Deploying Foundation Model Powered Agent Services: A Survey

Wenchao Xu, Jinyu Chen, Peirong Zheng, Xiaoquan Yi, Tianyi Tian, Wenhui Zhu, Quan Wan, Haozhao Wang, *Member, IEEE*, Yunfeng Fan, Qinliang Su, and Xuemin Shen, *Fellow, IEEE*

***Abstract*—Foundation model (FM) powered agent services are regarded as a promising solution to develop intelligent and personalized applications for advancing toward Artificial General Intelligence (AGI). To achieve high reliability and scalability in deploying these agent services, it is essential to collaboratively optimize computational and communication resources, thereby ensuring effective resource allocation and seamless service delivery. In pursuit of this vision, this paper proposes a unified framework aimed at providing a comprehensive survey on deploying FM-based agent services across heterogeneous devices, with the emphasis on the integration of model and resource optimization to establish a robust infrastructure for these services. Particularly, this paper begins with exploring various low-level optimization strategies during inference and studies approaches that enhance system scalability, such as parallelism techniques and resource scaling methods. The paper then discusses several prominent FMs and investigates research efforts focused on inference acceleration, including techniques such as model compression and token reduction. Moreover, the paper also investigates critical components for constructing agent services and highlights notable intelligent applications. Finally, the paper presents potential research directions for developing real-time agent services with high Quality of Service (QoS).**

***Index Terms*—Foundation model, AI agent, cloud/edge computing, serving system, distributed system, AGI.**

Received 14 November 2024; revised 5 March 2025 and 9 May 2025; accepted 11 June 2025. Date of publication 18 June 2025; date of current version 2 January 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 62302184, and in part by the Research Grants Council of the Hong Kong Special Administrative Region, China, under Grant PolyU15222621 and Grant PolyU15225023. *(Corresponding authors: Jinyu Chen; Haozhao Wang.)*

Wenchao Xu is with the Division of Integrative Systems and Design, Hong Kong University of Science and Technology, Hong Kong, SAR, China (e-mail: wenchaoxu@ust.hk).

Jinyu Chen, Peirong Zheng, and Yunfeng Fan are with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, SAR, China (e-mail: jinyu.chen@connect.polyu.hk; peirong.zheng@connect.polyu.hk; yunfeng.fan@connect.polyu.hk).

Xiaoquan Yi and Haozhao Wang are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: yixiaoquan@hust.edu.cn; hz_wang@hust.edu.cn).

Tianyi Tian and Wenhui Zhu are with the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: tian_tianyi@bupt.edu.cn; wenhui_zhu@bupt.edu.cn).

Quan Wan is with the School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: muyou@bupt.edu.cn).

Qinliang Su is with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510275, China (e-mail: suqliang@mail.sysu.edu.cn).

Xuemin Shen is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: sshen@uwaterloo.ca).

Digital Object Identifier 10.1109/COMST.2025.3580745

I. Introduction

THE RAPID advancement of artificial intelligence (AI) has positioned foundation models (FMs) as a cornerstone of innovation, driving progress in various fields such as natural language processing, computer vision, and autonomous systems. These models, characterized by their vast parameter spaces and extensive training on broad datasets, incubate numerous applications from automated text generation to advanced multi-modal question answering and autonomous robot services [1]. Some popular FMs, such as GPT, Llama, ViT, and CLIP, are pivotal in pushing the boundaries of AI capabilities, offering sophisticated solutions for processing and analyzing large volumes of data across different formats and modalities. The continuous advancement of FMs significantly enhances AI's ability to comprehend and interact with the world in a manner akin to human cognition.

However, traditional FMs are typically confined to providing question-and-answer services and generating responses based on pre-existing knowledge, often lacking the ability to incorporate the latest information or employ advanced tools. FM-powered agents are designed to enhance the capability of FM. These agents are incorporated with dynamic memory management, long-term task planning, advanced computational tools, and interactions with the external environment [2]. For example, FM-powered agents can call different external APIs to access real-time data, perform complex calculations, and generate updated responses based on the most current information available. This approach improves the reliability and accuracy of the responses and enables more personalized interactions with users.

Developing a serving system with low latency, high reliability, high elasticity, and minimal resource consumption is crucial for delivering high-quality agent services to users. Such a system can efficiently manage varying query loads while maintaining swift response and reducing resource costs. Moreover, constructing a serving system on heterogeneous edge-cloud devices is a promising solution to leverage the idle computational resources at the edge and the abundant computational clusters available in the cloud. The collaborative inference of edge-cloud devices can enhance overall system efficiency by dynamically allocating tasks to various edge-cloud machines based on computational load and real-time network conditions.

Although many research works investigate edge-cloud collaborative inference for small models, deploying FMs under this paradigm for diverse agent services still faces several

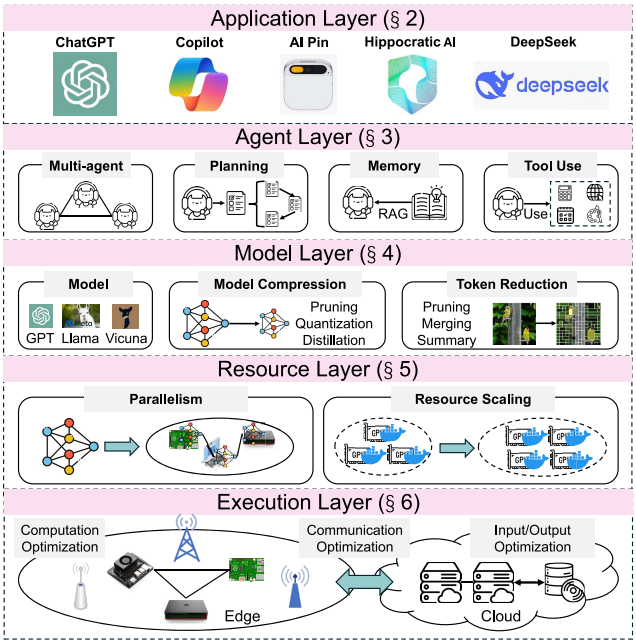

Fig. 1. The framework of FM-powered agent services.

severe challenges. First, the fluctuating query load severely challenges the model serving. The rapidly growing number of users want to experience intelligent agent services with FMs. For example, as of April 2024, ChatGPT has approximately 180.5 million users, with around 100 million of users being active weekly [3]. These users access the service at different times, resulting in varying request rates. An elastic serving system should dynamically scale the system capacity according to the current system characteristics. Secondly, the parameter space of an FM is particularly large, reaching the scale of several hundred billion, which is a significant challenge to the storage system. However, the storage capacity of edge devices and the consumer GPU is limited, making it unable to accommodate an entire model. The large number of parameters results in significant inference overhead and long execution latency. Therefore, it is necessary to design model compression methods and employ different parallelism approaches in diverse execution environments. In addition, users have different service requirements and inputs in different applications. For example, some applications prioritize low latency, while others prioritize high accuracy. This necessitates dynamic resource allocation and adjustment of the inference process. Moreover, AI agents need to deal with lots of hard tasks under complex environments, which require effective management of large-scale memory, real-time processing of updated rules, and specific domain knowledge. Additionally, agents possess distinct personalities and roles, necessitating the design of an efficient multi-agent collaboration framework.

To address the aforementioned challenges and promote the development of the real-time FM-powered agent service, this survey proposes a unified framework and investigates various research works from different optimization aspects. This framework is shown in Fig. 1. The bottom layer is the execution layer, where edge or cloud devices execute an inference with FMs. Joint computation optimization, I/O optimization, and communication optimization are applied to accelerate inference and promote the building of a powerful infrastructure for FMs. The resource layer, comprised of two components, facilitates the deployment of the model on various devices. Parallelism methods design different model splitting and placement strategies to utilize the available resources and improve throughput collaboratively. Resource scaling dynamically adjusts the hardware resources during runtime based on query load and resource utilization, thereby improving overall scalability. The model layer focuses on the optimization of FMs. Two lightweight methods, including model compression and token reduction, are specifically designed to promote the widespread adoption of FMs. Based on these FMs, many AI agents are constructed to accomplish various tasks. Numerous methods have been proposed to enhance the four key components of agents, which encompass the multi-agent framework, planning capabilities, memory storage, and tool utilization. Ultimately, leveraging the aforementioned techniques, all kinds of applications can be developed to deliver intelligent and low-latency agent services to users.

To make it easier to understand and write, we give a table of acronyms in Table I.

TABLE I
ALL ACRONYMS

Acronym	Full Form
AGI	Artificial General Intelligence
AI	Artificial Intelligence
AIGC	Artificial Intelligence Generated Content
API	Application Programming Interface
CoT	Chain-of-Thought
CV	Computer Vision
DL	Deep Learning
DNN	Deep Neural Network
FFN	Feed-Forward Network
FM	Foundation Model
IoT	Internet-of-Things
KD	knowledge distillation
KV	Key-Value
LLM	Large Language Model
MHA	Multi-Head Attention
MLLM	Multimodal LLM
MoE	Mixture-of-Experts
NLP	Natural Language Processing
VM	Virtual Machine or Vision Model
RL	Reinforcement Learning

A. Previous Works

Many research works focus on system optimization to deploy machine learning models in edge-cloud environments. KACHRIS reviews some hardware accelerators for Large Language Models (LLMs) to address the computational challenges [4]. Tang et al. summarize scheduling methods designed for optimizing both network and computing resources [5]. Miao et al. present some acceleration methods to improve the efficiency of LLMs [6]. This survey includes system optimizations, such as memory management and kernel optimization, as well as algorithm optimizations, such as architectural design and compression algorithms, to accelerate model inference.

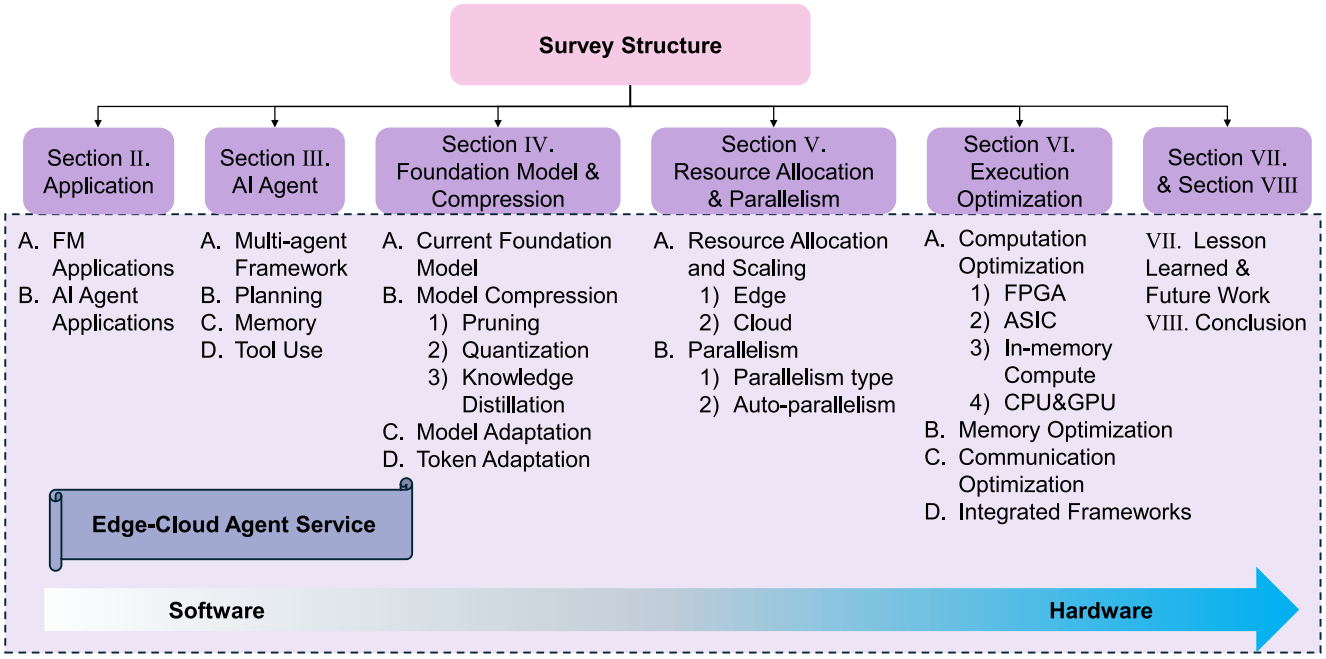

Fig. 2. The framework of our survey. Each technical session corresponds to a layer in Fig. 1.

Xu et al. focus on the deployment of Artificial Intelligence-Generated Content (AIGC), and they provide an overview of mobile network optimization for AIGC, covering the processes of dataset collection, AIGC pre-training, AIGC fine-tuning, and AIGC inference [7]. Djigal et al. investigate the application of machine learning techniques in resource allocation for Multi-access Edge Computing (MEC) systems [8]. The survey includes resource offloading, resource scheduling, and collaborative allocation. Many research works propose different algorithms to optimize the design of FMs and agents. References [1], [9] and [10] present popular FMs, especially LLMs. References [11], [12] and [13] summarizes model compression and inference acceleration methods for LLM. References [2], [14], and [15] review the challenges and progress for the development of agents.

In summary, the above studies either optimize edge-cloud resource allocation and scheduling for small models or design acceleration or efficiency methods for large FMs. To the best of our knowledge, this paper is the first comprehensive survey to review and discuss the deployment of real-time FM-powered agent services in heterogeneous devices, a research direction that has gained significant importance in recent years. We design a unified framework to fill this research gap and review current research works from different perspectives. This framework not only delineates essential techniques for the deployment of FMs but also identifies key components of FM-based agents and corresponding system optimizations specifically tailored for agent services.

B. Contribution

This paper presents a comprehensive survey on the deployment of FM-powered agent services in edge-cloud environments, covering optimization approaches spanning from hardware to software layers. For the convenience of readers, we provide an outline of the survey in Fig. 2. The contributions of this survey are summarized in the following aspects:

- This survey proposes the first comprehensive framework to provide a deep understanding of the deployment of FM-powered agent services within the edge-cloud environment. Such a framework holds the potential to foster the advancement of AGI greatly.
- From a low-level hardware perspective, we present research on various runtime optimization methods and resource allocation and scheduling methods. These techniques are designed to establish a reliable and flexible infrastructure for FMs.
- From a high-level software perspective, we elucidate research efforts focused on model optimization and agent optimization, thereby offering diverse opportunities for building intelligent and lightweight agent applications.

C. Background and Overview

In recent years, FMs, particularly LLMs, have experienced rapid development, laying the foundation for AI agent services. However, LLMs are only capable of processing text. To enable FMs to perceive the world through multiple modalities like humans, multimodal LLMs have emerged, which has broadened the application prospects for AI agents. To address the hallucination problem of LLMs, reasoning models that combine RL with methods such as CoT have become a new solution. For example, reasoning models like OpenAI-o1 [16] and DeepSeek-r1 [17] have demonstrated the strong potential of this approach, further advancing the development of FMs.

LLM agent refers to an AI system that utilizes the capabilities of large-scale language models to autonomously process natural language inputs and generate contextually appropriate responses or actions. These agents are designed to facilitate human-computer interactions, performing tasks such as

information retrieval, decision support, content generation, and process automation. By leveraging the deep learning techniques underlying LLMs, these agents exhibit a high degree of adaptability, enabling them to understand and generate complex language patterns, making them applicable across various domains, including customer service, virtual assistance, data analysis, and more. The fundamental characteristic of LLM agents lies in their ability to generate dynamic, context-aware outputs based on vast linguistic knowledge rather than relying solely on predefined rules or scripts.

