabling significantly larger batch sizes compared to existing methods.

ZeRO-Offload [20] model facilitates the training of large model heterogeneous on GPU + CPU systems, enabling the handling of models up to 10× larger on a single GPU without sacrificing efficiency by using a unique optimal offload strategy. Also, the design achieves

TABLE 6. Comparative analysis between different strategies [A].

Technique	Performance	Cost	Scalability
FlexGen [17]	1) Batch size of 64 or 2048 tokens: Throughput speedup of 40× with a latency of 5000 seconds.2) Batch size of 256 or 8192 tokens: Throughput speedup of 79× with a latency of 12000 seconds.3) Batch size of 144 or 4608 tokens: Throughput speedup of 100× with a latency of 4000 seconds (using 4-bit quantization compression).	Enabling high-throughput LLM inference on resource-constrained devices. Minimizing the need for multiple high-end GPUs.	Efficiently running the OPT-175B model on NVIDIA T4 (16 GB) GPUs, showcasing its ability to handle LMs on resource-constrained hardware.
SwapAdvisor [68]	Memory allocation: achieves up to 4× reduction in serving latency and boosts training throughput by 20% to 1100%.	Train models up to 12× beyond GPU memory limit.	Supports efficient training and inference of LMs on standard GPUs, significantly extending their capability.
NLP-Fast [58]	Up to 2.92×, 1.59×, and 4.47× higher throughput on CPU, GPU, and FPGA respectively.	Reduce the need for high-end, resource-intensive hardware.	Scales to different hardware platforms (CPU, GPU, FPGA).
Byte Transformer [4]	Up to 87% better performance compared to other frameworks (PyTorch JIT).	Reducing redundant computations. Improves the efficiency of running inference on transformer models, potentially reducing the cost of deployment.	Scales to different sequence lengths and transformer architectures.
Sheared LLAMA [72]	Superior performance compared to other open-source models of similar size.	Reduce the size of the LLaMA2-7B model to 1.3B and 2.7B parameters. Significantly reduced training cost (3% of original).	Potentially efficient for deployment on resource-limited devices.
GrowLength [73]	Lower loss compared to fixed-length training (LLM128).	Potentially lower training cost (faster training).	Dynamically increasing training sentence length from 128 to 1024 tokens, enhancing efficiency in handling diverse text data, while maintaining lower loss.
PagedAttention [67]	PagedAttention and vLLM achieve 2-4× higher throughput in LLM serving compared to existing systems, especially for large models, long sequences, and complex decoding algorithms.	vLLM's efficiency improvements can potentially reduce deployment costs by requiring fewer servers for the same workload.	Handles large LLMs and vLLM's memory management scales well with diverse LLM architectures.
LightSeq [61]	Up to 14x and 1.4x speedups compared to TensorFlow and FasterTransformer respectively.	Reduced operational costs during deployment due to lower computational demands (inference on resource-constrained devices).	Well-suited for various transformer architectures.
LigthSeq2 [52]	Achieves 1.4-3.5× faster training compared to previous systems across various models and benchmarks, and 308% speedup on WMT14 English-German machine translation compared to PyTorch.	Potentially lower cost (by enabling faster training).	Supports various transformer architectures, including BERT, GPT, and vision transformers.
GPipe [3]	With a 557-million-parameter AmoebaNet model achieving 84.4% top-1 accuracy on ImageNet-12 dataset.	Potentially lower cost (reduced hardware requirements).	Handling large and complex models across multiple accelerators, and achieving better quality than all bilingual models.
Megatron-LM [19]	Achieves SOTA results on NLP tasks (perplexity of 10.8 on WikiText103, 66.5% accuracy on LAMBADA).	Allows utilizing fewer training instances or smaller model sizes for achieving similar performance.	Scales to train models with billions of parameters using multiple GPUs (demonstrated with 8.3B parameter model on 512 V100 GPUs).
AlphaTuning [74]	Maintains competitive performance on various tasks with over 1000× fewer parameters compared to full fine-tuning.	Reduces deployment costs by enabling less powerful hardware for inference.	Works with a wider range of LLMs, and its efficiency increases with even larger models due to quantization.
QFT [75]	Maintains similar performance across various benchmarks.	Potentially reduces deployment costs due to lower memory requirements. Allows utilizing less powerful hardware, potentially leading to lower acquisition and maintenance costs.	Handles large models with efficient memory management. Demonstrates successful fine-tuning of a 7B parameter LLaMA model, suggesting scalability for working with large language models.
LOMO [56]	Enables full parameter fine-tuning (65 billion parameter) of LLMs on limited resource GPUs.	Potentially reducing deployment costs through lower hardware acquisition and maintenance expenses.	Especially suited for handling very large models.
AdaLomo [57]	Achieves scores of 30.8, 39.7, 51.0, and 56.9 on the LLaMA benchmark for models with 7B, 13B, 30B, and 65B parameters (performance comparable to AdamW)	Potentially lower training cost (due to reduced memory requirements).	Enables LLM training on resource-constrained environments by significantly reducing memory footprint while achieving comparable performance to AdamW, especially for models with a large number of parameters.
LoRA [33]	Maintains competitive performance on various tasks compared to full fine-tuning despite significantly fewer parameters (reduction by 10,000× for GPT-3)	Reduces deployment and training costs (3× reduction for GPT-3).	Applicable to various Transformer models (e.g., RoBERTa, DeBERTa, GPT-2, GPT-3). Designed to be efficient even for extremely LMs like GPT-3 (175B parameters).
TurboTransformers [11]	Enhances latency and throughput for transformer models, achieving better speed than PyTorch and comparable performance to TensorFlow-XLA, TensorRT, and FasterTransformers.	Reduces operational costs by optimizing memory usage through a sequence-length-aware memory allocation algorithm, ensuring efficient resource utilization.	Highly scalable, efficiently handling variable-length inputs with a sequence-length-aware batch scheduler to maximize throughput in diverse deployment scenarios.
PetS [59]	Enhances serving throughput by 1.53x on Desktop GPUs and 1.63x on Server GPUs.	Reduces the cost by requiring less storage space due to their smaller size compared to traditional transformers.	Supports up to 26× more concurrent tasks compared to existing serving systems.
QMoE [77]	compresses a 1.6 trillion-parameter SwitchTransformer model to 160 GB (0.8 bits per parameter), resulting in a 20x reduction in size with minimal accuracy loss.	Achieves a 20× compression ratio, QMoE significantly reduces storage requirements.	Enables running trillion-parameter models on readily available hardware (e.g., single server with 4× NVIDIA A6000 GPUs) due to its compressed size.
SWARM Parallelism [10]	Trains large models on unreliable, heterogeneous devices with low network bandwidth.	Enables training on preemptible cloud instances or pooled resources from various regions, potentially reducing training costs compared to dedicated high-performance computing clusters.	It is designed for heterogeneous and unreliable devices, making it scalable to large deployments with varying computational power and network connectivity.

TABLE 7. Comparative analysis between different strategies [B].