The remainder of this article is organized as follows: Section II presents AI agent applications. Section III illustrates key components for agents. Section IV discusses current FMs, as well as techniques for model compression and token reduction. Section V describes resource allocation and parallelism mechanisms. Section VI presents some low-level execution optimization methods. Section VII discusses the lessons learned and future works. Finally, Section VIII draws conclusions.

II. Application Layer

We describe some representative applications of foundation models and AI agents in this section.

A. Foundation Model Applications

Generative AI has demonstrated robust performance and vast potential in commercial applications, playing a significant role across various industries. Commercial generative AI can be categorized into four types: (1) text-generating AI (including some multimodal AI), (2) code-generating AI, (3) image-generating AI, and (4) video-generating AI. Among these, text-generating, code-generating AI, and image-generating AI have already seen widespread application.

- Text-generating and multimodal AI. ChatGPT is the most famous chatbot developed by OpenAI, and performs well in various NLP tasks [18]. ERNIE Bot (Wenxin Yiyan) is similar to ChatGPT, supporting input and output of images and text, and outperforms ChatGPT 4.0 in several Chinese tasks [19].
- Code-generating AI. Github Copilot is the most widely adopted code-generating tool in the world. It can automatically generate code and comments, and provide code explanations, allowing developers to focus on problem-solving and collaboration [20]. Codex, similar to GitHub Copilot, is a product based on GPT-3 and is proficient in lots of programming languages. OpenAI claims that Codex is more powerful than GitHub Copilot [21].
- Photo and Video-generating AI. Midjourney can generate high-quality images according to natural language. It entered public beta on July 12, 2022, and is continuously evolving [22]. Sora, based on diffusion models, GPT and DALL·E, can generate videos from images and language, or expand on already generated videos. OpenAI claims that it is an important step towards AGI [23]. Runway Gen-2 is a high-performance video-generation AI developed by Runway AI, which can take text, image, or video input and convert it into new videos [24].

B. AI Agent Applications

The rise of LLMs has significantly accelerated research related to AI agents, which are now considered a principal avenue toward achieving AGI. Andrej Karpathy, co-founder of OpenAI, has stated that AI agents represent a significant future direction for AI. In the development of digital entities across various industries, these agents are expected to play a pivotal role in applying AGI. AI agents, as products, are anticipated to conduct business operations.

AutoGPT, a project developed by Significant Gravitas, automatically optimizes GPT's hyperparameters by searching for algorithms that enhance its performance across diverse tasks [25]. Users simply set one or more objectives, and AutoGPT can autonomously generate and execute tasks, continually analyzing and refining its strategies throughout the process. Researchers at Stanford and Google have developed a simulated environment named Smallville, where 25 AI agents exhibit complex behaviors and interactions [26]. Each agent possesses a unique seed memory, which enables them to form and recall relationships through social interactions with other agents. VOYAGER, designed with the long-term goal of 'discovering as many different things as possible', is the first AI agent in Minecraft to continuously explore the world, acquiring a diverse set of skills and making new discoveries [27]. It introduces an expanding arsenal of skills for storing and retrieving complex behavioral executables, alongside a novel iterative cueing mechanism that enables the agent to autonomously explore and adapt to unknown environments without human intervention, demonstrating superior proficiency. ChatDev (Chat-powered Software Development) is proposed as a virtual software company operated by multiple intelligent entities [28]. After a user specifies a task, ChatDev utilizes the software engineering waterfall model. Through a sequence known as the Chat Chain, consisting of atomic tasks, it facilitates automated interactions, collaboration, and decision-making among diverse AI agents to produce comprehensive software, including source code, environment dependency specifications, and user manuals.

In addition to research developments, recent advancements in AI agent technology have demonstrated their capacity to autonomously perform complex tasks across diverse applications. For example, OpenAI's Operator leverages the new Computer-Using Agent, which combines GPT-4o's advanced visual processing with the sophisticated reasoning of the o-series models. This integration enables the agent to autonomously operate Web browsers to complete multifaceted tasks, such as creating travel itineraries, retrieving academic papers, and summarizing key findings—all without direct human intervention. OpenAI has also launched Deep Research, a new AI agent built on a customized version of its upcoming o3 model. Designed for Web browsing and data analysis, Deep Research can automatically search, interpret, and synthesize massive amounts of online information. Similarly, Google has introduced a suite of AI agent projects based on its Gemini 2.0 model, including Project Astra, Project Mariner, and Jules. Notably, Jules serves as a programming assistant for software developers by autonomously identifying and

correcting software errors as well as preparing necessary code changes. GLM-PC, developed by ZhiPu, is the world's first public-oriented AI agent. It can operate a computer with human-like proficiency to perform a wide range of complex tasks, and it also boasts advanced logical reasoning and code generation capabilities. Microsoft announced the integration of ten autonomous AI agents into Dynamics 365. These agents can automate a wide range of business processes—including customer service, sales, finance, and warehousing. Built on OpenAI's o1 model and equipped with autonomous learning capabilities, they streamline even the most complex, cross-platform business operations. Manus, conceived and developed by the Chinese research collective Monica, is a versatile AGI agent capable of autonomously formulating plans and executing complex multi-modal tasks.

Research on AI Agents is currently led primarily by the academic community and developers. The outstanding performance of AI Agents in independent operation, collective collaboration, and human-machine interaction is increasingly recognized as a potent tool for enhancing productivity across various industries in the digital economy. However, the widespread application of LLM-based agents faces significant challenges due to the costs associated with token interactions. Additionally, the absence of a profit-sharing mechanism indirectly hampers the development of AI Agents. The main strategies for reducing costs include combining different model sizes and optimizing inference infrastructure. Furthermore, rapid advancements in hardware and models are expected to resolve cost issues in the near future. OpenAI predicts that AI will surpass human intelligence levels within the next decade, achieving what is termed superintelligence. AI Agents are expected to become mainstream in future products, with numerous agent-centered products likely to emerge and be implemented across various fields in the coming years.

C. Lessons Learned

The recent advancements and deployment of foundation model (FM)-based agents across diverse domains have yielded several important lessons:

- Multimodal generative capabilities have matured unevenly: While text-, code-, and image-generating systems have demonstrated strong commercial viability, video-generating models remain in earlier stages of development. Continued research into temporal modeling and diffusion techniques is essential to close this gap.
- Operational cost remains a critical constraint: The high cost associated with token-level inference continues to limit large-scale adoption. Hybrid architectures—combining large and small models—alongside infrastructure-level optimization are emerging as key mitigation strategies.

These findings inform ongoing research into scalable, efficient, and economically sustainable deployment strategies for LLM-based agents, offering valuable guidance for future advancements in both academic and industrial contexts.

Fig. 3. LLM Agent framework. The LLM serves as the central nervous system or cognitive core of the agent and plays a pivotal role in processing information, making decisions, and generating actions.

III. AI Agent

In this section, we introduce the agents that power the above agent applications. The application of intelligent agents, historically linked to benchmarks like the Turing test assessing human-like intelligence, builds upon philosophical concepts of entities possessing autonomy and intent. Within AI, agents are distinguished by their capacity to move beyond mere instruction execution, enabling them to autonomously make decisions, solve problems, and navigate complex environments [29], [30]. In an autonomous agent system powered by LLM, the LLM serves as the agent's central nervous system or cognitive core, playing a pivotal role in processing information, making decisions, and generating actions. The agent is augmented by other essential elements that ensure the agent's robustness, adaptability, and efficiency: Multi-agent framework, Planning, and Tool use.

A. Multi-Agent Framework

An LLM multi-agent framework is a system that leverages multiple LLMs as independent agents working collaboratively to handle complex tasks [31], [32]. Each agent may specialize in different areas or subtasks, and they communicate and cooperate to share information, verify outputs, and enhance overall performance [33].

In multi-agent systems, coordination and communication are the foundations of achieving cooperation. It is necessary to design practical communication protocols to ensure that agents exchange information efficiently and accurately. The collaboration structure can be centralized, with a supervisor responsible for task planning, decomposition, and allocation, or decentralized, allowing agents to coordinate and collaborate autonomously. For instance, AWS Bedrock's hierarchical design features a central hierarchy, expert agent layers, and recursive hierarchical designs, creating an adaptive network for collaboration [34]. Agents must maintain consistent semantic understanding and objectives to avoid misunderstandings and

conflicts. The reasonable allocation and division of tasks are central to multi-agent cooperation. Tasks should be distributed among agents reasonably to avoid overloading some agents while leaving others idle. Knowledge sharing is the basis for achieving collaborative work in multi-agent systems. Different agents may have different backgrounds, and efficiently integrating this knowledge is challenging. LLM-Co explores the potential of LLM for multi-agent coordination, focusing on their ability to collaborate in various scenarios effectively [35]. CLIP [36] explores a novel approach to enhance the ability of LLM to execute tasks in embodied settings, such as with robots, where understanding the physical world is crucial. Semantic In-Context Learning (ICL) offers a promising approach to improve conversational agents by leveraging both semantic search and LLM [37]. Yashar [38] introduces a novel framework to enhance the capabilities of LLM by leveraging multi-agent systems, allowing for collaborative environments where intelligent agents work to handle complex tasks efficiently. The authors demonstrate the framework's superior performance by conducting case studies on models such as Auto-GPT, BabyAGI, and Gorilla, which incorporate external APIs into the LLM.

The deployment of LLM in multi-agent systems faces significant challenges stemming from semantic understanding and consistency across different agents [39]. Each agent may interpret inputs or instructions slightly differently due to variations in their training data or internal parameters. This can lead to inconsistencies in the collective output, where the final result may not accurately reflect the intended goals or actions. Agents must communicate effectively to achieve common goals [32], [40]. DyLAN [41] aims to optimize the collaboration of LLM agents for complex tasks such as reasoning and code generation. DyLAN is introduced to enhance the performance and efficiency of LLM-agent collaboration on complex tasks by enabling dynamic interaction architecture and inference-time agent selection. AgentVerse [42] is a multi-agent framework designed to facilitate collaborative problem-solving among autonomous agents powered by LLMs. AgentVerse emulates human group dynamics to improve task performance and explores emergent behaviors within agent collaborations. Agentbench [43] presents a comprehensive benchmark designed to evaluate the capabilities of LLM when acting as autonomous agents across a variety of real-world challenges. It consists of eight distinct environments categorized into code-grounded, game-grounded, and Web-grounded scenarios to test the LLMs' reasoning, decision-making abilities, and their performance in multi-turn, open-ended generation settings. CoELA [44] presents a novel framework for building cooperative embodied agents using LLM, focusing on multi-agent cooperation without requiring fine-tuning or few-shot prompting. This framework is evaluated in various embodied environments, demonstrating that agents can effectively plan, communicate, and cooperate.

B. Planning

Intelligent agents enhance task-handling capabilities in complex systems by breaking down large tasks into smaller, more specific sub-goals. This decomposition makes task management more feasible, boosting overall efficiency and effectiveness [45], [46]. The planning has the following characteristics:

- Hierarchical approach: Agents solve problems layer by layer by creating a hierarchical structure of tasks. From high-level strategic decisions to low-level specific operations, let LLM think step-by-step.
- Parallel processing: After task decomposition, agents can process multiple sub-goals in parallel, utilizing resources effectively. For example, in multi-agent systems, different agents can simultaneously deal with various tasks.
- Dynamic adjustment: Agents adjust the priority and resource allocation of sub-goals based on real-time feedback to adapt to environmental changes and unforeseen circumstances.

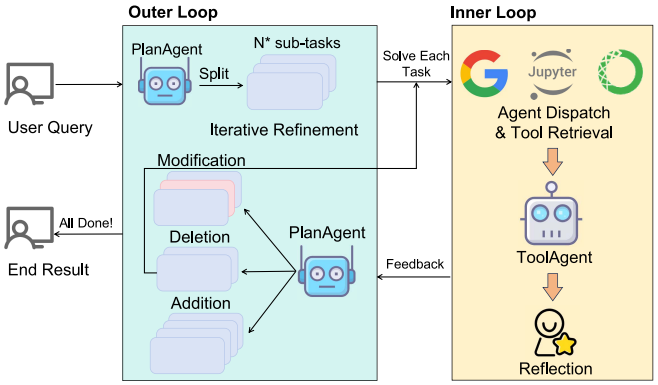

Fig. 4. LLM for planning. Intelligent agents enhance task-handling capabilities in complex systems by breaking down large tasks into smaller, more specific sub-goals.

Intelligent agents can enhance their ability to handle complex tasks through effective sub-goal setting, task decomposition, and continuous reflection and improvement. This strengthens the agents' ability to solve problems independently and optimizes the entire system's performance, contributing to efficient, adaptive, intelligent system design.

MindAgent [47] is designed to evaluate planning and coordination capabilities in gaming interactions. DEPS [48] is an innovative approach leveraging LLM to plan tasks in multi-task agents in open-world environments. DEPS uniquely integrates interactive planning with LLMs, focusing on error correction and efficiency improvement in plan execution through a goal selector module. It significantly improves task success rates in Minecraft and other tasks like ALFWorld and tabletop manipulation. MCTS [49] explores the integration of LLM with Monte Carlo Tree Search for task planning. This combination, termed LLM-MCTS, leverages the commonsense knowledge of LLMs to enhance the efficiency of planning algorithms. The key findings indicate that LLM-MCTS significantly outperforms traditional MCTS and LLM-induced policies in complex task scenarios, demonstrating the advantage of using LLMs as both a world model and a heuristic policy guide.

There are limitations to the direct use of LLMs as planners, such as their limited reasoning and planning capabilities and

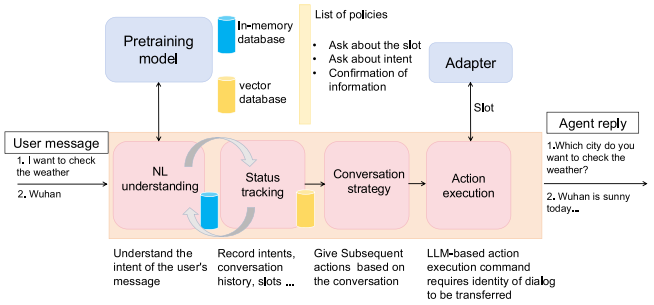

Fig. 5. LLM Agent of Memory. The memory in the historical sequence is dynamically maintained, and it is extracted through a retrieval method.

the inefficiency of the utilization of human feedback. PDDL [50] proposes a new method for planning tasks using pre-trained LLM. The method uses LLMs to construct PDDL models, employs PDDL validation and human feedback to correct initial errors in these models, and generates plans using the corrected PDDL models [50]. ReCon [51] incorporates two processes: formulation contemplation for generating initial thoughts and speech and refinement contemplation for polishing these thoughts. MPC [52] proposes a novel approach for creating high-quality conversational agents without fine-tuning. This approach harnesses the power of pre-trained LLMs as discrete modules, thereby ensuring sustained coherence and adaptability in open-domain conversational contexts. The author demonstrates that MPC performs comparably with fine-tuned chatbot models in open-domain settings, offering an effective solution for generating consistent and engaging chatbots through human evaluations.

In an LLM-powered autonomous agent system, LLM functions as the agent's brain. The agent breaks down large tasks into smaller, manageable subgoals, enabling efficient handling of complex tasks. The agent can engage in autonomous introspection and critique its past actions, extract lessons from its errors, and subsequently refine its strategies for subsequent endeavors, culminating in an enhancement of the ultimate outcome's caliber.

C. Long/Short Term Memory

Running multiple instances of LLMs simultaneously can be resource-intensive, requiring significant computational power and memory. Resource constraints can limit the number of agents that can be deployed concurrently, affecting the overall efficiency and effectiveness of the system. Efficient resource allocation strategies and optimization techniques are needed to maximize performance under limited resources [53]. Memory of the historical chat sequence is dynamically maintained and extracted through a retrieval method. The historical chatbot manages and updates a memory system to ensure relevant information is retained and can be accessed efficiently. This rethinking method helps refine, organize, and integrate this information for future use, enhancing the model's ability to learn from past interactions and apply this knowledge to new situations. This methodology effectively improves the model's decision-making capabilities.

Long-term memory refers to the model's ability to retain information over time, often across multiple interactions or tasks. It is designed to store knowledge for the long haul, allowing the model to remember important details such as a user's preferences, previous conversations, or factual information that is not immediate but could be useful later. Long-term memory typically requires external storage or reinforcement learning techniques to manage and update these memories. Many scholars are currently studying long-term memory in agents. Agents can store and retrieve information over extended periods, allowing them to maintain context and consistency across multiple interactions. REMEMBERER [54] introduces a novel framework for LLM that integrates a long-term experience memory, allowing the model to leverage past interaction experiences for decision-making across different tasks. This approach, termed RL with experience memory, enables the LLM to evolve its capabilities without fine-tuning parameters, positioning it as a semi-parametric reinforcement learning agent. The methodology updates the experience memory through RL processes, and experiments on two RL task sets demonstrate REMEMBERER's superiority over state-of-the-art methods.

Short-term memory can process the current context rapidly because it only needs to focus on a limited amount of recent input. In contrast, long-term memory may involve more time-consuming operations, like querying external databases or searching for stored knowledge, which can slow down the process. MoT [55] aimed at enhancing the capabilities of LLM like ChatGPT without the need for additional annotated datasets or computationally expensive fine-tuning. This approach draws inspiration from the human ability to self-improve through self-reflection and short-term memory. It proposes a mechanism for LLMs to self-improve by leveraging their internal generation processes and an external memory component. Liu et al. [56] introduces reasoning and acting through scratchpad and examples (RAISE), a sophisticated architecture designed to improve the integration of LLM like GPT-4 into conversational agents. The architecture uses a short-term memory system to maintain the continuity of contextual conversations better. It outlines a comprehensive method for constructing agents, including historical conversation selection, scene extraction, chain of thought completion, and scene augmentation, leading to the LLMs training phase.

Traditional LLMs remember patterns, facts, and information across the data they were trained on, allowing them to generate knowledgeable responses. With the support of the long/short-term memory system, LLMs can quickly retrieve information from memory, making them highly effective for tasks that require rapid access to a broad range of information, such as question-answering systems or recommendation engines.

D. Tool Use

In the context of multi-agent systems, particularly those involving LLM, the capability for agents to call external application programming interfaces (APIs) significantly enhances their functionality and adaptability [57], [58]. This approach

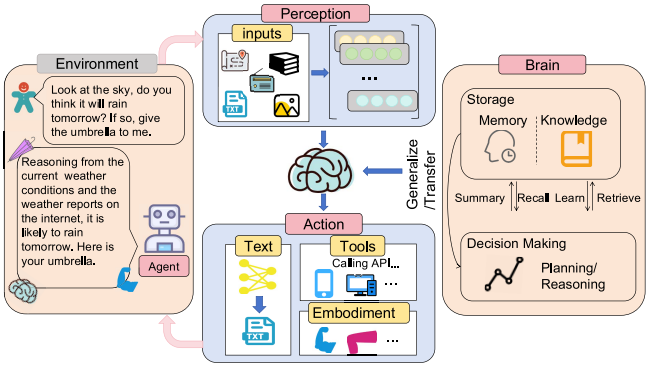

Fig. 6. LLM agent of using tools. External APIs and tools can extend the abilities of an LLM beyond its original training.

addresses several limitations of pre-trained models, offering the following benefits:

- Access to current information: LLMs, once trained, do not inherently update their knowledge unless retrained with new data. By enabling agents to call external APIs, they can access the most current information available, which is crucial for tasks that depend on up-to-date data such as news updates, stock prices, or weather forecasts.
- Enhanced Functional Capabilities: External APIs can extend the abilities of an LLM beyond its original training. For instance, agents can perform calculations, access mapping services, or retrieve specialized data that is not stored within their pre-trained weights.
- Interactivity with Proprietary Systems: Agents can interact with proprietary information sources that are not publicly available or part of their training data. This is especially valuable in enterprise environments where access to internal databases or industry-specific systems is necessary.
- Dynamic Adaptation to New Domains: The ability to call external APIs allows agents to dynamically adapt to new domains or changes in their operating environment without the need for retraining. This makes the system more flexible and responsive to user needs.

Many researchers design different automation and standardization methods for integrating diverse tools into the execution of LLMs, enabling LLMs to invoke various APIs and external resources seamlessly. They are dedicated to developing context-aware tool selection mechanisms that allow LLMs to intelligently choose the most suitable tools based on the dialogue content and achieve multi-step decision-making in complex tasks. Toolformer [59] is specifically designed to make informed decisions regarding the selection and invocation of appropriate Application Programming Interfaces (APIs). It determines not only the optimal timing for API calls but also the most suitable arguments to be passed to these interfaces. They integrate a diverse array of utilities, encompassing tools such as a computational calculator, a question-and-answer system, a robust search engine, a multilingual translation service, and a scheduling calendar. Remarkably, Toolformer demonstrates significantly enhanced performance in zero-shot scenarios across a wide array of downstream tasks, frequently matching or even outperforming much larger models. Gpt4tools [60] introduces an innovative methodology aimed at empowering LLMs, such as LLaMA and OPT, with the capacity to leverage tools. This is achieved by crafting an instruction adherence dataset through the strategic prompting of a sophisticated teacher model across diverse multimodal environments. Low-Rank Adaptation optimization techniques are then employed during the fine-tuning process, enabling these LLMs to undertake visual understanding tasks and generate images. The resultant system showcases marked enhancements in accuracy for both familiar and novel tool applications. Ruan [61] focuses on task planning and tool usage. They introduce two types of agents, one-step and sequential, to execute tasks through planning and using external tools. Agent-to-agent (A2A) communication enables direct and real-time information exchange, which is crucial for effective collaboration when utilizing external tools [33]. Additionally, multi-agent coordination protocols (MCP) establish a high-level strategic framework and governance structure that orchestrates tool usage across the system [62], thereby promoting coherent, efficient, and scalable inter-agent collaboration.

LLMs have gained significant prominence in recent years, and optimizing the efficiency of their API calls in different environments and contexts is crucial for enhancing their performance and practical applications. Liu et al. [63] presents CliqueParcel to improve the efficiency of LLMs via prompt batching. Existing strategies often compromise output quality, leading to a discounted output problem. CliqueParcel redefines efficiency measurements by excluding the reduction in running time due to shorter lengths. It provides a comprehensive trade-off between efficiency and faithfulness. Kinjal introduces NESTFUL, a benchmark to evaluate LLMs on nested sequences of API calls. Autonomous agent applications powered by LLMs rely on planning and executing the use of tools and external APIs in sequence [64]. Zhang [65] proposes a simple yet controllable target-driven approach. Given that most open-source LLMs have limited tool-use or tool-plan capabilities, Reverse chain employs LLMs to implement simple tasks like API selection and argument completion. A generic rule is used for controllable multiple function calling. After selecting a final API to handle a task via LLMs, required arguments are filled from user query and context. Missing arguments can be completed by letting LLMs select another API based on the API description. Extensive numerical experiments show the impressive capability of Reverse Chain in implementing multiple function calling and improving the tool-use capabilities of existing LLMs like ChatGPT. Guillem [66] focuses on neural caching to curtail the frequency of costly API calls to LLMs. A smaller language model, a student, is continuously trained on the responses of the LLM. A policy decides which requests should be processed by the student alone and which should be redirected to the LLM, aiding the student's learning. This approach can optimize API calls in different classification tasks and environments.

Using APIs, agents can leverage external computing resources, which can be more efficient and scalable than

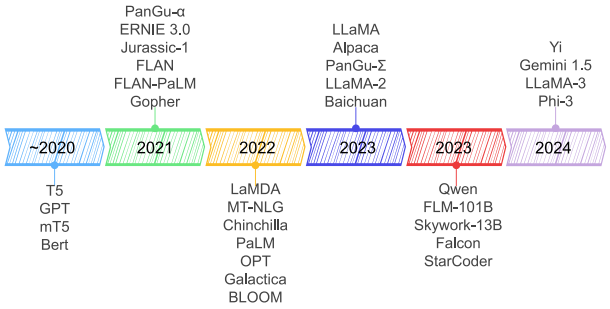

Fig. 7. The timeline of some popular LLMs. Each year highlights major releases that reflect advancements in model architecture, training scale, and capability, showcasing the rapid development of the foundation model ecosystem across different institutions and regions.

processing everything locally. This is especially valuable for resource-intensive tasks. This approach transforms the agent from a static information processor to a dynamic information seeker, greatly enhancing its utility and ensuring its relevance regardless of when it was last trained. This capability is central to developing intelligent systems that are expected to operate in real-world, ever-changing environments.

E. Lesson Learned

Analysis of the architecture, capabilities, and challenges of AI agents, particularly LLM-powered systems, yields the following crucial lessons learned:

- Synergistic architecture: Effective agents require LLM cores integrated with planning, memory, and tool-use modules for real-world efficacy.
- Multi-Agent coordination: Achieving success in complex environments necessitates robust communication protocols that ensure semantic coherence and enable efficient task allocation among specialized agents.
- Sophisticated planning: Addressing complex tasks demands sophisticated planning strategies involving hierarchical task decomposition, dynamic adaptability, and reflective learning mechanisms.

IV. Foundation Model and Compression

In this section, we introduce FMs and their compression and adaptation methods that power the above agents.

A. Current Foundation Model

Since the release of ChatGPT, FMs, especially LLMs, have become increasingly important in daily life. 7 presents a timeline of some popular LLMs. Table II highlights the structural features of several basic LLMs.

1) Large Language Models: Currently, LLMs serve as the core components of agent services, enabling them to process and generate human-like responses, while also providing the flexibility to adapt to different user needs and tasks.

T5 is based on the encoder-decoder transformer architecture [67]. The authors found that transfer learning can significantly enhance performance, especially when combined with a lot of high-quality data [68], so they design pre-training tasks as text-to-text tasks to pre-train a model. *GPT-3* is published by OpenAI, with a major enhancement in in-context learning capabilities compared with GPT-2 [69] through model scaling [70]. To improve the multilingual understanding, Zeng et al. train *PanGu-*α on a 1.1TB high-quality Chinese text corpus, which exhibits decent performance in various Chinese NLP tasks under few-shot or zero-shot conditions [71]. *ERNIE 3.0* integrates autoregressive and autoencoding networks, which allows easy customization for natural language understanding and generation tasks, solving the disadvantage of downstream language understanding in previous LLMs trained on plain text [72]. At a later stage, *Gopher* is introduced and achieves state-of-the-art performance in most of the 152 tasks [73].

To enhance LLMs' safety and response quality, LaMDA is fine-tuned on labeled data and can reference external knowledge sources. LaMDA generates multiple response candidates in dialogues, filters out those with lower safety scores, and outputs the one with the highest quality score [74]. *MT-NLG*, a 540B model with strong zero, one, and few-shot capabilities, is trained with an efficient and scalable 3D parallel system [75]. Previous studies focus on increasing the model parameters without enlarging the size of pretraining tokens. Hence, DeepMind studied how to balance the number of parameters and tokens within a given computational budget and found that the number of parameters and tokens should scale equally. Based on this, *Chinchilla* is trained with 1.4T tokens and demonstrates superior performance on numerous downstream tasks compared to other LLMs such as Gopher (280B) [76] and GPT-3 (175B) [70], [73]. *PaLM* uses a decoder-only transformer architecture instead of an encoder-decoder architecture [69] to enhance few-shot capability. *OPT*, comparable to GPT-3, is developed to address the problem that most LLMs are not open-sourced and have limited access to other developers [77]. To help researchers find useful information, Galactica is trained on a vast amount of scientific corpora, reference materials, and other academic databases and outperforms other LLMs on various scientific tasks [78]. To promote the transparency of LLM research, *BLOOM* is published and open-sourced. It is trained on a dataset with hundreds of sources in 46 natural languages and 13 programming languages. Benchmarks suggest that fine-tuning BLOOM with multitask prompts can improve its performance [79].

Based on Chinchilla's contribution, which indicates the number of tokens and parameters should scale equally [73], *LLaMA* focuses on using more training tokens to achieve optimal performance. Although Hoffman et al. suggest training a 10B model on 200B tokens, LLaMA-7B, a small model with a bytesize of 14GB (float16), is trained on 4T tokens, indicating that increasing the number of tokens can still enhance the model's performance [80]. Building on this observation, MetaAI publishes and open-sources *LLaMA 2*. It utilizes a larger corpus, longer context lengths, and grouped-query attention to train the model [81]. Subsequently, MetaAI employed higher-quality training data and larger-scale pretraining to release the Llama 3 series. Its 405B version is capable of rivaling GPT-4 [82]. Due to its open-source nature and outperforming performance, the Llama series is widely

TABLE II
BASIC LANGUAGE MODELS

Model	Time(Y.M)	n_{para}	n_{layer}	n_{head}	d_{model}	AF	Attention Type	PE
T5	19.10	60M∼11B	6∼24	8∼128	512∼1024	ReLU	Multi-head	Relative
GPT-3	20.05	125M∼175B	12∼96	12∼96	768∼12288	GELU	Multi-head	Sinusoidal
PanGu-α	21.04	2.6B∼207.0B	32∼64	40∼128	2560∼16384	GELU	Multi-head	Learned
ERNIE 3.0	21.07	10B	48, 12	64, 12	4096, 768	GELU	Multi-head	Relative
Jurassic-1	21.08	7.5B, 178B	32, 76	32, 96	4096, 13824	GELU	Multi-head	Sinusoidal
Gopher	21.12	44M∼280B	8∼80	16∼128	512∼16384	GELU	Multi-head	Relative
LaMDA	22.01	2B∼137B	10∼64	40∼128	2560∼8192	gated-GELU	Multi-head	Relative
MT-NLG	22.01	530B	205	128	20480	GELU	Multi-head	\
Chinchilla	22.04	44M∼16.183B	8∼47	8∼40	512∼5120	\	Multi-head	\
PaLM	22.04	8.63B∼540.35B	32∼118	16∼48	4096∼18432	SwiGLU	Multi-query	RoPE
OPT*	22.05	125M∼175B	12∼96	12∼96	768∼12288	ReLU	Multi-head	RoPE
Galactica*	22.11	125M∼120.0B	12∼96	12∼80	768∼10240	GELU	Multi-head	Learned
BLOOM*	22.11	559M∼176.274B	24∼70	16∼112	1024∼14336	GELU	Multi-head	Alibi
LLaMA*	23.02	6.7B∼65.2B	32∼80	32∼64	4096∼8192	SwiGLU	Multi-head	RoPE
LLaMA 2*	23.07	7B∼70B	32∼80	32∼64	4096∼8192	SwiGLU	Grouped-query	RoPE
Baichuan*	23.09	7B, 13B	32, 40	32, 40	4096, 5120	SwiGLU	Multi-head	RoPE, AliBi
Qwen*	23.09	1.8B∼14B	24∼40	16∼40	2048∼5120	SwiGLU	Multi-head	RoPE
Skywork-13B*	23.10	13B	52	36	4608	SwiGLU	Multi-query	RoPE
Falcon*	23.11	7B∼170B	32∼80	64	4544∼14848	GELU	Multi-group	RoPE
StarCoder*	23.12	15.5B	40	48	2048	\	Multi-query	Learned
Yi*	24.03	6B, 34B	32, 60	32, 56	4096, 7168	SwiGLU	Grouped-query	RoPE
Llama 3*	24.04	8B, 70B	\	\	\	SwiGLU	Grouped-query	RoPE

* indicates open-source. PE: Positional Embedding. AF: Activation Function

deployed in LLM-based agent services. In float16, the 1B version of Llama 3.2 has a size of ∼2.5GB and occupies ∼2.3GB of GPU memory during inference, which can be deployed on edge devices like mobile phones to provide relatively simple agent services. Conversely, the 405B version has a size of 811.9GB and occupies ∼810GB of GPU memory during inference, requiring deployment across multiple H100 nodes to meet more complex demands. Following the LLaMA series, *Baichuan* is released to enhance the performance of Chinese NLP tasks. To enhance the compression rate for Chinese, Byte-Pair Encoding is adopted as the tokenization algorithm, and the tokenization model is trained on 20M multilingual corpora. It separates all numbers into individual digits to enhance the model's mathematical capabilities and integrates various optimizations for Chinese support, including operator, tensor partitioning, mixed-precision, training recovery, and communication technologies [83], [84]. Bai et al. publish the *Qwen* series of open-sourced language models, which includes QWEN, Qwen-Chat, CODE-QWEN, CODE-QWEN-CHAT, and MATH-QWEN-CHAT. QWEN series shows strong performance compared to other open-source models, while a little inferior compared to the proprietary models [85]. The ensuing Qwen 2 and Qwen 2.5 use more and higher-quality pretraining datasets, adopt techniques like dual chunk attention [86] to increase the context window, and offer a more diverse range of model sizes. It is worth mentioning that the 405B version uses the MoE architecture. Due to the advantages of the Qwen series, it is one of the most often used model series for agent services. The smallest Qwen model (0.5B, ∼1GB in float16) can be deployed on smart glasses for agent services. Other Qwen models (1.5B ∼ 72B, bytesize ranges 3GB∼145.5GB in float16) are suitable for deployment on PCs with consumer-grade GPUs to provide more personalized services while ensuring privacy [87], [88].