Technique	Performance	Cost	Scalability
GPTQ [78]	Achieves high accuracy with post-training quantization to 3 or 4 bits per parameter.	Reduces the bit width of the weights (down to 3 or 4 bits), significantly reducing the model size.	Allows inference of large GPT models on a single GPU due to its compressed size.
FPTQ [79]	Achieves SOTA performance on popular LLMs (BLOOM, LLaMA) with 4-bit weights and 8-bit activations (W4A8).	Utilizes a 4-bit weight quantization strategy, reduces the model size compared to full precision models.	Enables deployment of LLMs on resource-constrained devices by achieving high-performance W4A8 quantization (low memory footprint) without sacrificing accuracy.
Norm Tweaking [54]	Achieves high accuracy for large language models (GLM-130B, OPT-66B) even at 2-bit quantization.	Enables effective quantization down to even 2 bits, significantly reduces the model size compared to full precision.	Allows for deploying LLMs on devices with limited memory or computational power.
FineQuant [81]	Up to 3.65× higher throughput on LLM inference with minimal accuracy loss for large models (OPT-175B).	Focuses on weight-only quantization, reduces model size efficiently, potentially enabling deployment on less powerful hardware.	Enables deployment of massive LLMs (like OPT-175B) on resource-constrained environments by achieving efficient weight-only quantization with high throughput and minimal accuracy loss.
PETALS [60]	Achieves faster inference for large language models, with an optimal setup inferring a 176 billion parameter model in 5.5 seconds.	8-bit compression reduces resource requirements.	Scales by distributing computations across a network, enabling it to handle even larger models or more inference requests simultaneously.
QuantEase [82]	Up to 15% better accuracy (perplexity, zero-shot) than GPTQ. Sub-3-bit quantization with minimal accuracy loss.	Enables effective quantization down to 3-bit or even lower precisions, significantly reducing the model size.	Quantizes large models (Falcon-180B) on 1 GPU in 3 hours.
LLM-Pruner [83]	Up to 95% performance retention with 20% parameter reduction (LLaMA, Vicuna, ChatGLM).	Potentially lower training cost due to less data needed for fine-tuning (3 hours, 50K samples).	Applicable to various LLM architectures (LLaMA, Vicuna, ChatGLM).
SparseGPT [70]	Up to 50% sparsity (weight reduction) with minimal accuracy loss (perplexity). Larger models prune more easily (with less accuracy drop).	Potentially lower computational cost due to single-shot pruning (no retraining).	Processes very large models (OPT-175B, BLOOM-176B) efficiently.
Cramming [16]	Achieves reasonable performance by training on a single GPU in one day (trade-off between model size and training time).	Lower computational cost due to single GPU training.	Not designed for large-scale training, but explores trade-offs for resource-constrained settings.
DFX [86]	5.58x speedup in text generation compared to 4x NVIDIA V100 GPUs.	8.21x more cost-effective than a GPU appliance with similar performance.	Designed for model parallelism across multiple FPGAs (scalability not explicitly quantified).
Narayanan *et al.*, [69]	High speed via pipeline, tensor, and data parallelism.	Potentially cost-efficient due to parallel processing, but not explicitly quantified.	Megatron-LM enables training LLMs (like trillion-parameter models) on thousands of GPUs by combining data, pipeline, and tensor parallelism (PTD-P) for efficient scaling and achieving high throughput.
ZeRO [18]	10x speedup, trains trillion parameter models (8x larger than previous models).	Potentially reduces memory requirements for training large models.	Scales to trillion parameter models on large GPU clusters.
Colossal-ai [15]	Up to 2.76x faster training with various parallelism methods.	Potentially lower cost due to faster training and improved hardware utilization.	Modular design for customization and supports distributed training.
FlexFlow [71]	Achieves 2-10x speedup in performance for CNN workloads compared to existing architectures.	Improves power efficiency by 2.5-10x compared to existing architectures.	Highly scalable with growing computing engine size.
Alpa [88]	Matches Megatron-LM on GPT models, surpasses DeepSpeed on GShared MoE models (up to 9.7x speedup).	Alpa automates efficient model-parallel training for large deep learning models, potentially reducing development and infrastructure costs.	Designed for distributed deep learning.
ZeRO-Offload [20]	Trains 10x larger models on single GPUs (40 TFlops/GPU for 10B parameters). Supports models over 70B parameters with model parallelism.	Potentially lower training cost due to efficient single-GPU or smaller system training.	Enables large model training on single GPUs and scales to larger systems using model parallelism.
ZeRO-Infinity [1]	Trains models with trillions of parameters on GPU clusters. Enables fine-tuning on a single DGX-2 node. Achieves over 25 petaflops (exceeds peak performance by 40%).	Potentially reduces memory requirements for training large models.	Highly scalable for training models with trillions of parameters.
Splitwise [63]	Up to 1.4x higher throughput for LLM inference compared to existing methods.	Potentially lower cost due to 20% reduction in resource requirements for inference.	Scales well using homogeneous or heterogeneous machines for prompt computation and token generation phases.
Easy and Efficient Transformer (EET) [62]	Up to 27.43x speedup in transformer inference compared to Fairseq on A100 GPUs. Significant speedups over LightSeq and Faster Transformer as well.	Potentially lower cost due to reduced inference time (potentially leading to lower resource usage).	The library is designed to work with large model sizes and potentially scales well on different hardware configurations (2080Ti and A100 GPUs are mentioned).
Eliseev and Mazur [89]	Achieves 2-3 tokens per second inference speed on consumer GPUs for large sparse MoE models (Mixtral-8x7B).	Enables running large MoE models on limited hardware (consumer GPUs and free-tier Google Colab) potentially reducing costs.	Enables running large MoEs language models (like Mixtral-8x7B) on resource-constrained hardware (consumer GPUs and even free-tier Google Colab) by leveraging MoE-specific optimizations.
FP8-LM [90]	Achieves 75% faster training speed and 39% memory reduction compared to BF16 training for a GPT-175B model on H100 GPUs (outperforms NVIDIA's Transformer Engine by 37%).	Significantly reduces training costs for large models due to faster training and lower memory usage.	Reduces memory usage by 39% and speeds up training of LLMs (like GPT-175B) by 75% through an innovative FP8 mixed-precision framework, enabling training on resource-constrained environments.

a highly scalable multi-GPU configuration by integrating the offload strategy with ZeRO-powered data parallelism, enabling ZeRO-Offload to achieve nearly linear scalability, and smooth integration with model-parallel training. This combination allows for the training of even larger models than using ZeRO-Offload or model parallelism independently. Moreover, the model enhances CPU performance by introducing a high-performance Adam optimizer, achieving a 6× improvement over SOTA Adam implementations. It also employs a one-step delayed parameter update strategy to overlap GPU forward and backward passes with CPU parameter updates. Additionally, the model's size has increased by a factor of 10 compared to widely used frameworks such as PyTorch. To maintain computational efficiency, the model minimizes data traffic to and from the GPU, increases GPU memory utilization, and allows offloading data and computation to the CPU. On a single NVIDIA V100 GPU, the model can achieve 40 TFlops/GPU for 10 billion parameters, and it can scale up to 128 GPUs when available. The model also supports model parallelism, enabling training models with more than 70 billion parameters on a single DGX-2 box, resulting in a 4.5× increase in model size compared to employing model parallelism alone.

Eliseev and Mazur [89] propose a model to efficiently run large sparse MoE language models on hardware with limited GPU memory. Using parameter offloading and leveraging the properties of MoE models enabled Mixtral-8 × 7B with mixed quantization to operate on desktop hardware and free-tier Google Colab instances. The study showed that some experts are reused between adjacent tokens, and early layers can predict subsequent experts. This led to an MoE-specific offloading strategy employing an LRU (Least Recently Used) cache and advanced prediction of needed experts. The model significantly improves speed, achieving 2-3 tokens per second on various consumer GPUs, and offers a practical solution for running large MoE models on limited hardware.

2) MIXED PRECISION

Mixed precision training [92] proposes a method for training deep neural networks using half-precision floating-point numbers, aiming to reduce memory requirements by almost half and accelerate arithmetic on modern GPUs without compromising model accuracy or requiring adjustments to hyperparameters.

Cramming [16] conducts all experiments and ablation studies using a consistent setup that employs automated mixed precision for both standard 16-bit and 32-bit floating-point precision.

LightSeq2 [52] (section IV-A4) optimizes the training process by implementing batched updates on reduced-precision parameters instead of numerous individual updates on full-precision parameters. In mixed precision training, where parameters and gradients are in FP16 during forward and backward propagation, maintaining FP32 copies is necessary for accuracy during the update values calculation. Typically, a system copies each piece of gradients, parameters to/from its FP32 counterpart in one training step, ensuring the accurate update of FP32 parameters with the loaded FP32 gradient by the trainer kernel.

FP8-LM [90] introduces a novel FP8 automatic mixed-precision framework for training LLMs, optimizing mixed-precision and distributed parallel training through three levels of FP8 utilization. By gradually incorporating 8-bit gradients, optimizer states, and distributed learning, the framework significantly enhances training efficiency. During the training of a GPT-175B model on the H100 GPU platform, the FP8 framework reduced memory usage by 39% and increased training speed by 75% compared to the BF16 framework, outperforming Nvidia's Transformer Engine by 37%. This advancement leads to substantial cost reductions for training large models and is adaptable to various tasks such as instruction tuning and reinforcement learning with human feedback.

C. HARDWARE OPTIMIZATION: CHALLENGES AND KEY FINDINGS

Challenges of Hardware Optimization:

- Memory limitation: Deep learning models can require vast amounts of memory to store parameters, activations, and gradients. This limits the size and complexity of models that can be trained on a single device.
- Limited hardware utilization: Traditional training methods may not fully utilize the capabilities of modern hardware like GPUs.
- Balancing speed and accuracy: Techniques like mixed precision training aim to improve training speed by reducing memory usage, but this can potentially compromise model accuracy.

Key Findings:

- Memory management: Techniques like sequence length aware allocation and dynamic memory management can significantly reduce memory usage during training.
- Hardware-aware optimization: Offloading computations to CPUs or leveraging mixed precision training can improve hardware utilization and training speed.
- Model parallelism: Splitting models across multiple devices can handle larger models but can introduce communication overhead, impacting training speed.
- Large model training: Frameworks like ZeRO-Offload [20] enable training models significantly larger than what a single GPU can handle.