Wei et al. develop *Skywork-13B*, which is trained on a corpus of over 3.2T tokens extracted from English and Chinese texts [89]. *Falcon* series is trained on high-quality corpora primarily assembled from Web data. Almazrouei et al. release a custom distributed training codebase that allows efficient pretraining of these models on up to 4,096 A100s on cloud AWS infrastructure [90]. Given the widespread use of code-generating LLMs, *StarCoder*, trained in over 80 programming languages with multi-query attention, is released and open-sourced for public use [91].

In addition to training a model from scratch, using existing pre-trained models and fine-tuning them on a formatted language dataset is also a commonly employed method called instruction-finetuning [92]. It can improve the model's understanding and response to inputs with specific instructions by training them on instructional task datasets, which is significant for deploying agent service due to instruction-finetuned models can perform better in domain-specific tasks. An instance of the dataset usually consists of instruction, input, and output. For example, the instruction is "What is the answer to the formula?", the input is "7+3", and the output is "10".

Xue et al. employed this method by fine-tuning T5 on a dataset comprising 101 languages and released *mT5*, significantly enhancing its multilingual capabilities. References [68], [93]. FLAN is based on LaMDA 137B [74] and instruction-tuned on over 60 datasets with natural language instruction templates, significantly improving its performance. Ablation studies reveal that the number of fine-tuning datasets, model scales, and the utilization of natural language instructions are crucial to the success of instruction tuning [94]. The process of fine-tuning *Flan-T5, Flan-PaLM, Flan-cont-PaLM*, and *Flan-U-PaLM* is expanded to include datasets from Muffin, T0-SF, NoV2, and CoT. Notably, adding nine CoT datasets greatly enhances the models' reasoning

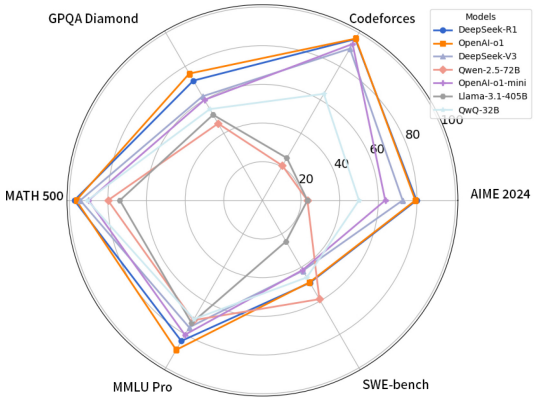

Fig. 8. The radar chart presents the performance of various models across multiple benchmarks categorized into general tasks, mathematics and science tasks, and coding tasks.

capabilities [95]. To promote academic research of LLMs, Taori et al. train *Alpaca* based on LLaMA-7B, using 52k instruction-following demonstrations generated by OpenAI's text-davinci-003. Alpaca's performance is very similar to that of text-davinci-003, yet surprisingly, it is smaller in scale and also inexpensive to train($< 600\$$) [96].

Although the models have achieved notable success in fields such as dialogue systems, their issues with hallucinations cannot be overlooked. Currently, using large-scale reinforcement learning (RL) [97] to enable models to perform reasoning through specific mechanisms, such as chain-of-thought reasoning [98], has emerged as a new research direction.

OpenAI-o1 and DeepSeek-r1 utilize such method, and achieve the state-of-the-art performance in closed-source and open-source domain respectively. In detail, OpenAI-o1 is trained with RL to perform complex reasoning, which allows it to produce a long chain of thought before outputting the answer to the user [16]. Similarly, the DeepSeek-AI team chooses DeepSeek-V3-Base, a 671B MoE model as the base model [99], employs GRPO [100] as the RL framework to enhance the model's reasoning capability, and incorporates a small amount of cold-start data and a multi-stage training pipeline to avoid problems such as poor readability and language mixing [17]. These reasoning-focused models are significant for agent services that often need to solve complex and multi-step tasks such as customer support and homework help agent services.

We have summarized the performance of several commonly used models in relation to General Tasks (MMLU Pro [101]), Mathematics and Science Tasks (GPQA Diamond [102], AIME 2024, MATH-500), and Coding Tasks (Codeforces, SWE-bench [103]) in 8. Notably, OpenAI-o1, OpenAI-o1-mini, and DeepSeek-r1 stand out for their exceptional performance, achieving excellent results across various domains, which highlights their versatility and applicability in different types of agents. Among the smaller-scale models, QwQ-32B [104] also shows remarkable performance, emphasizing its potential for agent applications in resource-constrained environments.

2) Multimodal Models: LLMs are designed to process and generate text-based information, whereas humans interact with the world through multiple sensors, such as visual and auditory modalities. To bridge this gap, multimodal LLMs (MLLMs) are designed to handle text, images, videos, audio, points, boxes, inertial measurement unit, functional magnetic resonance imaging, etc, enabling them to process diverse tasks. The emergence of MLLMs has empowered agent service with new capabilities. By integrating multimodal inputs, agent services are able to perceive and comprehend complex real-world scenarios in a more comprehensive manner, thereby delivering more precise and intelligent services [105]. MLLMs are the core of agent services like embodied agents, multimodal QA, and interactive decision-making. Currently, the training of MLLMs typically involves three stages: pre-training, fine-tuning, and prompting. Training an MLLM from scratch is very costly. Most prior works on MLLMs have focused on aligning existing VMs (Vision Models) with pre-trained LLMs and fine-tuning the alignment module to improve performance.

Many research works design different methods to align a VM to an LLM. *Flamingo* designs new benchmarks for few-shot visual and language tasks. It uses the Perceiver Resampler and Gated Xattn-Dense to align an LLM and a VM, allowing it to process and integrate visual and textual data sequences. It demonstrates strong performance with few-shot capabilities in visual question answering and close-ended tasks [106]. Different from Flamingo, miniGPT-4, mPLUG-owl, and PandaGPT use a linear layer to align. *miniGPT-4* demonstrates that correctly aligning visual features with an LLM can unlock the advanced multimodal capabilities of GPT-4 [107]. Training solely on a large-scale image-text paired dataset might lead to unnatural language outputs like repetition and fragmentation. To overcome this, MiniGPT-4 is fine-tuned on a small but higher-quality dataset with more detailed textual descriptions [108]. Previous works align LLM and VM by training them on joint image-text tasks, which could compromise LLM's language capabilities. *CogVLM* innovatively incorporates trainable visual expert modules between the attention and feed-forward neural network layers to align LLM and VM. This approach allows for deep integration of visual and textual elements without compromising language capabilities because all parameters of the LLM are fixed [109].

Lots of research works focus on enhancing the MLLMs to process different input and output modalities because previous models can only deal with a small set of modalities. PandaGPT utilizes ImageBind as an encoder that can embed data from different modalities into the same feature space [110]. Therefore, PandaGPT has strong cross-modal zero-shot capabilities, allowing it to naturally integrate multimodal inputs and perform complex multimodal tasks efficiently [111]. *NExT-GPT* is an end-to-end, general-purpose any-to-any MLLM. It connects the Vicuna with multimodal adapters and various diffusion decoders, enabling it to perceive different inputs and generate outputs in arbitrary combinations of text, images, videos, and audio. It leverages pre-trained encoders and decoders and requires only a few parameters to be tuned, thus reducing training costs and facilitating expansion to more modalities [112]. *OneLLM* aligns eight modalities through a

universal encoder and projection modules, along with a step-by-step multimodal alignment process, pioneering a universal MLLM architecture. Initially, it connects the LLM and visual encoder via an image projection module. It then expands to additional modalities using a universal projection module and dynamic routing, demonstrating its scalability and generality [113]. *Gemini* is an MLLM pre-trained on a large multimodal dataset, capable of handling a variety of text inputs interleaved with audio, video, and images. Gemini 1.5 utilizes an MoE architecture, enabling it to process multimodal inputs up to 10M tokens in length, thus exhibiting exceptional performance across modalities [114], [115].

A few research works enhance the MLLM to understand fine-grained visual information in an image. Based on QWEN [85], *Qwen-VL* uses a high-quality dataset with fine-grained vision-language annotation and a larger image input resolution, achieving remarkable performance on fine-grained visual understanding capabilities. It outperforms other models in various vision-centered benchmarks such as image description, question answering, and visual localization [85]. Building on this foundation, *Qwen2.5-VL* introduces dynamic resolution processing and absolute time encoding, which enables it to perform exceptionally well in the processing of graphics and long videos. By employing Vision Transformer and window attention mechanisms, it significantly reduces computational overhead while maintaining native resolution [116]. *Shikra* is the first MLLM capable of detecting specific areas in images. It takes natural language as input and outputs the coordinates. This feature supports visual question answering, image description, and more specialized spatial tasks without additional complex setups or external modules [117]. To address spatial understanding issues, especially in referring and grounding tasks, *Ferret* designs a hybrid regional representation that integrates discrete coordinates and continuous features to represent an area in an image jointly. Considering that grid-based processing (e.g., convolution, patch attention) has difficulty in handling regions with an irregular shape, a spatial-aware visual sampler is adopted, allowing it to handle various regional inputs [118]. *NExT-Chat* adopts the innovative pix2emb approach instead of pix2seq [119], enabling it to process and output different positional formats, which allows for more flexible and precise object localization and representation, such as bounding boxes and segmentation masks [120]. Llama-4 utilizes an MoE architecture, supporting long-text processing and multimodal understanding. It includes Llama-4 Scout with 17 billion active parameters, Llama-4 Maverick which also has 17 billion active parameters but with 128 experts, and the under-training Llama-4 Behemoth with 288 billion active parameters. These models have demonstrated excellent performance in various benchmarks [121].

B. Model Compression

1) Pruning: LLMs demonstrate remarkable capabilities in various tasks, including text generation, translation, and sentiment analysis. However, the deployment of them in real-world applications is often hindered by their huge size and computational requirements. Pruning, a technique aimed

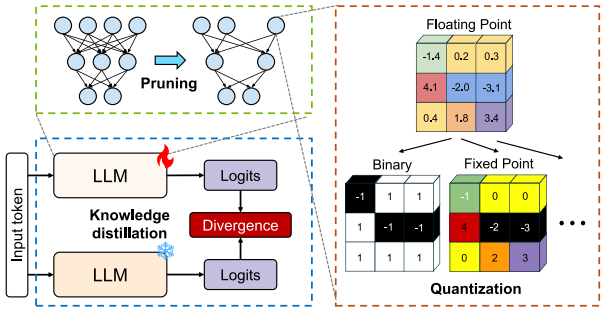

Fig. 9. The illustration of model compression methods. (1) Pruning, which removes redundant parameters from the model structure; (2) Quantization, which reduces numerical precision to binary or fixed-point formats; and (3) Knowledge Distillation, where a smaller student model learns from the softened outputs of a larger teacher model.

at lowering the computational cost and maintaining the performance of LLMs, has attracted significant attention from researchers. Pruning selectively removes weights or neurons from a pre-trained model to reduce its size and computational requirements, as demonstrated in Fig. 9. However, pruning LLMs poses unique challenges due to their complex architectures and the need to preserve specific language patterns during compression. Firstly, the size of LLMs is particularly large, which often consists of countless parameters, leading to significant computational expenses during both training and inference stages. Secondly, existing pruning methods often necessitate retraining or fine-tuning, resulting in additional computational overhead. Moreover, pruning techniques should retain performance across various NLP tasks, such as text modeling, text classification, and machine translation. Achieving explainability in pruned models is also essential for understanding their decision-making processes.

Recent advancements in pruning techniques have addressed several challenges associated with LLMs. They typically design their pruning methods by simplifying dependency on pruning techniques or enhancing compatibility. For instance, they aim to eliminate the need for retraining or weight updates, avoid task-specific compression, and reduce dependence on the original training corpus. Alternatively, they combine the strengths of various pruning methods or focus on making the pruning techniques compatible with hardware platforms. To improve the effectiveness and flexibility of these methods, new criteria are also introduced for pruning, such as sensitivity-based sparsity allocation.

Wanda [122] aims to induce sparsity for LLMs without requiring retraining or extensive computational resources. Wanda utilizes a unique pruning metric that considers both the weights and the relevant norm input activation. LLM-Pruner [123] removes non-critical coupled structures according to gradient data. LLM-Pruner reduces reliance on the original training data and is an automatic structural pruning framework. Loraprune [124] considers neural network pruning with LoRA, introducing a criterion for weight importance estimation and integrating parameter-efficient fine-tuning, demonstrating superior compression rates and reduced memory usage over existing methods on LLaMA models. An et al. [126] propose FLAP, which introduces structured importance metrics

TABLE III
PRUNING METHODS

Ref.	Challenge	Method
Wanda [122]	Pruning LLMs requires costly retraining.	Prune weights with the smallest magnitudes multiplied by the corresponding input activations.
Llm-pruner [123]	The training corpus of LLMs is enormous.	Remove non-critical coupled structures based on gradient information.
LoRAPRUNE [124]	Unstructured pruning cannot work with LoRA weights.	Use the weights and gradients of LoRA for importance estimation.
LoRAShear [125]	The enormous size of LLM leads to computational costs.	A dynamic fine-tuning scheme with dynamic data adaptors.
An et al. [126]	Existing retraining-free pruning approaches require hardware support for acceleration.	Fluctuation-based adaptive structured pruning.
Shao et al. [127]	Existing pruning methods require extensive retraining of pruned models.	Allocate sparsity adaptively based on sensitivity.
Compresso [128]	One-shot structured pruning leads to performance decline.	Learn optimal pruning decisions during the training process.
SHEARED LLAMA [129]	Training smaller yet powerful LLMs from scratch is costly.	Targeted structured pruning and dynamic batch loading.
Ji et al. [130]	The manual design of pruning features leads to a complex optimization pipeline.	Train a non-neural model as an accuracy predictor.
GBLM-Pruner [131]	Prior approaches overlooked informative gradients derived from pretrained LLMs.	Harness normalized gradients from calibration samples to determine pruning metric.
ZipLM [132]	Generalize to different pruning settings.	Identify and remove components with the worst loss-runtime trade-off.

and compensation mechanisms to mitigate performance loss. Shao et al. [127] introduces a pruning method based on mixed Hessian-sensitive sparsity to achieve at least 50% sparsity in LLMs without retraining, which reduces pruning errors and maintains overall sparsity levels by adaptively allocating sparsity based on sensitivity.

Sheared llama [129] prunes LLaMA2 from 7B to 1.3B and 2.7B parameters, outperforming equivalent-sized models while needing only 3% of the compute for training. It effectively addresses challenges in optimizing pruned architectures and continuing pre-training, showing superior performance. Ji et al. [130] employ gradient boosting decision trees (GBDT) as an accuracy predictor to guide pruning process based on specific performance requirements. This predictor is then utilized to optimize the search space and select the most suitable pruned model. Das et al. [131] introduce GBLM-Pruner, a sparsity-centric pruning method for billion-parameter LLMs, operating without retraining and surpassing competitors like SparseGPT and Wanda across benchmarks. It unveils structural patterns in unstructured pruning within LLMs, maintaining simplicity and efficiency. Structured pruning algorithm in ZipLM [132] considers both local and global correlations, ensuring precise pruning. Additionally, it is enhanced with a layer-wise token-level distillation technique. FitPrune [133] automatically generates a comprehensive pruning recipe for MLLMs based on a predefined computational budget. The method significantly reduces computational complexity while preserving model accuracy. For instance, in LLaVA-NEXT, FitPrune achieves a 54.9% reduction in FLOPs with only a marginal 0.5% accuracy degradation.

2) Quantization: LLMs' widespread deployment presents severe challenges, such as the massive memory requirements and computational overhead. To alleviate these issues, researchers have developed different quantization techniques tailored specifically for LLMs. Quantization reduces the bit-width of models, thereby diminishing memory footprint and enhancing inference speed. Key concepts in quantization include post-training quantization (PTQ), quantization-aware training (QAT), and methodologies for managing weight and activation quantization.

In recent years, the research community has made significant efforts to address challenges in quantizing LLMs, such as managing the substantial accumulation of quantization errors due to their vast number of parameters and layers, mitigating sensitivity to outliers in weights and activations, effectively quantizing activations with wide dynamic ranges, navigating constraints associated with post-training quantization, and ensuring compatibility across diverse hardware platforms. The quantization methods can be divided into three categories: 8-bit weight and 8-bit activation (W8A8) quantization, low-bit weight-only quantization and quantization-aware training. The W8A8 method employs a quantization process that reduces both weights and activation to 8-bit formats. Smoothquant [134] achieves a balance between accuracy and hardware efficiency by considering activation outliers and simplifying quantization complexities. It delivers significant performance enhancement with up to 1.56× speedup and 2× memory reduction without sacrificing accuracy. OSTQuant [146] uses orthogonal and scaling transformations to optimize the distribution of weights and activations across the entire quantization space. OSTQuant introduces QSUR, a metric to evaluate quantization efficiency, and a new loss function (KL-Top) to enhance semantic retention during calibration, achieving superior accuracy in low-bit settings and significantly outperforming prior methods, especially on LLaMA-3-8B. RPTQ [135] saves up to 80% memory with high accuracy levels when quantizing OPT-175b. LoftQ [136] quantizes LLMs while identifying an appropriate low-rank initialization for LoRA fine-tuning, thereby enhancing generalization in downstream tasks. Outlier suppression+ [137] presents efficient techniques for determining optimal shifting and scaling values. Its performance is comparable to floating-point precision and establishes new benchmark criteria for 4-bit BERT models. FPTQ [138] introduces layerwise activation quantization strategies, including a novel logarithmic equalization technique, to enhance performance. FPTQ achieves outstanding W4A8 quantized performance without the need for additional fine-tuning, thereby simplifying the production of LLMs. QLLM [145] is an efficient low-bitwidth quantization method with a channel reassembly technique

TABLE IV
QUANTIZATION METHODS

Type	Ref.	Challenge	Method
W8A8	Smoothquant [134]	The presence of outliers in activation.	Outlier smoothing and per-channel scaling transformation.
	RPTQ [135]	Varying ranges across channels.	Rearrange channels and quantize them in clusters.
	LoftQ [136]	Performance differences between 2 fine-tuning method.	Find a proper low-rank initialization for LoRA fine-tuning.
	Outlier suppression+ [137]	Existence of detrimental outliers in activations.	Channel-wise shifting for asymmetry and channel-wise scaling for concentration.
	FPTQ [138]	The W4A8 faces notorious performance degradation.	Layerwise activation quantization strategies.
Low-bit weight-only	OWQ [139]	The presence of activation outliers.	Prioritize structured weights sensitive to quantization in high-precision.
	AWQ [140]	The significant model sizes of modern LLMs.	Search for optimal per-channel scaling by observing the activation.
	Zhang et al. [141]	The superiority of low-bit Integer versus Floating Point formats is unclear.	Select the optimal format on a layer-wise basis.
	Omniquant [142]	Hand-craft quantization parameters lead to low performance.	Learnable Weight Clipping (LWC) and Learnable Equivalent Transformation (LET).
	IntactKV [143]	Previous quantization methods compromise LLM performance.	Construct KV cache of pivot tokens from full-precision model.
	Kim [144]	Previous quantization methods are designed for inference.	Update solely the quantization scales during fine-tuning.
	QLLM [145]	Activation outliers in particular channels.	Reallocate the magnitude of outliers to other channels.

W8A8: 8-bit weights and activation.

for handling activation outliers. It also includes an adaptive strategy to determine the optimal number of disassembled channels and an efficient error correction mechanism with low-rank parameters.

The low-bit weight-only quantization methods quantize LLM weights into low-bit integers, usually 4 bits or fewer. OWQ [139] prioritizes critical weights for high-precision storage while applying quantization to the remaining dense weights, achieving desirable performance by reducing the quantization error significantly. AWQ [140] utilizes activation information to identify significant weights and optimizes per-channel scaling to preserve these important weights during quantization. AWQ surpasses existing methods on various language modeling and domain-specific benchmarks, exhibiting outstanding quantization performance for instruction-tuned and multi-modal LLMs. The Mixture of Formats Quantization (MoFQ) [141] selects the optimal format on a layer-wise basis, performing well in weight-only and weight-activation post-training quantization scenarios. MoFQ incurs no hardware overhead compared to INT/FP-only quantization. Omniquant [142] efficiently optimizes quantization parameters and preserves original full-precision weights with limited learnable parameters, performing well at low-bit scenarios. IntactKV [143] uses full-precision model to generate KV cache of pivot tokens, thereby effectively reducing the quantization error and achieving lossless weight-only INT4 quantization. PEQA [144] combines parameter-efficient fine-tuning with quantized LLMs. It updates only the quantization scales, minimizing memory overhead and model sizes. PEQA-tuned LLMs exhibit competitive performance in language modeling, few-shot learning, and comprehension, even at sub-4-bit precision.

These approaches mark significant progress in overcoming the challenges of quantizing LLMs, paving the way for more efficient and scalable language models that can be seamlessly deployed across diverse applications and platforms.

3) Knowledge Distillation: Researchers have also explored knowledge distillation (KD) techniques to compress LLMs into smaller, more deployable models while retaining their performance. These techniques leverage teacher-student paradigms, where a large teacher model transfers its knowledge to a smaller student model.

However, the optimization of large model distillation faces some challenges. One major challenge is the significant capacity gap between teacher and student models, leading to suboptimal distillation performance. Additionally, noisy pseudo-labels generated by the teacher model may negatively influence the distillation process. LLMs possess the capability for chain-of-thought reasoning, and it is crucial to transfer these reasoning abilities to smaller models through distillation.

Researchers have put forward innovative methods to improve knowledge distillation with large FMs. Distilling step-by-step [147] is a novel method with a multi-task framework that utilizes LLM rationales for additional supervision. It performs better with fewer training examples and enhances performance with smaller model sizes. Zephyr [148] employs distilled direct preference optimization to enhance user intent alignment in chat models, leveraging preference data from AI feedback. Lion [149] has three-stage adversarial loops, including imitation, discrimination, and generation, shifting knowledge from a sophisticated large LLM to a compact, open-source one. Program-aided Distillation (PaD) [150] utilizes reasoning programs to check errors in synthetic data for distillation, enabling small models to outperform large LLMs with significantly fewer parameters and training data. Distillspec [151] uses KD to align a compact draft model with a larger target model in speculative decoding. This method has a substantial reduction in latency with minimal performance drop. TED [152] aligns hidden representations and selects pertinent knowledge by task-aware filters, achieving notable advancements in compressing language models. Homodistil [153] mitigates large prediction discrepancies between teacher and student models. This approach initializes the student model from the teacher and gradually prunes neurons until reaching the desired width, ensuring consistent

knowledge transfer with minimal prediction discrepancies throughout the distillation process.

Li et al. [154] introduce Symbolic Chain-of-Thought Distillation (SCoTD) techniques, which equip smaller language models with the capability for chain-of-thought prompting, thus performing better in supervised and few-shot settings. Liu et al. [155] presents EvoKD, which leverages active learning to enhance data generation with LLMs, thereby improving the capabilities of smaller domain models. EvoKD integrates evolving knowledge distillation and active learning to optimize model training and distill informative knowledge effectively. Zhang et al. [156] introduce Minimal Distillation Schedule (MINIDISC), which aims to identify an optimal teacher assistant in a single trial for extreme compression scenarios, such as compressing to 5% scale. Slam [157] introduces the teacher model's noise to modify the student's loss function and improve the performance. Universalner [158] aims to train student models that excel in open named entity recognition, thus creating more cost-efficient student models. Scott [159] utilizes contrastive decoding to extract rationales that support gold answers from the teacher model. It also employs counterfactual reasoning to ensure faithful distillation in the student model. Scott generates more faithful CoT rationales compared to baselines while maintaining comparable end-task performance. Li et al. [160] instructs LLMs to transform structural triplets into context-rich segments and introduces auxiliary tasks for smaller knowledge graph completion (KGC) models, enhancing KGC models by leveraging LLMs. Recent advancements in knowledge distillation techniques have enabled smaller language models to effectively perform complex reasoning tasks.

C. Model Adaptation

Model adaptation aims to dynamically select and possibly adapt ML models for inference tasks based on the current execution context, such as available computational resources, network conditions, and specific requirements of the task. Dynamic model selection and adaptation enable elastic acceleration in edge-cloud systems. By exploring the trade-off between performance, efficiency, privacy, and cost, this approach significantly enhances the feasibility and effectiveness of AI applications across different edge-cloud scenarios.

As shown in Fig. 10, we categorize research works of model adaptation into three types. *1) Model Selection.* Model selection dynamically chooses an appropriate model for inference based on system characteristics. Key factors include hardware resources, input complexity, and service demands. Users' requirements for accuracy and latency can vary significantly based on their use cases. For instance, those using entertainment devices like virtual reality may prioritize low-latency AI services [167]. Conversely, users in the medical industry require impeccable accuracy [168]. The selector takes these factors as input and outputs the model selection strategy. *2) Model Iteration.* Model iteration progressively executes inference, starting with a small model and escalating based on prediction confidence to balance efficiency and accuracy. The existing methods are designed based on the entropy of the probability, where a higher entropy value indicates increased uncertainty and necessitates forwarding the request to a larger model. *3) Speculative Decoding.* Speculative decoding has emerged as an efficient and widely adopted technique in LLM inference to address the limitations of multi-step decoding. It utilizes a smaller model to generate a sequence of candidate words, which are then simultaneously verified by a larger model, resulting in improved performance.

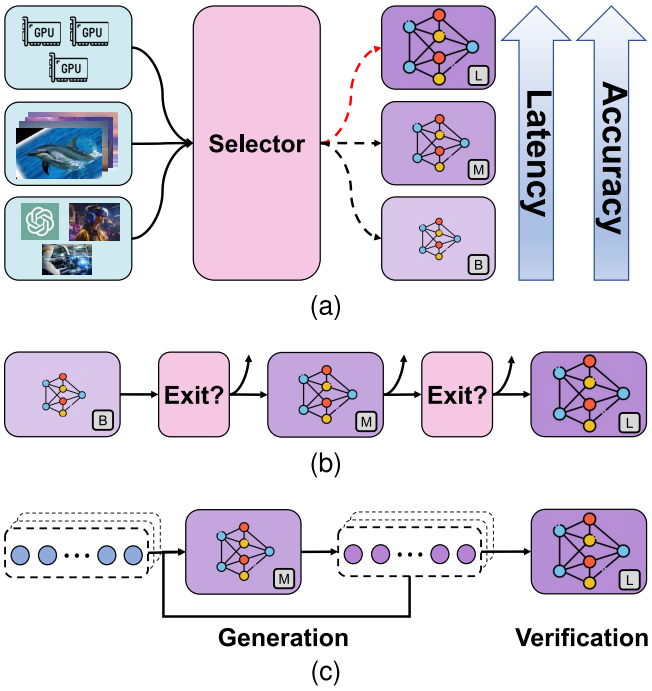

Fig. 10. The illustration of model adaptation methods. (a) Model Selection dynamically chooses among models of different sizes based on task complexity; (b) Model Iteration progressively escalates model capacity with early exit mechanisms; and (c) Speculative Decoding accelerates generation by letting a smaller model propose outputs which are then verified or corrected by a larger model.

We summarize recent works of model adaptation in Table V. INFaaS is an automated model-less serving system, which generates model variants optimized along different dimensions and automatically selects the most appropriate variant for each query based on performance, cost, and accuracy objectives [161]. The objective function optimizes for cost-efficient scaling: $\text{Minimize} \sum_{i,j} C_{ij}(\delta_{ij} + \lambda T_{\text{load},ij} \max(\delta_{ij}, 0))$, where C_{ij} is the hardware cost per second for model variant v_{ij}, $T_{\text{load},ij}$ represents the loading latency of the model variant, and λ is a tunable parameter balancing cost and response time. INFaaS employs a greedy heuristic that approximates the optimal scaling decision with sub-second response time. This heuristic estimates the current load capacity and selects between replicating an existing model or upgrading/downgrading to a more suitable variant based on cost-effectiveness. STI is an on-device inference system with two novel techniques: model sharding and elastic pipeline planning with a preload buffer [164]. STI manages model parameters as independently tunable shards, profiling their importance to accuracy and managing them on disk. Moreover, the elastic pipeline planning module utilizes a small preload buffer to initiate execution without delays, selecting and assembling shards according to their importance to

TABLE V
THE MODEL ADAPTATION METHODS

Type	Ref.	Selector/Exit	Target	Scenario
Model Selection	INFaaS [161]	A Greedy Heuristic	Accuracy & Latency & Recourse cost	Cloud
	Edgeadaptor [162]	Online optimization and approximate optimization	Accuracy & Latency & Recourse cost	Edge
	JellyBean [163]	Profile and a beam-search.	Accuracy & Throughput	Edge/Cloud
	STI [164]	A greedy method	Accuracy & Latency	Edge
Model Iteration	Tabi [165]	Confidence of logits	Accuracy & Latency	Cloud
	CATs [166]	A meta consistency classifier	Accuracy & Latency	Edge/Cloud

maximize inference accuracy within resource constraints. Tabi is an inference system that employs a multi-level inference engine to serve queries using smaller models by default and only switches to more computationally expensive LLMs for more complex applications [165]. Tabi uses a calibrated confidence score to decide whether the results from smaller models are accurate or if a query should be rerouted to a larger model for processing.