In the domain of hardware optimization, a continuous stream of novel methodologies is emerging, demonstrably expanding the frontiers of feasibility within the training paradigm.

VII. SCALABILITY AND RELIABILITY OPTIMIZATION

Scalability optimization focuses on improving hardware systems' capacity to flexibly handle varying workloads, enabling smooth scaling adjustments to meet evolving demands [1], [5], [18], [19], [20], [69], and reliability optimization aims

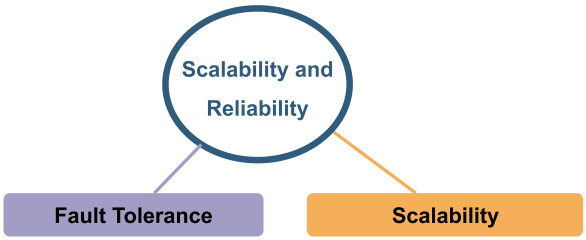

FIGURE 7. Scalability and reliability optimization.

to strengthen the dependability and stability of hardware infrastructure, reducing the likelihood of failures, errors, or disruptions [10], [60] (Fig. 7).

A. FAULT TOLERANCE

SWARM Parallelism [10] (section V-D) allows high-performance devices to handle inputs from several preceding sources, share their outcomes with less powerful peers, and adjust the workload distribution in the event of a failure, enhancing resource utilization. The model ensures continuous training and boosts overall efficiency by redistributing workload in case of device failure or premature termination.

PETALS [60] (section IV-B6) is a distributed Transformer model that can be easily scaled and fault-tolerant. It uses a load-balancing algorithm to distribute servers evenly among Transformer blocks and a routing algorithm to find the fastest path for inference. It also stores past inputs to each server in case one fails, so that the client can quickly continue with a replacement server. PETALS is a reliable and scalable Transformer model that can be used for both inference and training. It uses a combination of load balancing, routing, and fault tolerance to ensure that it can handle network disruptions and server failures without impacting performance.

B. SCALABILITY

ZeRO-Offload [20] is a highly scalable multi-GPU design achieved through an integrated offload strategy and ZeRO-powered data parallelism. This combination leads to nearly linear scalability, allowing for the training of significantly larger models than when using ZeRO-Offload or model parallelism independently. The model further optimizes CPU execution with a high-performance Adam optimizer, resulting in a 6 time higher than SOTA Adam implementation. Despite a growth in model size by a factor of 10, the approach minimizes data traffic to and from the GPU, maximizes GPU memory utilization, and facilitates offloading data and computation to the CPU. ZeRO-Offload maintains a single copy of optimizer states in CPU memory, ensuring constant communication volume and CPU computation, regardless of data parallelism. This design choice enables excellent scalability on up to 128 GPUs, and ZeRO-Offload can also be combined with model parallelism for higher memory savings when multiple GPUs are available.

C. SCALABILITY AND RELIABILITY OPTIMIZATION: CHALLENGES AND KEY FINDINGS

Challenges of Scalability and Reliability:

- In the context of optimizing LLMs, a trade-off exists between achieving high scalability and maintaining reliability. Scalability, which involves handling increased workloads, often necessitates the integration of more complex components. However, this added complexity can introduce new potential points of failure, thereby impacting the system's overall reliability. Balancing these two objectives is crucial to ensure both effective performance and robustness in large-scale deep learning systems.

Key Findings:

- Fault tolerance: This approach involves creating mechanisms to handle failures gracefully. Two notable techniques are SWARM Parallelism [10] and PETALS [60]. SWARM Parallelism distributes the workload across multiple devices and compensates for failures by redistributing tasks if a device fails. Similarly, PETALS, a distributed Transformer model, employs load balancing and routing strategies to maintain smooth operation even in the event of server failures.
- Scalability techniques: Technique like ZeRO-Offload [20] achieve high scalability for training large models. This method combines data parallelism with an offloading strategy, minimizing data traffic and maximizing resource utilization.

VIII. CASE STUDIES

The following case studies delve into the practical application of advanced optimization strategies on LLMs. With the rapid growth and increasing complexity of LLMs, efficient deployment and execution have become critical challenges. These case studies illustrate how cutting-edge techniques in model compression, pruning, and inference optimization can significantly enhance the performance and feasibility of deploying these massive models on more accessible hardware. By examining specific implementations and outcomes, these examples provide valuable insights into overcoming the computational and resource constraints associated with large-scale language models, thereby promoting their broader adoption and utility in real-world applications.

A. OPTIMIZING MODEL TRAINING WITH SPARSEGPT

Background: LLMs like GPT-3 have billions of parameters, which pose significant challenges in terms of storage, computational requirements, and energy consumption. Pruning, or removing less important parameters, can help mitigate these issues, but traditional pruning methods often require multiple iterations of fine-tuning, which is computationally expensive. This approach (SparseGPT [70]) proposes a one-shot pruning method that significantly reduces the number of parameters without the need for extensive retraining.

Context and Problem: In this case study, the focus is on training a LLM with billions of parameters on

TABLE 8. Summary on reviewed papers excluding those already covered in Tables 3 and 4 or the main text.