Leviathan et al. first introduce speculative decoding [169]. Let M_p be the target model with probability distribution $p(x_t|x_{<t})$, and M_q be an efficient approximation model providing a distribution $q(x_t|x_{<t})$. The process consists of the following key steps: 1. Generating speculative tokens. The approximation model M_q produces γ candidate completions $x_1, x_2, \ldots, x_\gamma$ autoregressively. These tokens are sampled from *q(x)*. 2. Validation by the target model. The target model M_p evaluates the generated tokens in parallel, computing their probabilities *p(x)*. A token x_i is accepted if: $q(x_i) \leq p(x_i)$. Otherwise, it is rejected with probability: $1 - \frac{p(x_i)}{q(x_i)}$. If a token is rejected, an additional token is sampled from an adjusted distribution: $p'(x) = \text{norm}(\max(0, p(x) - q(x)))$. 3. Efficiency analysis. The expected number of tokens generated per step is given by: $\mathbb{E}[\#\text{tokens}] = \frac{1-\alpha^{\gamma+1}}{1-\alpha}$, where α is the acceptance rate: $\alpha = \mathbb{E}_{x\sim q}[\min(1, \frac{p(x)}{q(x)})]$. This metric quantifies how well M_q approximates M_p, influencing the decoding speedup. LLMCad designs an on-device system with three novel techniques: constructing a token tree for broader candidate token pathways, a self-adjusting fallback strategy for error correction, and a speculative token generation during the verification process to maintain efficiency [170]. SpecInfer introduces a novel approach where small speculative models predict the LLM's outputs, organizing these predictions into a token tree, with each node representing a candidate token sequence [171]. The system then verifies the correctness of all candidate sequences in parallel against the LLM, significantly reducing end-to-end latency and computational requirements while maintaining model quality. Sequoia employs a dynamic programming algorithm to determine the optimal speculative token tree structure, enabling it to scale with the size of the speculation budget [172]. Sequoia also features a hardware-aware tree optimizer that selects the optimal token tree size and depth based on the available resources to maximize speculative decoding performance.

D. Token Adaptation

Although model adaptation can accelerate the inference of a transformer model, it may lead to large I/O latency because it needs to switch different versions of models between GPU memory and disk, which becomes the bottleneck during inference. By analyzing the inference process, researchers find that the significant computational cost of transformer models primarily comes at the self-attention mechanism [173]. It calculates the matrix multiplication between the key and query and scales quadratically with the sequence length. Processing long sequences or large batches of data can become computationally intensive because the number of tokens increases dramatically. Token reduction, encompassing token pruning, token merging and token summary, is an advanced technique in the field of artificial intelligence aimed at enhancing the efficiency and performance of large transformer models [174]. The core idea behind token reduction is to shorten the input data that a model processes, thereby reducing computational overhead and potentially improving the model's ability to focus on the most important information. A key research topic in token reduction is to identify redundant or similar tokens in the input and remove or merge them without negatively affecting model performance.

A number of token reduction methods have been developed to accelerate the inference of the transformer. *1) Token pruning.* There are only a limited number of words or image patches that contribute to the prediction of final results, and a lot of redundant tokens can be regarded as noisy information. Therefore, token pruning selectively removes tokens from the input sequence during the inference process of a transformer model. The core idea is to identify and retain only the most informative tokens for the task, thereby reducing the sequence length and the computational load. As shown in Fig. 11(a), there are two common approaches to indicate the significance of tokens that can guide the pruning process. The first method is *training a prediction module*, which takes tokens as input and outputs the importance score for each token. This module is constructed with a two-layer linear network and trained with soft masking of tokens. During inference, the *k* tokens with the lowest importance scores will be removed to reduce the token number. The second method is directly *collecting the attention weight* to prioritize tokens. The attention mechanism calculates the interdependence among tokens, and the attention weight reflects the relationship between them. Tokens with lower attention values contribute less to the output of other tokens, therefore indicating their smaller significance. Therefore, the attention weight serves as a valuable metric for evaluating the token importance.

2) Token Merging. Token merging, on the other hand, combines adjacent or similar tokens into single tokens, effectively condensing the content and further reducing the sequence

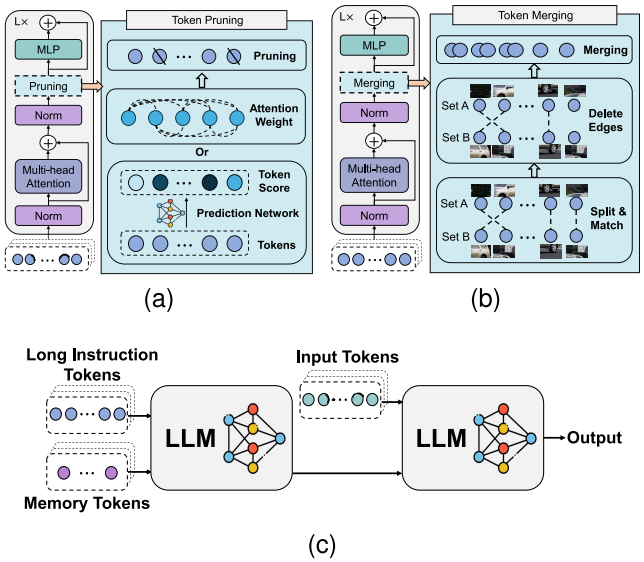

Fig. 11. The illustration of token reduction methods. (a) Token Pruning selectively removes low-importance tokens based on attention weights or learned prediction scores. (b) Token Merging compresses token sequences by identifying and merging semantically similar tokens. (c) Token Summarization leverages instruction or memory tokens to distill long sequences into compact representations.

length without a substantial loss of information. This technique is particularly useful for large transformer models, where processing extensive sequences of tokens can be computationally intensive. The motivation behind token merging is the presence of numerous similar patches in images and videos, which exhibit similar functionalities within the transformer model. One of the most well-known methods is TOME, which demonstrates the effectiveness of token merging [175]. As shown in Fig. 11(b), the input tokens are initially divided into two sets, with tokens in set A selecting the most similar token in set B using cosine similarity of features. The top k edges, representing the highest similarity of tokens, are retained. Finally, these tokens are merged using a weighted average approach to reduce the length of the input.

3) Token Summary. Token summary is a crucial technique for optimizing LLMs by condensing lengthy instructions and external knowledge sources. Instructions typically contain detailed task information and example references, while knowledge pools provide additional external information to assist the LLM's inference process. However, these sources can be voluminous, potentially exceeding the model's window size and causing latency issues. To address this, token summary expedites inference by summarizing the content into concise forms. It distills the content into smaller memory tokens using the LLM and concatenates them with user input tokens for inference. By leveraging token summaries, LLMs can effectively manage the overloaded information and enhance their ability to generate accurate responses.

Based on the application scenarios, we can classify existing token reduction methods into three types. *1) Language:* In recent years, several studies propose token reduction techniques specifically tailored for LLMs. The self-information, such as entropy and perplexity, is used to measure the token importance for pruning [176]. LLMLingua uses a small model to compress prompts, sets different compression ratios for instructions, questions and demonstrations, and iteratively prunes fine-grained tokens to improve the accuracy [177]. A few methods aim to reduce the size of the Key-Value (KV) caches to expedite the decoding process. The authors of Heavy Hitter Oracle (H2O) observe that a small subset of tokens, termed Heavy Hitters (H2), contribute most of the value when computing attention scores [178]. Based on this, they propose the H2O, a KV cache eviction policy based on the accumulated attention scores that dynamically retains a balance of recent and H2 tokens, effectively reducing the cache size without sacrificing performance. This method can improve the throughput by up to 29× on OPT-6.7B and OPT-30B compared to other leading inference systems. FastGen incorporates four policies to dynamically adjust the KV cache [179]. These policies are as follows: 1. Retaining special tokens like $<s>$ and $<INST>$. 2. Preserving punctuation tokens like "." and "?". 3. Evicting tokens that are distantly located from the current token. 4. Retaining tokens with high attention scores. StreamingLLM is an efficient method for deploying LLMs in streaming applications, such as multi-round dialogue [180]. StreamingLLM leverages an observed phenomenon called "attention sink", where maintaining the KV states of initial tokens significantly improves performance. This method can accelerate the inference of Llama-2 by up to 22.2× compared to other methods. Unlike traditional approaches that focus on sequence length, ThinK identifies redundancy in the channel dimension of the KV cache and employs a query-dependent pruning strategy to remove less significant channels [181]. DuoAttention introduces a method that categorizes attention heads into 'Retrieval Heads' and 'Streaming Heads', applying full key-value caching only to the former [182]. MagicPIG is a novel method that employs Locality Sensitive Hashing (LSH) sampling to approximate attention mechanisms in LLMs, significantly reducing computational overhead and memory usage [183]. For token summary, AutoCompressor and ICAE adopt compact summary vectors to distill information from the original contexts [184], [185]. Gist compresses instruction prompts into "gist" tokens that represent a specific task [186].

2) Vision. There are a lot of redundant tokens in an image, such as the background. DynamicViT introduces a trainable prediction module to determine the token scores. HeatViT deploys attention-based token pruning methods on FPGA devices and also incorporates hardware optimization strategies such as 8-bit quantization [187]. The motivation of STAR is to evaluate and prune patches based on their importance within and across layers [188]. It combines online intra-layer importance assessment with offline inter-layer importance analysis, using a fusion mechanism to selectively prune patches, aiming to retain those most critical for model performance. The idea of token merging originated from EViT [189], which divides tokens into attentive and inattentive tokens based on the attention weight and fuses inattention tokens into one token.

For token merging, TOME gradually combines similar tokens within a transformer, achieving a balance of speed and accuracy [175]. Recent works focus on combining token

pruning and token merging to improve accuracy. Long et al. delve into the importance and diversity among image patches to preserve discriminative tokens while maximizing global token diversity [190]. First, tokens are decoupled into two separate groups by leveraging class token attention. Inattentive tokens are then clustered using a density peak clustering algorithm, and attentive tokens are merged with a gentle matching method. TPS is also a token pruning and squeezing method [191]. First, it splits tokens into reserved and pruned subsets. Then, it employs nearest-neighbor matching and similarity-based fusing steps to combine the information from pruned tokens into the reserved tokens.

KV cache pruning for multimodal LLMs has emerged as a prominent research focus recently. FastV is a plug-and-play method designed to enhance the computational efficiency of multi-modal LLM [192]. It adopts a two-stage strategy: in the early layers, FastV analyzes and learns attention patterns to identify essential image tokens; in the later layers, it dynamically prunes less informative image tokens based on the learned patterns. LOOK-M introduces a text-prior compression strategy that prioritizes textual tokens over visual tokens to effectively reduce the KV cache size in multimodal long-context inference. To preserve important visual information, it further employs KV pair merging techniques that compensate for potential losses during compression [193].

Adaptive token reduction. Token reduction techniques demonstrate remarkable advantages in expediting transformer inference while maintaining prediction performance. Furthermore, it is crucial to develop adaptation methods for token reduction to cater to various applications across diverse execution environments. By doing so, we can fully leverage the potential of token reduction and optimize its effectiveness. OTAS incorporates token prompting and token reduction in an elastic serving system, enabling dynamic selection of an optimal token number [194]. To determine the optimal number of tokens for each batch, OTAS formulates an optimization problem aimed at maximizing the overall utility, which is a function of inference accuracy and computational efficiency. This optimization is subject to constraints such as user-defined latency requirements and available memory resources. The objective is to find a balance between the benefits of increased accuracy (through additional tokens) and the costs in terms of processing time and resource consumption. The optimization problem is solved using a dynamic programming algorithm. This algorithm efficiently allocates an appropriate number of tokens to each batch by considering factors like current system load, individual query characteristics, and predefined service-level objectives. By doing so, OTAS ensures that each batch is processed with a token number that aligns with both performance goals and resource constraints. AdaTape enhances the flexibility and performance of a transformer model by dynamically adjusting the computation based on the input's complexity [195]. It employs an elastic input sequence mechanism through adaptive tape tokens (i.e., prompt), which are generated from a tape bank and appended to the input sequences, allowing for dynamic read-and-write operations. This method enables the model to adaptively control both the content and the number of tape tokens used for each input, thereby adjusting the computational budget and potentially improving efficiency and performance on tasks.

E. Lessson Learned

The rapid development of FMs, particularly LLMs and MLLMs, has yielded critical insights for both research and deployment:

- Architectural Optimization Enhances Performance: The dominance of decoder-only Transformer architectures in LLMs (e.g., GPT series) underscores their superiority in few-shot learning. Meanwhile, ultra-large-scale LLMs like DeepSeek-V3 and Gemini highlight the advantages of MoE models. Innovations such as grouped query attention (e.g., LLaMA 2) or adding virtual expert modules (e.g., CogVLM) further demonstrate that tailored architectural modifications can significantly enhance performance in specific tasks.
- Training Models Requires High-Quality Data: Recent powerful models such as the Llama series, Qwen series, and DeepSeek series, all have utilized high-quality data for large-scale training, which highlights the crucial role of dataset quality in model training.
- Agent-Centered Model Design Balances Efficiency and Capability: Different demands on agent services necessitate model usability across various devices (e.g., Llama 3.2-1B for edge devices; Qwen-405B for cloud inference), and, under certain conditions, multimodal adaptability (e.g., visual-language alignment in Qwen-VL). Additionally, methods combining RL with CoT to reduce hallucinations have gradually become a hot direction in fields requiring complex reasoning (e.g., using DeepSeek-r1 or its distilled small models for homework tutoring).

V. Resource Allocation and Parallelism

To support the aforementioned agents and applications, this section discusses resource allocation and parallelization.

A. Resource Allocation and Scaling

Edge-cloud computing leverages strong cloud servers and distributed edge devices to handle tasks near the data source. As shown in Fig. 12, adaptive resource allocation is crucial for achieving an optimal balance between system performance and cost in edge-cloud environments. Resource management facilitates the optimal utilization of system resources, including processing power, storage space, and network bandwidth. Adaptive algorithms are designed to automatically adjust resource configurations based on real-time workloads and environmental changes, ensuring optimal performance under varying conditions.

Despite their numerous advantages, several challenges also arise in edge-cloud environments. 1) In edge-cloud environments, resources need to be allocated and managed between the central cloud and multiple edge nodes. Distributed resource management may increase the system's complexity and require more fine-grained scheduling and coordination mechanisms. 2) The load in edge computing scenarios is

TABLE VI
SUMMARY OF RESOURCE ALLOCATION AND ADAPTATION METHODS

Scenario	Ref.	Year	Target	Method
Cloud	Clipper [196]	17	Accuracy, Latency and Throughput	Model Containers.
	MArk [197]	19	Latency and Resource Cost	Predictive scaling.
	Nexus [198]	19	Latency and Throughput	Squishy bin packing.
	InferLine [199]	20	Latency and Resource Cost	1. The low-frequency planner. 2. The high-frequency tuner.
	Clockwork [200]	20	Latency	DNN Workers.
	INFaaS [161]	21	Latency, Throughput and Resource Cost	Dynamic Model Variant Selection and Scaling.
	Cocktail [201]	22	Accuracy, Latency and Resource Cost	1. Resource controller. 2. Autoscaler.
	Kairos [202]	23	Throughput	Query-distribution mechanism.
	SHEPHERD [203]	23	Throughput and Utilization	HERD: Planner for Resource Provisioning.
	SpotServe [204]	23	Resource Cost	Device Mapper.
Edge	Tong et al. [205]	20	Latency and Energy consumption	Deep Reinforcement Learning (DRL) Approach.
	Xiong et al. [206]	20	Latency and Utilization	Improved deep Q-network (DQN) Algorithm.
	CE-IOT [207]	20	Resource Cost	1. Delay-Aware Lyapunov Optimization Technique 2. Economic-Inspired Greedy Heuristic.
	Chang et al. [208]	20	Latency and Energy consumption	Online Algorithm for Real-Time Decision-Making.
	he et al. [209]	24	Latency and Utilization	Active Inference-based Algorithm
	xu et al. [210]	25	Resource Cost	1.test-time DRL algorithm 2.Double Dutch Auction Mechanism.