Studies	Aims	Outcomes
ZeRO-Infinity [1]	Effectively breaks the GPU memory barrier, making large-scale model training **accessible on constrained resources**.	Models scaled to trillions of parameters on GPU clusters, with fine-tuning possible on a single NVIDIA DGX-2 node. Consistently reach over 25 petaflops, exceeding peak performance by 40%.
ZeRO [18]	Efficiently train large models, overcome limitations of the existing methods.	Achieved 15 Petaflops during training with 100B parameter models on 400 GPUs, showing super-linear speedup. This means an 8× larger model size and a 10× performance boost compared to prior benchmarks.
SwapAdvisor [68]	An approach for deep learning memory management, enables training and serving of large models **despite limited GPU memory**.	Training models up to 12× beyond the usual GPU memory limit.
Sheared LLAMA [72]	A **dynamic batch loading**, efficiently adjusts the composition of sampled data within each training batch based on varying losses observed across different domains.	The LLaMA2-7B model is reduced to 1.3B and 2.7B parameters, needing only 3% of the usual computing resources for training. Tests were conducted using a maximum of 16 Nvidia A100 GPUs (80 GB).
GrowLength [73]	Accelerates the pre-training process of LLMs by dynamically and **progressively growing the training sentence length**.	Three different setups were investigated: LLM128 with 128-token sentences and 0.36B parameters; LLM1024 with longer sentences but the same total tokens; and GrowLength, which grows from 128 to 1024 tokens. GrowLength shows lower loss than LLM128, emphasizing its computational efficiency and practicality in resource-limited scenarios.
AlphaTuning [74]	It combines quantization of PLMs with fine-tuning, **only a subset of quantized parameters is fine-tuned for the target task**.	Applied to GPT-2 and OPT, achieved over 10x compression under 4-bit quantization and reduced trainable parameters by over 1,000-fold, maintaining competitive performance on various tasks.
Cramming [16]	Investigates the trade-offs involved in **scaling down** language model and training on a single GPU in one day.	Explored two setups: one with a classical RTX2080Ti GPU and another with modern RTXA4000 or RTXA6000 GPUs, each paired with 4 CPU cores and 32 GB RAM.
SWARM Parallelism [10]	Train a large model with **unreliable heterogeneous devices** with low network bandwidth by using dynamically generated, **randomized pipelines**.	Trained a large transformer language model with 1B shared parameters using compression strategy on preemptive T4 GPUs with network bandwidth below 200Mb/s.
GPTQ [78]	High accurate, and efficient **post-training quantization** method which is known as a new one-shot weight quantization.	Precisely quantized models to 3 or 4 bits per parameter, taking a few hours on models with hundreds of billions of parameters. Experiments on OPT-175B and BLLOM-176B, it took around 4 GPU hours with minimal loss of accuracy compared to the uncompressed baseline.
Norm Tweaking [54]	Presents a strategy to minimize computational and storage demands in large language models without compromising their performance.	Achieving high accuracy, GLM-130B and OPT-66B maintain accuracy even at 2-bit quantization. Improvements in weight-only and joint quantization surpass existing post-training quantization (PTQ) methods.
SparseGPT [70]	A **post-training pruning** method to prune massive GPT-family models efficiently and accurately.	Ran on open-source models OPT-175B and BLOOM-176B in under 4.5 hours. Achieved 50-60% unstructured sparsity with minimal impact on perplexity and removed over 100 billion weights with negligible accuracy loss.
ZeRO-Offload [20]	Democratize large-scale model training, making it accessible to a wider audience.	Trained large model heterogeneously on GPU + CPU systems, achieving 10× greater model size on a single GPU without sacrificing efficiency. Achieved 40 TFlops/GPU for 10 billion parameters on a single NVIDIA V100 GPU, scalable up to 128 GPUs. Supports model parallelism, enabling training models with over 70 billion parameters on a single DGX-2 box, resulting in a 4.5× increase in model size.
Alpa [88]	Automated system that generates execution plans for **distributed model-parallel training**.	Achieves comparable training performance to Megatron-LM on GPT models and surpasses DeepSpeed on GShared MoE models with up to 9.7× speedup.
Efficient large-scale language model training on GPU clusters using megatron-LM [69]	Introduces PTD-P, a novel technique for training LLMs on GPU clusters, combining pipeline, tensor, and data parallelism for high computational performance and scalable training.	Offers significant performance enhancements over ZeRO-3, delivering a 70% improvement for models with 175 and 530 billion parameters, mainly due to reduced cross-node communication overhead.
DFX [86]	A low-latency multi-FPGA appliance for accelerating transformer-based text generation.	Utilized four Xilinx Alveo U280 FPGAs to evaluate performance on the GPT-2 language model, achieving a 5.58× speedup and 3.99× energy efficiency improvement over four NVIDIA V100 GPUs. Demonstrated to be 8.21× more cost-effective than a GPU appliance with similar performance.
Colossal-AI [15]	A **unified deep learning system**. This system would streamline the process of **training complex models with billions of parameters on multi-GPU clusters**.	Colossal-AI is a user-friendly system that offers various parallel training techniques and integrates with advanced methods for enhanced performance. Notably, Colossal-AI achieved training speedups of up to 2.76 times for large models compared to traditional methods.
LoRA [33]	A method that improves LLM adaptation by **reducing the number of trainable parameters during fine-tuning**.	Reducing trainable parameters by 10,000x and memory usage by 3× compared to traditional methods. It maintained or improved model performance while offering faster training and efficient task-switching.
AdaLomo [57]	Aims to address the **memory limitations of existing optimizers** like AdamW by using memory-efficient techniques while retaining the benefits of adaptive learning rates.	The outcome is a successful optimizer that achieves performance comparable to AdamW on various tasks. This allows for training LLMs with significantly less memory usage, making large-scale LLM training more accessible.
Li *et al* (PagedAttention) [67]	A novel attention algorithm inspired by virtual memory. This technique aims to improve memory efficiency in LLM training by **reducing fragmentation and enabling efficient sharing.**	Achieves significant improvements in throughput (2-4×) for LLM serving, particularly for large models, complex decoding algorithms, and long sequences.
FlexFlow [71]	A novel dataflow architecture that leverages complementary parallelism effects to achieve **improved resource utilization within CNN accelerators.**	The outcome is a significant improvement in performance (2-10× speedup) and power efficiency (2.5-10×) compared to existing architectures on various CNN workloads.
QFT [75]	Develop a memory-efficient framework (QFT) for **fine-tuning LLMs.**	Achieves significant memory reduction during fine-tuning by utilizing quantization techniques, the Lion optimizer, and integer-based model states. This allows fine-tuning of large models on a single GPU with minimal performance loss compared to traditional methods.
QMoE [77]	A framework that **significantly reduces memory usage for large MoE models**.	Achieves significant memory reduction through a custom compression algorithm that shrinks models to less than 1 bit per parameter, enabling execution on affordable hardware like single servers with multiple GPUs.
FPTQ [79]	A **post-training quantization** technique for compressing LLMs.	Achieves significant memory reduction and computational efficiency during inference with minimal accuracy loss.
FineQuant [81]	A method to **improve the efficiency of LLM inference**.	Achieves significant memory reduction and faster inference speeds with minimal accuracy loss for LLMs.
QuantEase [82]	A framework for **efficiently deploying LLMs** by making them smaller.	Achieved SOTA performance in quantizing LLMs. This also improved perplexity and zero-shot accuracy by up to 15% compared to existing methods. It quantifies large models like Falcon-180B on a single GPU in 3 hours.
LLM-Pruner [83]	A task-agnostic framework for **compressing large LLMs** with minimal reliance on the original training data.	Compressed LLMs (LLaMA, Vicuna, ChatGLM) by 20% while maintaining 94.97% of their original performance.
Li *et al* [87]	A **memory-efficient method** called sequence parallelism to train Transformers with much longer sequences on GPUs.	Achieved × longer maximum sequence length and 13.7 × larger batch size compared to SOTA tensor parallelism on 64 GPUs. With sparse attention, it can handle sequences over 27 × longer than existing methods.
FP8-LM [90]	A new framework called **FP8-LM for training LLMs using mixed precision** to improve efficiency.	Reduced memory usage by 39% and increased training speed by 75% compared to the BF16 framework. It outperformed Nvidia's Transformer Engine by 37% in training speed.
LOMO [56]	A memory-efficient **training method for LLMs on limited GPU resources**.	Enables fine-tuning of massive LLMs (65 billion parameters) on consumer-grade GPUs (RTX 3090) by significantly reducing memory usage compared to traditional methods.
Eliseev and Mazur [89]	A method to run large, **sparse MoE LMs on limited GPU memory.**	This method uses parameter offloading and MoE properties to run on desktop hardware and free Google Colab instances. It leverages expert reuse and early layer prediction to achieve an MoE-specific offloading strategy. This significantly improves speed (2-3 tokens per second) on various consumer GPUs, making large MoE models practical for limited hardware.

Bold text in the "Aims" column indicates the framework's primary area of specialization or the range of tasks it is designed to address.
This table summarizes the reviewed papers excluding those already covered in Tables 3 and 4 or the main text.

limited hardware. The initial challenge was the high computational and memory requirements that exceeded the capabilities of available resources, making it difficult to efficiently train the model within a reasonable timeframe and budget.

Optimization Strategy: The primary optimization strategies involved in SparseGPT are:

One-Shot Pruning: To achieve significant sparsity in the LLM in a single pruning step, eliminating the need for iterative pruning and retraining. One-Shot Pruning: SparseGPT implements its pruning strategy through a streamlined process. First, a thorough model analysis is conducted to pinpoint parameters that can be removed without significant impact. This analysis leverages pruning criteria that assess parameter importance without requiring gradient calculations, saving on computational resources. Finally, SparseGPT employs a single step pruning approach, achieving substantial sparsity (at least 50% for massive models) in a single step. This one-shot approach significantly reduces the time and complexity compared to iterative pruning methods.

Unstructured Sparsity: To reduce the number of parameters while maintaining model accuracy through unstructured pruning, where individual weights are removed based on their importance. This approach focuses on eliminating individual weights within the model that are deemed less important. By analyzing the model's internal structure, SparseGPT achieves impressive sparsity levels of 50-60%, significantly reducing model size. This aggressive pruning strategy is remarkable because it achieves this with minimal impact on the model's ability to perform language modeling tasks accurately. For instance, SparseGPT can remove over 100 billion weights from massive models like OPT-175B and BLOOM-176B without compromising their performance on language modeling tasks.

Parametrization Without Gradient Dependence: To leverage the parametrization of massive GPT models to enable pruning without relying on gradient information. This method allows the identification of sparse counterparts within a close range of the original dense model, ensuring these sparse models maintain similar performance. Interestingly, the strategy highlights that larger models are even easier to prune using this approach. They experience minimal accuracy drops even at significant sparsity levels (e.g., 50%). This observation underscores the effectiveness of the parametrization technique in enabling aggressive pruning while preserving model performance.

Outcomes: The application of SparseGPT led to remarkable results:

- **Model size reduction:** SparseGPT achieved 50-60% sparsity, significantly reducing the model size by removing more than 100 billion weights in models like OPT-175B and BLOOM-176B.
- **Processing time:** The pruning process was completed in less than 4.5 hours for the largest open-source models, demonstrating high efficiency.
- **Accuracy maintenance:** The pruned models exhibited negligible increases in perplexity and retained performance levels very similar to their dense counterparts.
- **Scalability:** The study revealed that larger models are easier to prune, with practically no accuracy decrease observed at 50% sparsity.

This case study demonstrates the efficacy of SparseGPT's one-shot pruning approach for reducing the size of massive language models. By leveraging unstructured sparsity and parametrization strategies without gradient dependence, SparseGPT achieves substantial reductions in model size and resource requirements while maintaining high levels of performance. This approach enables more efficient and accessible deployment of large language models in various applications, making them more practical for real-world use.