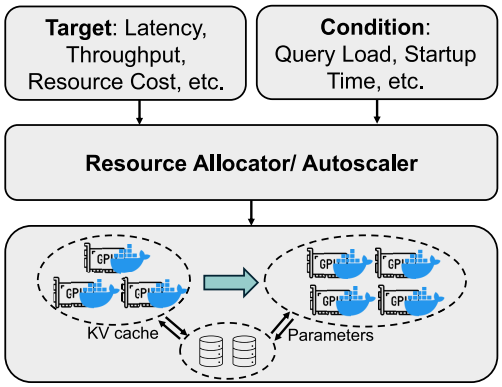

Fig. 12. The illustration of resource allocation. Resource allocation in a serving framework primarily involves dynamically adjusting the resource allocation strategy based on real-time resource conditions and query load.

highly dynamic. Accurately predicting this query load and dynamically adjusting resource allocation demands is difficult. 3) To meet the real-time processing requirements, the system must quickly adapt to changes in execution environments and adjust processing strategies and resource allocations in time. 4) Modern computing environments provide a variety of computing resources (CPU, GPU, TPU, etc.). Optimizing the allocation of these heterogeneous resources for various models based on their computational requirements and current workloads is a complex task.

We have summarized previous research works on resource allocation and adaptive optimization in Table VI. The collected research works are divided into two categories: optimization for model services in cloud environments and optimization for IoT applications in edge computing environments.

1) Cloud Environments. The following articles present various advanced solutions to optimize resource allocation in the cloud. Clipper uses model containers to encapsulate the model inference process in a Docker container [196]. It supports replicating these model containers across the cluster to increase the system throughput and utilize additional hardware accelerators for serving. Nexus adopts the squishy bin packing method to batch different types of tasks on the same GPU, enhancing resource efficiency by considering the latency requirements and execution costs of each task [198]. It also merges multiple tasks into the same GPU execution cycle as long as the latency constraints are not violated.

To cope with changes in query load, INFaaS adopts two automatic scaling mechanisms: vertical auto-scaling at the model level and horizontal auto-scaling at the Virtual Machine (VM) level [161]. The model auto-scaling is handled by the Model-Autoscaler, which decides each model variant's scaling operations (replication, upgrade, or downgrade) by solving an integer linear programming problem. The vertical auto-scaling adds a new VM if the utilization of any hardware resource exceeds a configurable threshold. Cocktail designs a resource controller to manage CPU and GPU instances in a cost-optimized manner and a load balancer to allocate queries to appropriate instances [201]. It also proposes an autoscaler that leverages predictive models to predict future request loads and dynamically adjusts the number of instances in each model pool based on the importance weight of the models.

Resource auto-scaling for LLMs poses new system challenges due to frequent instance preemptions and the necessity of migrating instances to handle these preemptions. Specifically, these challenges include efficiently migrating model parameters and KV cache to new instances to maintain service continuity and minimize performance degradation. Given the large size of LLMs, transferring parameters and cache state must be optimized to reduce latency and avoid excessive memory pressure. Besides, variable resource allocation requires dynamically adjusting the parallelism strategy of LLMs. SpotServe is a serverless LLM system that adjusts the GPU instances and updates the parallelism strategy flexibly [204]. When a new parallel configuration needs to be switched, SpotServe maps the available GPU instances to this logical device grid. It uses a bipartite graph matching algorithm to orchestrate model layers to hardware devices, thus maximizing the reusable model parameters and key-value

caches. ServerlessLLM is another serverless LLM system that introduces a loading-optimized checkpoint format and a multi-tier loading system to expedite model initialization [211]. It supports live migration of ongoing inferences, allowing the system to reassign tasks to different servers with minimal disruption. A locality-aware scheduler evaluates the status of each server in a cluster and effectively schedules model startup time to capitalize on local checkpoint placement. These systems are crucial for efficient agent deployment, as they enable adaptive resource scaling, rapid model initialization, and seamless inference migration. They ensure stable and responsive agent execution under dynamic cloud environments.

Traditional resource allocation strategies have predominantly relied on static replication and bin packing. Recent systems increasingly leverage predictive scaling, intelligent scheduling, and adaptive parallelism strategies. In highly dynamic cloud environments, the ability to rapidly migrate and reallocate resources while maintaining continuous inference is crucial—especially for LLM-based applications.

2) Edge Environments. Given the limited and heterogeneous nature of edge computing resources, efficient management and scaling of these resources are crucial. The following papers present efficient solutions to improve the performance and efficiency of edge systems. The authors of [209] propose an active inference-based algorithm to efficiently manage LLM inference tasks. The algorithm evaluates task types, available resources, and latency constraints to determine the most suitable server for offloading tasks. Its primary objective is to minimize overall latency and maximize task completion while strictly adhering to resource constraints. To enhance resource allocation efficiency, [210] proposes a test-time DRL algorithm that optimizes deployment and execution strategies for long-context LLM serving. This algorithm proactively manages cached models and service requests, adapting to context changes and usage patterns during execution. Additionally, it employs a Double Dutch Auction mechanism to dynamically adjust to the varying needs of mobile devices, edge servers, and cloud data centers, ensuring resources are allocated where they are most needed.

3) Resource Optimization for FM-based Agents. FM-based agents, particularly LLM agents, present distinct challenges and opportunities in resource allocation and parallelism. Leveraging FM capabilities, these agents are increasingly used for automated decision-making, contextual reasoning, and multi-step task execution. The deployment of LLM agents imposes stringent requirements on computational resources due to their large-scale architectures and inference complexity. Several challenges arise in resource allocation for these agents, including complex workflow adaptability, efficient multi-agent coordination, and latency constraints. LLM-based agent applications typically involve multiple interacting components, such as reasoning modules, retrieval-augmented generation, and external tool usage. These dynamic and often unpredictable workflows necessitate adaptive resource adjustments to efficiently allocate compute, memory, and bandwidth based on varying task demands. When multiple LLM agents operate concurrently, resource contention and scheduling conflicts can occur, requiring efficient allocation strategies to prioritize tasks, manage memory, and optimize inference pipelines. Additionally, many agent applications, such as real-time chatbots or automated decision systems, require low-latency responses, making it crucial to optimize GPU scheduling and memory access to minimize inference delays.

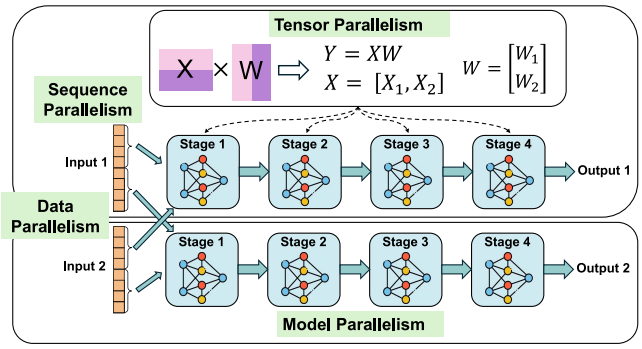

Fig. 13. The illustration of different parallelism methods. (1) Data Parallelism, where the same model processes different input batches in parallel; (2) Model Parallelism, which partitions the model across multiple devices in a pipeline fashion; (3) Sequence Parallelism, which distributes input sequences across compute nodes; and (4) Tensor Parallelism, which splits tensor operations across devices.

To address these challenges, advanced resource management strategies should be proposed. Adaptive resource scaling mechanisms are essential given the dynamic nature of LLM agent workflows, with workload-aware scheduling and dynamic GPU allocation helping balance compute loads and ensure efficient resource distribution based on demand. A robust scheduling framework, incorporating strategies such as priority-based task scheduling, reinforcement learning-based resource allocation, and preemptive scheduling, is vital to optimize multi-agent coordination and prevent bottlenecks. Furthermore, load balancers should dynamically distribute requests among different LLM agent instances based on resource utilization and response time predictions. Techniques such as query batching and multi-instance scheduling also mitigate GPU contention and improve efficiency.

B. Parallelism

1) Parallelism Type: The large FMs, such as GPT series and Llama series, and other transformer-based architectures, have significantly advanced the capabilities of AI applications. However, these models come with a substantial increase in computational requirements due to their size and complexity. Parallelism, including data parallelism (DP), model parallelism (MP), pipeline parallelism (PP), tensor parallelism (TP) and sequence parallelism (SP), is a critical design aspect that addresses the scalability, efficiency, and performance challenges associated with deploying and serving large-scale FMs.

As shown in Fig. 13, data parallelism is a technique where data is split into smaller batches and distributed across multiple processors. Different machines can execute the inference simultaneously, thus significantly improving the throughput. Model parallelism aims at splitting the model across different processors, with each processor responsible for a portion of

the model's layers or parameters. This approach is useful for very large models that cannot fit into the memory of a single processor. Model parallelism can be more challenging to implement than data parallelism because it requires careful partitioning of the model and management of the dependencies between layers. Pipeline parallelism includes data parallelism and model parallelism, where the data passes through the model in a sequential manner, transitioning from one processor to another as it traverses different layers of the model. Tensor parallelism is a more fine-grained approach than model parallelism. It splits the individual operations within a layer across multiple processors. For example, if a layer performs a large matrix multiplication, the computation of this matrix can be distributed across multiple processors. This approach can be particularly useful for operations that are computationally intensive and can be easily parallelized, albeit at the cost of increased communication volume. Sequence parallelism is a technique used to parallelize the processing of sequential data, such as text or time-series data, across multiple processors. In sequence parallelism, the sequence is partitioned into smaller chunks or segments, and each processor is responsible for processing a specific portion of the sequence. This approach can be particularly beneficial for long sequences that cannot fit into the memory of a single processor. Each processor handles computations for its assigned segment, and intermediate results are exchanged between processors for synchronization [212], [213]. Parallelism enhances performance by enabling simultaneous processing, significantly reducing execution time and increasing the scalability of agent applications.

There are some challenges when designing a parallelism strategy for FM in the edge-cloud environment. Memory Constraints: Large models may not fit into the memory of a single GPU, necessitating strategies to distribute them across multiple processing units. Computational Load: The volume of computations required for a single request can be substantial, requiring efficient distribution of computational tasks. Latency Requirements: Applications often require real-time responses, imposing strict latency constraints on the serving infrastructure. Scalability: The ability to serve a large number of concurrent requests without degradation in performance is crucial for ensuring user's satisfaction. Heterogeneous environment: Edge devices can vary widely in their computation and communication capabilities, operating systems, and available software. A parallelism strategy must be adaptable to different platforms and capable of optimizing execution depending on the specific characteristics of each device.

2) Auto-Parallelism: Existing research works design different automatic parallelism methods in cloud and edge scenarios. The key problem lies in determining an optimal partitioning scheme of a model and strategically allocating each stage to an appropriate device. NVIDIA Triton and Tensorflow-serving are two famous inference frameworks for machine learning models [214], [215]. Megatron-lm introduces an efficient intra-layer model parallel approach (i.e., tensor parallelism) that allows for the training of transformer models with billions of parameters without requiring new compilers or significant library changes, fully implementable in PyTorch [216]. DeepSpeed-inference is a comprehensive system designed to address the challenges of efficiently executing transformer model inference at large scales with heterogeneous memory systems, custom GEMM operations, and more [217].

LightSeq designs a sequence parallelism method for long-context transformers that performs partitioning solely the input tokens [218]. PETALS explores cost-efficient methods for inference and fine-tuning of LLMs on consumer GPUs that are connected by the Internet [219]. This approach could enable the pooling of idle compute resources from multiple research groups and volunteers to run LLMs efficiently. It introduces two main innovations: fault-tolerant inference algorithms and load-balancing protocols. SARATHI detects inference bubbles in pipeline parallelism caused by the imbalance (i.e., different execution times) between two distinct phases in the LLM: the prefill phase and the decode phase [220]. To tackle this issue, SARATHI addresses it through a decode-maximal batching approach and a chunked-prefills approach. This method divides a prefill request into equal-sized chunks and constructs a batch by utilizing a single chunk from the chunked-prefills and then populates the remaining batch slots with decode requests. DistServe enhances the performance of serving LLMs by disaggregating the prefill and decoding phases [221]. The disaggregation allows each phase to be assigned to different GPUs, eliminating interferences between prefill and decoding operations and allowing for tailored resource allocation and parallelism strategies for each phase. In summary, LLM deployment typically requires a coordinated approach that integrates multiple parallelism strategies to optimize performance across varying hardware and network conditions. Different parallelization methods must be dynamically adjusted based on specific requirements such as latency and throughput. LLM deployment often involves multi-node, multi-GPU setups, introducing complex environmental factors such as interconnect latency, heterogeneous hardware capabilities, and dynamic resource availability. As a result, systems must adaptively manage parallelism to balance computation and communication overhead. Furthermore, LLM inference consists of two distinct phases: the prefill phase and the decode phase. These phases have different compute and I/O demands, necessitating tailored parallelization techniques. Prefill is often compute-intensive and benefits from tensor and pipeline parallelism, whereas decode is more memory and bandwidth constrained, requiring sequence or expert parallelism.

Some research works design parallelism methods to deploy models on edge devices [222]. Galaxy introduces a hybrid model parallelism strategy that partitions Transformer models across multiple heterogeneous edge devices [212]. By implementing a heterogeneity-aware parallelism planning mechanism, Galaxy optimally distributes workloads based on each device's computational capabilities and memory constraints. Additionally, a tile-based fine-grained overlapping technique synchronizes communication and computation processes, reducing inference latency in bandwidth-limited edge environments. HexGen is a distributed inference engine designed to efficiently serve LLM across diverse GPUs and network connections [223]. HexGen supports asymmetric partitioning of inference computations, utilizing both tensor

and pipeline parallelism. It employs a sophisticated scheduling algorithm that adaptively assigns inference computations across GPUs with varying capabilities and network conditions. This algorithm formulates the allocation as a constrained optimization problem, balancing computational workloads and minimizing communication overhead. Helix is another distributed system designed to efficiently serve LLMs across heterogeneous GPU clusters [224]. It formulates the inference computation of LLMs as a max-flow problem on a directed, weighted graph, where nodes represent GPU instances and edges capture both GPU and network heterogeneity through their capacities. By employing a mixed integer linear programming algorithm, Helix discovers optimized strategies for model placement and request scheduling, jointly optimizing two entangled tasks.

C. Lessons Learned

The implementation and deployment of FM-based agents in real-world applications have provided several key lessons:

- Adaptive resource management is crucial: Static resource allocation strategies fail to meet the dynamic demands of LLM agents. Adaptive mechanisms that continuously adjust resources enhance efficiency and cost-effectiveness.
- Parallelism strategies must be optimized: Effective use of data, model, and pipeline parallelism significantly improves inference performance and reduces latency. Choosing the appropriate parallelism technique based on workload characteristics is essential.
- Energy efficiency cannot be overlooked: The high energy demands of large-scale LLM inference necessitate the use of energy-efficient execution techniques, including model pruning, quantization, and workload distribution.
- Scalability and fault tolerance are critical: Real-world deployment requires robust scaling mechanisms to handle fluctuating demand and fault-tolerant architectures to ensure uninterrupted service.

These insights provide a foundation for future research and development efforts, guiding improvements in resource allocation strategies, model optimization techniques, and system architectures for LLM agents.

VI. Execution Optimization

In this section, we introduce three low-level optimizations for the agent system.