B. ENHANCING INFERENCE EFFICIENCY WITH QMOE

Background: LLMs with trillions of parameters are becoming increasingly common. However, training and deploying these models is challenging due to their immense computational and memory demands. Existing compression techniques struggle to handle such large models effectively. QMoE [77] framework addresses this challenge by introducing novel compression methods to make these models more practical for real-world use.

Strategy Selection: QMoE was chosen as the optimization strategy. This approach allows for the compression of large models by quantizing their parameters to extremely low precision, which drastically reduces the model size while maintaining its performance. This strategy is particularly useful for handling the large parameter counts typical of MoE models.

Optimization Strategy: The core optimization strategies involved in QMoE are:

Scalable Compression Algorithm: QMoE tackles the challenge of massive model sizes with a scalable compression algorithm. This innovative technique achieves impressive sub-1-bit compression for trillion-parameter MoE models, without requiring retraining. In the case of the SwitchTransformer-c2048 model, this translates to a dramatic size reduction from 3.2 TB to a mere 160 GB (roughly 0.8 bits per parameter). Remarkably, this is achieved with minimal compromise on accuracy, as measured by performance on pretraining validation tasks and zero-shot data.

Customized Compression Format and GPU Kernels: QMoE takes advantage of custom designed GPU kernels to unlock the potential of its compressed format. These specialized kernels enable swift, on-the-fly decoding of the model, ensuring efficient processing during use. This allows the compressed model to run seamlessly on common hardware like 8 NVIDIA RTX 3090 or 4× NVIDIA A6000 GPUs. Even with this readily available hardware, the runtime overhead stays below 5% compared to an uncompressed model, which would require a staggering 20 times more GPUs.

Outcomes: The implementation of QMoE resulted in significant improvements:

- **Compression ratio:** The model size was reduced by approximately 95%, allowing the SwitchTransformer-c2048 model to fit within the memory constraints of standard hardware. This reduction from 3.2 TB to less than 160 GB translates to a compression ratio of around 0.8 bits per parameter.
- **Inference speed:** The QMoE framework enables the efficient execution of massive MoE models on commodity hardware with a runtime overhead of less than 5%. This efficiency allows the trillion-parameter SwitchTransformer-c2048 model to run on a single commodity GPU server.
- **Accuracy:** Despite the substantial compression, the model maintains high performance on pretraining validation tasks and zero-shot data, with only a minor decline in accuracy.

This case study demonstrates the feasibility of deploying trillion-parameter models in real-world applications through the use of advanced compression techniques. The QMoE approach not only reduces resource requirements but also enhances the deployability of cutting-edge language models across various environments. By leveraging a scalable compression algorithm, a customized compression format, and bespoke GPU kernels, QMoE achieves significant improvements in model efficiency and performance. This makes large-scale models more accessible and practical for real-world applications. It addresses key limitations of MoE architectures and promotes their wider adoption, paving the way for further research and advancements in this field.

IX. DISCUSSION

This section examines optimization and acceleration techniques for LLMs. We will discuss the relevant libraries and frameworks that facilitate these advancements, alongside challenges and key findings of various optimization strategies.

A. LLM TRAINING CHALLENGES

Training LLMs poses significant challenges due to their complexity and resource requirements. Recent advancements in frameworks like GPipe [3], ByteTransformer [4], Megatron-LM [19], LightSeq2 [52], and CoLLiE [53] have made significant strides in addressing these challenges:

Distributed Training: As LLMs become increasingly complex, training them on a single device becomes impractical. Megatron-LM [19] and CoLLiE [53] address this by employing distributed training algorithms that partition the model across multiple GPUs. This approach enables parallel processing and significantly accelerates training times. By distributing the workload, these frameworks mitigate the memory bottlenecks that arise when trying to train massive models on single devices.

Efficiency and Speed: Efficiency and speed are critical for the practical deployment of LLMs. LightSeq2 [52] enhances training speed through system-level optimizations such as layer-specific kernels and mixed-precision training, which improve GPU utilization and reduce memory usage. Similarly, ByteTransformer [4] is designed to accelerate transformer models, particularly for variable-length inputs in NLP tasks, thereby improving performance and reducing latency.

Memory Management: Efficient memory allocation is crucial for training large models. CoLLiE [53] addresses memory constraints in LLM training through a comprehensive strategy. It implements 3D parallelism to effectively distribute memory across training machines and GPUs. This approach allows CoLLiE to train large language models even in environments with limited resources.

Fine-Tuning and Performance: CoLLiE [53] also focuses on enhancing specific capabilities of LLMs through PEFT methods. These methods allow models to be fine-tuned for particular tasks or user instructions without compromising their overall performance. This targeted improvement is vital for developing models that can adapt to specific application needs while maintaining high general performance.

B. LLM TRAINING KEY FINDINGS

The advancements in these frameworks have led to several significant findings:

GPipe: Demonstrates the successful training of a large multilingual transformer model, achieving superior results compared to smaller, individually trained models [3].

ByteTransformer: Outperforms existing frameworks in terms of performance for BERT-like transformers on various benchmarks [4].

Megatron-LM: Enabled the training of LLMs with billions of parameters, achieving SOTA results on numerous NLP tasks while providing high throughput [19].

LightSeq2: Accelerates transformer model training by up to 308%, showcasing substantial performance improvements [52].

CoLLiE: Introduces collaborative training methodologies that improved efficiency and effectiveness in training large models like LLaMA-65B, exploring ways to enhance specific functionalities without impacting overall performance [53].

C. LLM INFERENCE CHALLENGES

Efficient inference of LLMs is critical for their practical application, as these models are computationally expensive due to their size and complexity. In this section, we will discuss and explore the challenges and key findings of various frameworks and libraries designed to enhance the efficiency of LLM inference.

Computational Expense: The massive size and complex architecture of LLMs make traditional inference methods inefficient, especially on resource-constrained devices.

Balancing Speed, Accuracy, and Resource Utilization: Achieving an optimal balance between these factors are crucial for real-world deployment of LLMs.

D. LLM INFERENCE KEY FINDINGS

Hardware Specialization: Frameworks like Splitwise [63] improve inference by separating compute-intensive and memory-intensive phases onto different machines with specialized hardware. This targeted approach optimizes resource usage and enhances performance.

Resource Optimization: FlexGen [17] employs techniques such as I/O scheduling, compression, and distributed processing to efficiently utilize resources across CPUs, GPUs, and disk storage. This holistic resource management approach significantly improves inference efficiency.

Algorithmic Optimizations: Libraries like EET [62] and LightSeq [61] implement custom algorithms and advanced memory management techniques to accelerate inference on GPUs. These optimizations reduce latency and improve throughput, making LLM inference more practical for real-time applications.

Heterogeneous Platforms: NLP-Fast [58] leverages different hardware platforms, including CPUs, GPUs, and FPGAs, by identifying performance-critical operations and applying targeted optimizations. This flexibility allows for efficient inference across various hardware configurations.

Distributed Inference: PETALS [60] facilitates collaborative inference and fine-tuning of LLMs across a network, enabling scalable and efficient resource utilization. This approach allows for distributed processing, which is essential for handling large-scale inference tasks.

E. LLM DEPLOYMENT AND SERVING CHALLENGES

Deploying and serving LLMs in real-world applications presents several challenges. This section explores these challenges, key findings from recent advancements, and future directions for making LLM deployment and serving more efficient and accessible.

Memory Limitation: LLMs often exceed the memory capacity of a single GPU, complicating their deployment and serving in practical applications.

Scalability: Handling multiple user requests simultaneously requires efficient scaling solutions to manage the large and complex models effectively.

Variability of Input: LLM performance can be inconsistent when dealing with input sequences of varying lengths, necessitating dynamic memory allocation strategies to maintain efficiency.

Ease of Deployment: Integrating complex LLM serving systems into existing workflows can be challenging, particularly for researchers and practitioners without extensive expertise in the field.

F. LLM DEPLOYMENT AND SERVING KEY FINDINGS

PagedAttention (vLLM): This algorithm breaks down the KV cache into manageable blocks, minimizing wasted memory and enabling efficient sharing across requests. This is a significant improvement for processing large LLMs [67].

Efficient GPU Utilization (TurboTransformers): Utilizes techniques like parallel GPU kernels and dynamic batch scheduling to optimize performance on GPUs, resulting in faster inference for transformer-based models [11].

System-Level Optimizations (LightSeq2): Demonstrates how system-level optimizations within the training process can significantly improve training speed and efficiency, translating to faster deployment of LLMs [52].

G. HARDWARE OPTIMIZATION IN LLM

Optimizing hardware for LLM involves overcoming memory limitations and improving utilization. Key findings include efficient memory management, hardware-aware optimization, and model parallelism. Future research should focus on efficient offloading strategies and advanced mixed precision training.

H. SCALABILITY AND RELIABILITY OPTIMIZATION IN HARDWARE SYSTEMS

Achieving scalable and reliable hardware systems requires balancing complexity with reliability. Techniques like SWARM parallelism and ZeRO-Offload [20] improve fault tolerance and scalability. Future research should develop advanced fault tolerance mechanisms and optimize for new hardware.

These advancements collectively enhance the efficiency, scalability, and accessibility of LLM training, inference, deployment, and serving, paving the way for more powerful language models.

X. CONCLUSION AND FUTURE DIRECTIONS

This SLR investigated optimization and acceleration techniques for LLMs. We identified the challenges associated with training, inference, and system serving for LLM with billion or trillion parameters. We presented a structured taxonomy of optimization techniques alongside a comprehensive analysis of recent libraries and frameworks. Following the PRISMA statement, we meticulously analyzed 65 relevant studies published between 2017 and December 2023. Our proposed taxonomy provides a roadmap for researchers to navigate the diverse landscape of optimization strategies and select the most suitable approaches for their specific tasks. Additionally, the review of libraries and frameworks empowers researchers to efficiently train and deploy LLMs, accelerating progress in real-world applications. Furthermore, the inclusion of two in-depth case studies demonstrates practical approaches to optimizing model training and enhancing inference efficiency, highlighting how resource limitations can be addressed while maintaining performance.

While recent advancements in LLM frameworks and optimization techniques are promising, further research is crucial to unlock their full potential. We identified several key areas for future exploration, focusing on enhanced efficiency, scalability, and flexibility for LLMs.

A. OPTIMIZATION FOR RESOURCE-CONSTRAINED ENVIRONMENTS

Hybrid Processing: Develop hybrid processing techniques, where computation is split between GPUs and CPUs to optimize memory usage and computational load.

Efficient Offloading Mechanisms: Extend the capabilities of models like FlexGen [17] and DeepSpeed Inference [5] by refining offloading techniques. This includes better utilization of CPU, GPU, and NVMe memory to handle larger models with fewer resources.

Resource-Aware Scheduling: Implement intelligent scheduling mechanisms that consider the specific resource constraints of the hardware, optimizing the allocation of GPU, CPU, and memory resources for different types of tasks.

B. MEMORY AND COMPUTATION OPTIMIZATION

Advanced Memory Management: Implement various techniques like dynamic catching, memory recycling, and efficient layer normalization (as presented in ByteTransformer [4] and LightSeq2 [52]) to overcome the memory overhead problem.

Mixed-Precision Training In order to significantly reduce training time and resource consumption without sacrificing accuracy, develop robust mixed-precision methods (like Megatron-LM [19] and LightSeq2 [52]).

Dynamic Input Handling: Focusing on variable-length inputs, like ByteTransformer [4], is seen as a promising area for improvement in ML, especially for NLP tasks that often deal with data of varying lengths. By developing more advanced algorithms to handle these inputs and minimize unnecessary computations, frameworks could achieve significant performance gains in NLP.

C. PARALLELISM AND DISTRIBUTION

Adaptive Parallelism: Develop more advanced techniques that can dynamically adapt the parallelism strategy based on the model size and hardware configuration. This includes both data and model parallelism that can be adjusted on-the-fly to optimize performance.

Distributed Training and Inference: Improve frameworks like PETALS [60] and CoLLiE [53] to better leverage distributed and heterogeneous hardware resources for efficient training and inference.

D. SCALABLE AND MODULAR ARCHITECTURE

Composable Frameworks: Design frameworks with modular components, similar to NLP-Fast [58]. These components act like building blocks for inference pipelines. Users can easily swap or optimize individual components independently, allowing for greater flexibility and customization.

Flexible APIs: Create user-friendly APIs, like those in PETALS [60]. These APIs allow users to customize inference and fine-tuning processes according to their specific needs without having to make extensive changes to the underlying framework. This provides greater control and adaptability for different use cases.

E. PERFORMANCE OPTIMIZATION TECHNIQUES

Adaptive Algorithms: Develop algorithms that can adapt to varying input sizes and sequences, optimizing both memory allocation and computational load dynamically.

Custom Kernel Implementations: Continue to develop and refine custom kernel implementations for key operations like Softmax and LayerNorm to achieve better performance, as seen in TurboTransformers [11]. This could also involve hardware-specific optimizations for different GPU architectures.

F. ADVANCED COMPRESSION AND QUANTIZATION

Sophisticated Compression Techniques: To reduce model size without significant accuracy loss instigate new methods for both lossless and lossy compression going beyond FlexGen's 4-bit quantization [17].

Dynamic Quantization: Develop dynamic quantization techniques that adjust the precision of weights and activations in real time based on the computational requirements and available resources.

XI. LIMITATIONS

m In this section, we will present the limitations of our SLR. Here, we acknowledge that while our review offers valuable insights, it is essential to consider its scope and boundaries. The limitations of our SLR can be stated as follows:

Timeframe: This SLR focused on studies published between 2017 and December 2023. While this timeframe deliberately captured a period of significant advancement in LLM optimization techniques, it is acknowledged that relevant research published before 2017 or after December 2023 might have been excluded. This could potentially limit the comprehensiveness of the analysis, particularly regarding foundational concepts or emerging advancements outside the chosen timeframe.

Search Strategy: The chosen search queries might not have encompassed all possible relevant terminology used in LLM optimization research. This limitation could result in missing out on studies that use different terminologies or keywords to describe similar concepts and techniques.

Database Coverage: If the search excluded specific databases that are highly relevant to LLM research, significant studies might have been overlooked. Comprehensive database coverage is crucial to ensure the inclusion of all pertinent research.

LIST OF ABBREVIATIONS

AdaLomo	Low-Memory Optimization with Adaptive Learning Rate
BART	Bidirectional and Auto-Regressive Transformers
BERT	Bidirectional Encoder Representations from Transformers

BLOOM	BigScience Large Open-science Open-access Multilingual Language Model
CD	Coordinate Descent
EET	Easy and Efficient Transformer
FPGA	Field Programmable Gate Arrays
FPTQ	Fine-Grained Post-Training Quantization
FT	Faster Transformer
GLM	General Language Model
GPT	Generative Pre-trained Transformer
GPU	Graphical Processing Unit
HAO	Hardware-Aware Optimization
IR	Information Retrieval
KV	Key Value
LAMBADA	LAnguage Modeling Broadened to Account for Discourse Aspects
LLaMA	Large Language Model Meta AI
LLM-QAT	LLM-Quantization-Aware Training
LM	Language Model
LOMO	Low-Memory Optimization
LoRA	Low-Rank Adaptation
MHA	Multi-Head Attention
MoE	Mixture-of-Experts
MMLU	Massive Multitask Language Understanding
NLP	Natural Language Processing
NN	Neural Network
OPT	Open Pre-trained Transformer
PET	Parameter Efficient Transformers
PetS	Parameter-Efficient Transformers Serving
PEFT	Parameter-Efficient Fine-Tuning
PIE	PET Inference Engine
PLM	Pre-trained Language Model
PRISMA	Preferred Reporting Items for Systematic Reviews and Meta-Analyses
PTM	Pre-Trained Model
PTQ	Post-Training Quantization
SLR	Systematic Literature Review
SWARM	Stochastically Wired Adaptively Rebalanced Model
VAE	Variational Autoencoder
W4A8	4-bit weights and 8-bit activations

ACKNOWLEDGMENT

The authors are grateful to the members of the Applied Machine Learning Research Group of Óbuda University John von Neumann Faculty of Informatics for constructive comments and suggestions. They would also like to acknowledge the support of the Doctoral School of Applied Informatics and Applied Mathematics of Óbuda University.

REFERENCES

[1] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, "ZeRO-infinity: Breaking the GPU memory wall for extreme scale deep learning," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2021, pp. 1–15.

[2] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "Pre-trained models for natural language processing: A survey," *Sci. China Technol. Sci.*, vol. 63, no. 10, pp. 1872–1897, 2020.

[3] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, and Y. Wu, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–20.

[4] Y. Zhai, C. Jiang, L. Wang, X. Jia, S. Zhang, Z. Chen, X. Liu, and Y. Zhu, "ByteTransformer: A high-performance transformer boosted for variable-length inputs," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2023, pp. 344–355.

[5] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley, and Y. He, "DeepSpeed-inference: Enabling efficient inference of transformer models at unprecedented scale," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2022, pp. 1–15.

[6] Y. Gong, "Multilevel large language models for everyone," 2023, *arXiv:2307.13221*.

[7] B. Spector and C. Re, "Accelerating LLM inference with staged speculative decoding," 2023, *arXiv:2308.04623*.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[9] L. Torbarina, T. Ferkovic, L. Roguski, V. Mihelcic, B. Sarlija, and Z. Kraljevic, "Challenges and opportunities of using transformer-based multi-task learning in NLP through ML lifecycle: A survey," 2023, *arXiv:2308.08234*.

[10] M. Ryabinin, T. Dettmers, M. Diskin, and A. Borzunov, "Swarm parallelism: Training large models can be surprisingly communication-efficient," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 29416–29440.

[11] J. Fang, Y. Yu, C. Zhao, and J. Zhou, "TurboTransformers: An efficient GPU serving system for transformer models," in *Proc. 26th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, Feb. 2021, pp. 389–402.

[12] R. Anil et al., "PaLM 2 technical report," 2023, *arXiv:2305.10403*.

[13] L. J. Laki and Z. G. Yang, "Sentiment analysis with neural models for Hungarian," *Acta Polytechnica Hungarica*, vol. 20, no. 5, pp. 109–128, 2023.

[14] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.

[15] S. Li, H. Liu, Z. Bian, J. Fang, H. Huang, Y. Liu, B. Wang, and Y. You, "Colossal-AI: A unified deep learning system for large-scale parallel training," in *Proc. 52nd Int. Conf. Parallel Process.*, Aug. 2023, pp. 766–775.

[16] J. Geiping and T. Goldstein, "Cramming: Training a language model on a single gpu in one day," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 11117–11143.

[17] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, D. Y. Fu, Z. Xie, B. Chen, C. Barrett, J. E. Gonzalez, P. Liang, C. Ré, I. Stoica, and C. Zhang, "FlexGen: High-throughput generative inference of large language models with a single GPU," 2023, *arXiv:2303.06865*.

[18] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "ZeRO: Memory optimizations toward training trillion parameter models," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2020, pp. 1–16.

[19] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2019, *arXiv:1909.08053*.

[20] J. Ren, S. Rajbhandari, R. Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He, "ZeRO-offload: Democratizing billion-scale model training," in *Proc. USENIX Annu. Tech. Conf.*, 2021, pp. 551–564.

[21] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," 2016, *arXiv:1604.06174*.

[22] B. Yuan, Y. He, J. Davis, T. Zhang, T. Dao, B. Chen, P. S. Liang, C. Re, and C. Zhang, "Decentralized training of foundation models in heterogeneous environments," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 25464–25477.

[23] M. Ryabinin and A. Gusev, "Towards crowdsourced training of large neural networks using decentralized mixture-of-experts," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 3659–3672.

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

[25] M. Shah Jahan, H. U. Khan, S. Akbar, M. Umar Farooq, S. Gul, and A. Amjad, "Bidirectional language modeling: A systematic literature review," *Sci. Program.*, vol. 2021, pp. 1–15, May 2021.

[26] F. Yu, D. Wang, L. Shangguan, M. Zhang, X. Tang, C. Liu, and X. Chen, "A survey of large-scale deep learning serving system optimization: Challenges and opportunities," 2021, *arXiv:2111.14247*.

[27] G. Bai, Z. Chai, C. Ling, S. Wang, J. Lu, N. Zhang, T. Shi, Z. Yu, M. Zhu, Y. Zhang, C. Yang, Y. Cheng, and L. Zhao, "Beyond efficiency: A systematic survey of resource-efficient large language models," 2024, *arXiv:2401.00625*.

[28] H. Wang, Z. Qu, Q. Zhou, H. Zhang, B. Luo, W. Xu, S. Guo, and R. Li, "A comprehensive survey on training acceleration for large machine learning models in IoT," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 939–963, Jan. 2022.

[29] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth, "Recent advances in natural language processing via large pre-trained language models: A survey," *ACM Comput. Surveys*, vol. 56, no. 2, pp. 1–40, Feb. 2024.

[30] Z. Wan, X. Wang, C. Liu, S. Alam, Y. Zheng, J. Liu, Z. Qu, S. Yan, Y. Zhu, Q. Zhang, M. Chowdhury, and M. Zhang, "Efficient large language models: A survey," 2023, *arXiv:2312.03863*.

[31] V. Cole and M. Boutet, "Researchrabbit," *J. Can. Health Libraries Assoc.*, vol. 44, no. 2, p. 43, 2023.

[32] M. Ouzzani, H. Hammady, Z. Fedorowicz, and A. Elmagarmid, "Rayyan—A Web and mobile app for systematic reviews," *Systematic Rev.*, vol. 5, no. 1, pp. 1–10, Dec. 2016.

[33] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," 2021, *arXiv:2106.09685*.

[34] J. Gao and C.-Y. Lin, "Introduction to the special issue on statistical language modeling," *ACM Trans. Asian Lang. Inf. Process.*, vol. 3, no. 2, pp. 87–93, Jun. 2004.

[35] A. Pauls and D. Klein, "Faster and smaller N-gram language models," in *Proc. 49th Annu. Meeting Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2011, pp. 258–267.

[36] S. M. Thede and M. P. Harper, "A second-order hidden Markov model for part-of-speech tagging," in *Proc. 37th Annu. Meeting Assoc. Comput. Linguistics Comput. Linguistics*, 1999, pp. 175–182.

[37] M. U. Hadi, R. Qureshi, A. Shah, M. Irfan, A. Zafar, M. B. Shaikh, N. Akhtar, J. Wu, and S. Mirjalili, "Large language models: A comprehensive survey of its applications, challenges, limitations, and future prospects," 2023, doi: 10.36227/techrxiv.23589741.v4.

[38] M. Crawford, T. M. Khoshgoftaar, J. D. Prusa, A. N. Richter, and H. A. Najada, "Survey of review spam detection using machine learning techniques," *J. Big Data*, vol. 2, no. 1, pp. 1–24, Dec. 2015.

[39] A. López-Chau, D. Valle-Cruz, and R. Sandoval-Almazán, "Sentiment analysis of Twitter data through machine learning techniques," *Softw. Eng. Era Cloud Comput.*, vol. 1, pp. 185–209, 2020.

[40] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12 pp. 2493–2537, Aug. 2011.

[41] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 26, 2013, pp. 1–17.

[42] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, *arXiv:1301.3781*.

[43] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 13, 2000, pp. 1–11.

[44] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–16.

[45] T. A. Chang and B. K. Bergen, "Language model behavior: A comprehensive survey," 2023, *arXiv:2303.11504*.

[46] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus, "Emergent abilities of large language models," 2022, *arXiv:2206.07682*.

[47] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[48] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," 2019, *arXiv:1910.13461*.

[49] R. Taylor, M. Kardas, G. Cucurull, T. Scialom, A. Hartshorn, E. Saravia, A. Poulton, V. Kerkez, and R. Stojnic, "Galactica: A large language model for science," 2022, *arXiv:2211.09085*.

[50] M. Shanahan, "Talking about large language models," *Commun. ACM*, vol. 67, no. 2, pp. 68–79, Feb. 2024.

[51] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke, "TensorFlow-serving: Flexible, high-performance ML serving," 2017, *arXiv:1712.06139*.

[52] X. Wang, Y. Wei, Y. Xiong, G. Huang, X. Qian, Y. Ding, M. Wang, and L. Li, "LightSeq2: Accelerated training for transformer-based models on GPUs," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2022, pp. 1–14.

[53] K. Lv, S. Zhang, T. Gu, S. Xing, J. Hong, K. Chen, X. Liu, Y. Yang, H. Guo, T. Liu, Y. Sun, Q. Guo, H. Yan, and X. Qiu, "CoLLiE: Collaborative training of large language models in an efficient way," in *Proc. Conf. Empirical Methods Natural Lang. Process., Syst. Demonstrations*, 2023, pp. 527–542.

[54] L. Li, Q. Li, B. Zhang, and X. Chu, "Norm tweaking: High-performance low-bit quantization of large language models," 2023, *arXiv:2309.02784*.

[55] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," 2019, *arXiv:1909.11942*.

[56] K. Lv, Y. Yang, T. Liu, Q. Gao, Q. Guo, and X. Qiu, "Full parameter fine-tuning for large language models with limited resources," 2023, *arXiv:2306.09782*.

[57] K. Lv, H. Yan, Q. Guo, H. Lv, and X. Qiu, "AdaLomo: Low-memory optimization with adaptive learning rate," 2023, *arXiv:2310.10195*.

[58] J. Kim, S. Hur, E. Lee, S. Lee, and J. Kim, "NLP-fast: A fast, scalable, and flexible system to accelerate large-scale heterogeneous NLP models," 2017, *arXiv:1712.06139*.

[59] Z. Zhou, X. Wei, J. Zhang, and G. Sun, "PetS: A unified framework for parameter-efficient transformers serving," in *Proc. USENIX Annu. Tech. Conf.*, 2022, pp. 489–504.

[60] A. Borzunov, D. Baranchuk, T. Dettmers, M. Ryabinin, Y. Belkada, A. Chumachenko, P. Samygin, and C. Raffel, "Petals: Collaborative inference and fine-tuning of large models," 2022, *arXiv:2209.01188*.

[61] X. Wang, Y. Xiong, Y. Wei, M. Wang, and L. Li, "LightSeq: A high performance inference library for transformers," 2020, *arXiv:2010.13887*.

[62] G. Li, Y. Xi, J. Ding, D. Wang, B. Liu, C. Fan, X. Mao, and Z. Zhao, "Easy and efficient transformer: Scalable inference solution for large NLP model," 2021, *arXiv:2104.12470*.

[63] P. Patel, E. Choukse, C. Zhang, A. Shah, Ì. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative LLM inference using phase splitting," 2023, *arXiv:2311.18677*.

[64] H. Zhang, A. Ning, R. Prabhakar, and D. Wentzlaff, "A hardware evaluation framework for large language model inference," 2023, *arXiv:2312.03134*.

[65] Y. Song, Z. Mi, H. Xie, and H. Chen, "PowerInfer: Fast large language model serving with a consumer-grade GPU," 2023, *arXiv:2312.12456*.

[66] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv:1907.11692*.

[67] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with PagedAttention," in *Proc. 29th Symp. Operating Syst. Princ.*, Oct. 2023, pp. 611–626.

[68] C.-C. Huang, G. Jin, and J. Li, "SwapAdvisor: Pushing deep learning beyond the GPU memory limit via smart swapping," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Mar. 2020, pp. 1341–1355.

[69] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on GPU clusters using megatron-LM," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2021, pp. 1–14.

[70] E. Frantar and D. Alistarh, "Sparsegpt: Massive language models can be accurately pruned in one-shot," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 10323–10337.

[71] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 553–564.
[72] M. Xia, T. Gao, Z. Zeng, and D. Chen, "Sheared LLaMA: Accelerating language model pre-training via structured pruning," 2023, *arXiv:2310.06694*.
[73] H. Jin, X. Han, J. Yang, Z. Jiang, C.-Y. Chang, and X. Hu, "GrowLength: Accelerating LLMs pretraining by progressively growing training length," 2023, *arXiv:2310.00576*.
[74] S. Jung Kwon, J. Kim, J. Bae, K. Min Yoo, J.-H. Kim, B. Park, B. Kim, J.-W. Ha, N. Sung, and D. Lee, "AlphaTuning: Quantization-aware parameter-efficient adaptation of large-scale pre-trained language models," 2022, *arXiv:2210.03858*.
[75] Z. Li, X. Liu, B. Zhu, Z. Dong, Q. Gu, and K. Keutzer, "QFT: Quantized full-parameter tuning of LLMs with affordable resources," 2023, *arXiv:2310.07147*.
[76] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," 2020, *arXiv:2001.08361*.
[77] E. Frantar and D. Alistarh, "QMoE: Practical sub-1-bit compression of trillion-parameter models," 2023, *arXiv:2310.16795*.
[78] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate post-training quantization for generative pre-trained transformers," 2022, *arXiv:2210.17323*.
[79] Q. Li, Y. Zhang, L. Li, P. Yao, B. Zhang, X. Chu, Y. Sun, L. Du, and Y. Xie, "FPTQ: Fine-grained post-training quantization for large language models," 2023, *arXiv:2308.15987*.
[80] T. L. Scao, "BLOOM: A 176B-parameter open-access multilingual language model," 2022, *arXiv:2211.05100*.
[81] Y. Jin Kim, R. Henry, R. Fahim, and H. Hassan Awadalla, "FineQuant: Unlocking efficiency with fine-grained weight-only quantization for LLMs," 2023, *arXiv:2308.09723*.
[82] K. Behdin, A. Acharya, A. Gupta, Q. Song, S. Zhu, S. Keerthi, and R. Mazumder, "QuantEase: Optimization-based quantization for language models," 2023, *arXiv:2309.01885*.
[83] X. Ma, G. Fang, and X. Wang, "LLM-Pruner: On the structural pruning of large language models," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2023, pp. 21702–21720.
[84] B. Peng, C. Li, P. He, M. Galley, and J. Gao, "Instruction tuning with GPT-4," 2023, *arXiv:2304.03277*.
[85] A. Zeng, X. Liu, Z. Du, Z. Wang, H. Lai, M. Ding, Z. Yang, Y. Xu, W. Zheng, X. Xia, W. Lam Tam, Z. Ma, Y. Xue, J. Zhai, W. Chen, P. Zhang, Y. Dong, and J. Tang, "GLM-130B: An open bilingual pre-trained model," 2022, *arXiv:2210.02414*.
[86] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "DFX: A low-latency multi-FPGA appliance for accelerating transformer-based text generation," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2022, pp. 616–630.
[87] S. Li, F. Xue, C. Baranwal, Y. Li, and Y. You, "Sequence parallelism: Long sequence training from system perspective," 2022, *arXiv:2105.13120*.
[88] L. Zheng, Z. Li, H. Zhang, Y. Zhuang, Z. Chen, Y. Huang, Y. Wang, Y. Xu, D. Zhuo, E. P. Xing, J. E. Gonzalez, and I. Stoica, "Alpa: Automating inter- and intra-operator parallelism for distributed deep learning," in *Proc. 16th USENIX Symp. Operating Syst. Design Implement.*, 2022, pp. 559–578.
[89] A. Eliseev and D. Mazur, "Fast inference of mixture-of-experts language models with offloading," 2023, *arXiv:2312.17238*.
[90] H. Peng, K. Wu, Y. Wei, G. Zhao, Y. Yang, Z. Liu, Y. Xiong, Z. Yang, B. Ni, J. Hu, R. Li, M. Zhang, C. Li, J. Ning, R. Wang, Z. Zhang, S. Liu, J. Chau, H. Hu, and P. Cheng, "FP8-LM: Training FP8 large language models," 2023, *arXiv:2310.18313*.
[91] Z. Dong, Y. Gao, Q. Huang, J. Wawrzynek, H. K. H. So, and K. Keutzer, "HAO: Hardware-aware neural architecture optimization for efficient inference," in *Proc. IEEE 29th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2021, pp. 50–59.
[92] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," 2017, *arXiv:1710.03740*.

ZHYAR RZGAR K. ROSTAM (Member, IEEE) received the B.Sc. and M.Sc. degrees in computer science from the University of Sulaimani, KRG-Iraq, in 2013 and 2019, respectively. He is currently pursuing the Ph.D. degree in information science and technology with Óbuda University, Budapest, Hungary. His current research interests include large language models, scientific text classification, deep learning techniques, machine learning algorithms, and artificial intelligence.

SÁNDOR SZÉNÁSI (Member, IEEE) received the Ph.D. degree from the Doctoral School of Applied Informatics and Applied Mathematics, Óbuda University, Budapest, Hungary, in 2013. Currently, he is a Professor with the John von Neumann Faculty of Informatics, Óbuda University. His research interests include (data) parallel algorithms, GPU programming, and medical image processing. He engages both in theoretical fundamentals and in algorithmic issues with respect to realization of practical requirements and given constraints.

GÁBOR KERTÉSZ (Senior Member, IEEE) received the Ph.D. degree in information science and technology, in 2019.

He is currently an Associate Professor and the Vice-Dean for Research with the John von Neumann Faculty of Informatics, Óbuda University, Budapest, Hungary, and also a part-time Research Fellow with the HUN-REN SZTAKI (Institute for Computer Science and Control). He is the Leader of the Applied Machine Learning Research Group, the John von Neumann Faculty of Informatics; the main areas of his research were computer vision, parallel processing, and deep machine learning. His current research interests include distributed deep learning, metric learning, and applied machine intelligence.

Dr. Kertész is the Founding President of the High Performance Computing Division, John von Neumann Computer Society, and the President of the IEEE Computational Intelligence Hungary Chapter.