A. Computation Optimization

Edge deployment of FMs poses severe challenges due to the heterogeneity of edge devices in terms of computing ability, hardware architecture, and communication bandwidth [225]. It is important to design computation optimization methods for different devices to accelerate model inference. Fig. 14 provides an overview of this chapter. This diagram provides a global perspective for algorithm design (e.g., computation optimization). It also depicts the components of an edge computing system, including heterogeneous backends and the network. At the hardware level, heterogeneous resources such as FPGAs, ASICs, IMCs, CPUs, and GPUs are utilized and optimized in terms of computation and memory. The integrated frameworks can support heterogeneous hardware. The network level focuses on semantic communication. VII shows some commonly used hardware devices at the edge. Traditionally, devices with CPUs have been deployed at the edge environment. However, due to their limited parallel computing capabilities and memory constraints, various specialized accelerators are designed for Deep Learning (DL) tasks. FPGAs have gained widespread attention for their programmable and parallel computing capabilities. ASICs, while non-programmable after manufacture, offer high speed and low power consumption. Many edge devices, such as personal computers and smartphones, have different computational resources, such as GPU and CPU. Consequently, many approaches focus on optimizing computational efficiency by jointly utilizing these resources for model inference. In-memory computing, with non-Von-Neumann architecture, has emerged as a prominent research area because its intrinsic parallelism can significantly reduce the I/O latency and improve computational efficiency. To enhance the understanding of different devices, we provide some detailed information about different hardware devices as below.

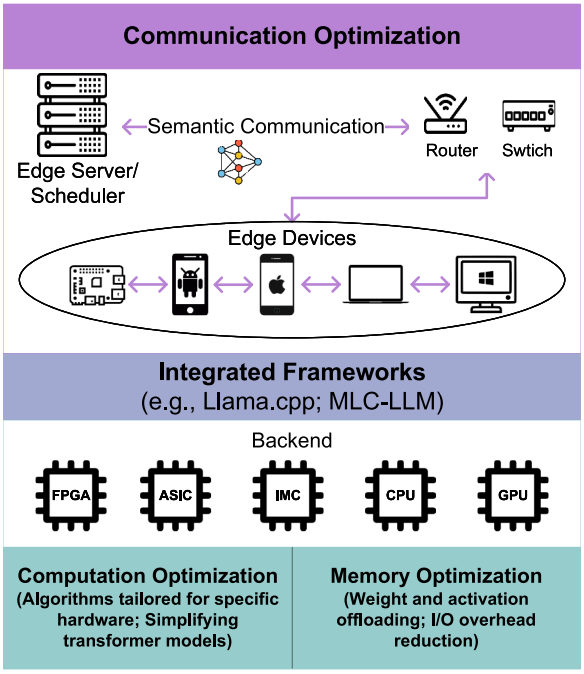

Fig. 14. Multi-layer optimization framework for edge deployment of foundation models. The top layer realizes semantic-aware communication optimization between the edge server and devices to adapt to varying network conditions; the middle layer are integrated frameworks within each device to utilize heterogeneous backends and expose FM inference interfaces; and the bottom layer implements hardware-specific computation and memory optimizations to maximize inference performance.

1) FPGAs: FPGAs are widely deployed in edge computing applications and are re-configurable hardware devices that are efficient in power consumption. FMs are built upon Transformers, whose attention mechanism is crucial for applications across NLP and CV. Transformers have

TABLE VII
COMPUTING RESOURCE TYPES AND FEATURES

Resource Type	Features and Functions
Field Programmable Gate Arrays (FPGAs)	FPGAs are integrated circuits that can be reconfigured at the hardware level after manufacturing. It can have higher performance and lower latency than GPU accelerators.
Application-Specific Integrated Circuit (ASIC)	ASIC is a customized accelerator for a specific use scenario. It is not programmable after manufacturing and offers computing comparable to FPGAs. The producer decides the specifications.
In-memory Compute (IMC)	IMC is a non-von-Neumann method, able to conduct computation in the memory. For instance, IMC utilizes physical processes to execute addition and multiplication, thus accelerating matrix-vector computing.
Central Processing Unit (CPU)	CPU is a General-purpose processor; It executes basic operations and instructions and can run lowly parallel computing.
Graphics Processing Unit (GPU)	GPU excels in parallel computing and is dominantly used as accelerators for DL.

non-linear computation components, such as layer normalization, SoftMax, and non-ReLU activation functions. Serving these models requires specific accelerator designs. On the other hand, the substantial computational complexity of matrix multiplications poses challenges for optimizing FPGA-based accelerators. To solve these problems, a specialized hardware accelerator is designed for MHA and FFN in Transformers [226]. It incorporates a matrix partitioning strategy to optimize resource sharing between Transformer blocks, a computation flow that maximizes systolic array utilization, and optimizations of nonlinear functions to diminish complexity and latency. MnnFast provides a scalable architecture specifically for Memory-augmented Neural Networks [227]. It incorporates a column-based streaming algorithm, zero-skipping optimization, and a dedicated embedding cache to tackle the challenges posed by large-scale memory networks.

Additionally, the development of Transformer-OPU (Overlay Processor) [228] provides a flexible and efficient FPGA-based processor that expedites the computation of Transformer networks. Moreover, implementing a tiny Transformer model through a Neural-ODE (Neural Ordinary Differential Equation) approach [229] leads to a substantial reduction in model size and power usage, ideal at the edge. FlightLLM [230] includes a configurable Digital Signal Processor (DSP) chain optimized for varying sparsity patterns in LLMs and always-on-chip activations during decoding and a length adaptive compilation method for dynamic sparsity patterns and input lengths.

In summary, the above designs for FPGA-based FM inference focus on several key aspects: enhancing computational efficiency through specialized architectures such as systolic arrays and DSP chains, optimizing memory usage via techniques like matrix partitioning and dedicated caches, reducing latency through efficient dataflows and model-hardware-aware optimizations, and improving adaptability and programmability to accommodate diverse and evolving model structures and non-linear functions.

2) ASIC: An ASIC is an integrated circuit chip customized for a specific application. For example, router ASICs can handle packet processing and signal modulation. It often includes microprocessors, memory, and other components as a System-on-Chip (SoC). Recent advancements in ASIC design significantly enhance the performance and efficiency of attention mechanisms. The A^3 [231] accelerator employs algorithmic approximations and a prototype chip to significantly enhance energy efficiency and processing speed. A^3 addresses the inefficiency of matrix-vector multiplication in the self-attention mechanism, by implementing an efficient greedy content-based candidate search method. Similarly, ELSA (Efficient, Lightweight Self-Attention) [232] tackles the quadratic complexity of self-attention by selectively filtering out less important relations in self-attention and implements an ASIC to achieve high energy efficiency.

SpAtten [233] prunes Transformer models at both the token and head levels and reduces the model size with quantization to minimize the computational and memory demands of Transformers. Sanger [234] framework enables sparse attention mechanisms through a reconfigurable architecture that supports dynamic software pruning and efficient sparse operations. Additionally, the Energon [235] co-processor, working together with other FM accelerators, introduces a dynamic sparse attention mechanism that uses a mix-precision multi-round filtering algorithm to optimize query-key pair evaluations. The above ASIC solutions demonstrate significant advancements in speed and energy efficiency compared to traditional devices while still preserving high accuracy. They pave the way for real-time, resource-efficient implementations for complex FMs.

3) In-Memory Compute: In-memory computing (IMC) is an emerging computational paradigm performing computational tasks directly within memory, eliminating frequent I/O requirements between processors and memory units. IMC is a scalable and energy-efficient solution to handle long sequence data in Transformers. Reference [236] introduces ATT, a fault-tolerant Resistive Random-access Memory (ReRAM) accelerator specifically designed for Attention mechanisms. To address the compatibility issues between traditional DNN and the complex data flow of attention mechanisms, they proposed a dedicated pipeline design to utilize the high-density storage capabilities and low leakage power of ReRAM, while reserving accuracy by adopting redundancy schemes. ReTransformer [237] is a ReRAM-based IMC architecture for Transformers. It accelerates the scaled dot-product attention mechanism, utilizes a matrix decomposition technique to avoid storing intermediate results, and designs the sub-matrix pipelines of MHA.

The iMCAT utilizes a combination of crossbar arrays to store the matrix in the memory array and uses Content Addressable Memories (CAM) to overcome the significant memory and computational bottlenecks in processing long sequences with MHA [237]. Recent works design different optimization methods to deploy computationally intensive Transformer models on edge AI accelerators [238]. The Google TPU, categorized as ASIC and IMC, is widely used in edge and cloud computing. Transformer models can be

executed efficiently on the Coral Edge TPU by optimizing the computational graph and employing quantization techniques, ensuring real-time inference with minimal energy consumption. The techniques employed in IMC, including matrix decomposition and quantization, reduce the computational resources required to serve FMs, thereby facilitating the deployment of FMs at the edge.

4) CPU and GPU: Current optimization algorithms, such as CPU-GPU hybrid computation, are hardware-independent and can be applied to various backends. Different methods are designed to accelerate Transformer inference by optimizing the MHA, FFN, skip connections, and normalization modules, which are identified as critical bottlenecks in Transformer architectures [239]. Reference [240] aims to optimize the execution time of the SoftMax layer by decomposing it into multiple sub-layers and then fusing them with adjacent layers. LLMA [241] leverages the overlap between the text generated by LLMs and existing reference texts to enhance computational parallelism and speed up the inference process without compromising output quality. These algorithms are hardware-agnostic and can be applied to various backends besides CPU and GPU. UltraFastBERT optimized inference by activating only a fraction of the available neurons, thereby maintaining competitive performance levels [242]. Intel CPU clusters can serve LLMs by adopting the algorithms [243], such as quantization and optimized kernels, specifically designed to accelerate computations on CPUs.

Many optimizations have been proposed by coordinately utilizing GPUs and CPUs to improve the efficiency and speed of Transformer models. Similarly, [244] introduces PowerInfer, a high-speed LLM inference engine that leverages the high locality and power-law distribution in neuron activation to reduce GPU memory demands and CPU-GPU data transfers. Reference [245] aims to reduce latency by coordinating CPU and GPU computing while mitigating the I/O bottlenecks by overlapping the data processing and I/O time. Reference [246] utilizes a single GPU to serve LLMs, prioritizing throughput at the expense of latency. It designs a scheduling algorithm to schedule model parameters, KV caches, and activations between CPU, GPU, and disk. These advancements focus on hardware-independent algorithms and CPU-GPU collaborative inference strategies, signifying a robust movement towards more efficient FMs.

B. Memory Optimization

1) Vanilla FMs: Besides computational costs, memory overhead is another major challenge in deploying LLMs at resource-constrained edge devices. The memory wall primarily prevents the loading of vast LLMs parameters and the maintenance of KV cache throughout inference.

Contextual sparsity, which refers to the input-dependent activation of a small subset of neurons in LLMs, has been exploited to reduce computational and memory costs. Reference [247] proposes a method to predict contextual sparsity on the fly and an asynchronous, hardware-aware implementation to accelerate LLM inference. Reference [248] explores storing model parameters in flash memory and loading only the required subset during inference, considering bandwidth, energy constraints, and read throughput.

Several studies have focused on optimizing the attention mechanism to reduce I/O costs. Reference [249] introduces the Multi-Query Attention to accelerate decoding by reducing memory bandwidth requirements during incremental inference. Reference [250] proposes Grouped-Query Attention (GQA), an intermediate between MHA and Multi-Query Attention (MQA), achieving a balance between quality and speed. Reference [251] introduces PagedAttention, which is inspired by virtual memory and paging techniques, to efficiently manage key-value cache memory and integrate it into vLLM, a high-throughput LLM serving system. Reference [252] proposes FlashAttention, an I/O-aware attention algorithm that reduces memory accesses between High Bandwidth Memory (HBM) and on-chip Static Random-Access Memory (SRAM) in GPU, which avoids fully loading the large attention matrix, reducing I/O overhead significantly. FlashDecoding++ [253] introduces asynchronized partial parallel SoftMax. Subtracting the maximum value of input can avoid the denominator overflow of SoftMax, but brings synchronization overhead. FlashDecoding++ addressed this by utilizing a unified max value predecided according to the statistical LLM-specified input distribution.

Reference [254] proposes Splitwise, a technique that schedules the prompt computation and token generation phases to different machines, optimizing hardware utilization and overall efficiency. FastServe is a distributed serving system that optimizes job completion time for LLMs in interactive AI applications [255]. Reference [171] introduces SpecInfer, a system that accelerates generative LLM serving through tree-based speculative inference and verification. LLMCad is an innovative on-device inference engine specifically designed for efficient generative NLP tasks [170]. It executes LLM on weak devices with limited memory capacity by utilizing a compact LLM that resides in memory to generate tokens and a high-precision LLM for validation.

2) MoE FMs: Mixture-of-Experts (MoE), replacing an FFN with a router and multiple FFNs, is a promising approach to enhance the efficiency and scalability of LLMs. However, deploying MoE-LLMs presents challenges in memory scheduling because of vast parameters and contextual expert routing, particularly in resource-constrained environments. Recent research has addressed these challenges through novel architecture designs, model compression techniques, and efficient inference engines. Reference [256] proposes DeepSpeed-MoE, an end-to-end solution for training and deploying large-scale MoE models. Reference [257] introduces EdgeMoE, an on-device inference engine designed for MoE-LLMs that loads popular experts to GPU to prioritize memory and computational efficiency. Reference [258] proposes a novel expert offloading strategy utilizing the intrinsic properties of MoE-LLMs. It uses Least Recently Used (LRU) caching to store experts since certain experts are reused across consecutive tokens. It also accelerates the loading process by predicting the selection of experts for future layers based on the hidden states of earlier layers.

Several studies have focused on developing efficient serving systems and inference engines for MoE-LLMs. Reference [259] introduces MOE-INFINITY, an efficient MoE-LLMs serving system that implements cost-efficient expert offloading through activation-aware techniques, significantly reducing latency overhead and deployment costs. Reference [260] proposes Fiddler, a resource-efficient inference engine that orchestrates CPU and GPU collaboration to accelerate the inference of MoE-LLMs in resource-constrained settings.

3) Memory Management: Traditional memory management techniques have also been adapted for LLM inference optimization. llama.cpp [261] leverages memory mapping (mmap) to dynamically load model weights into RAM, enabling single-device inference of models exceeding available RAM capacity. dllama [262] also leverages mmap in a distributed way that only loads model weight from the master node and sends partitioned model weight to worker nodes, hence reducing disk requirement. Furthermore, TPI-LLM [263] exploits the layer-wise execution pattern of LLM by overlapping sliding window weight loading with computation phases, effectively reducing memory pressure through pipelining while not increasing the end-to-end latency.

C. Communication Optimization

1) Resource Scheduling and Network Optimization: In addition to computation and memory optimization, communication overhead is another major challenge in deploying large models within the edge-cloud environment.

The heterogeneity of edge devices and communication channels necessitates a scheduler for computation and network resources when executing FM tasks across different devices. Reference [264] introduces an "Intelligence-Endogenous Management Platform" for Computing and Network Convergence (CNC), which efficiently matches the supply and demand within a highly heterogeneous CNC environment. The CNC brain is prototyped using a deep reinforcement learning model. It theoretically comprises four key components: perception, scheduling, adaptation, and governance, collectively supporting the entire CNC lifecycle.

Collaborative inference across edge-cloud devices involves vast optimization search spaces. Reference [265] discusses the integration of LLMs into 6G vehicular networks, focusing on the challenges related to computational demands and energy consumption. It proposes a framework where vehicles handle initial LLM computations locally and then offload intensive tasks to Roadside Units (RSUs). The authors formulate a multi-objective optimization framework that aims to minimize the cumulative cost incurred by both computational tasks and edge-cloud communication. Edgeshard [266] is a collaborative framework to distribute LLMs on heterogeneous edge servers and devices. It formulates an optimization problem under bandwidth, computing, and memory constraints and proposes a dynamic programming algorithm for optimizing latency and throughput. Reference [267] tries to improve the edge LLM service quality using historical interactions on the cloud as the external datastore. To tackle the challenge of the datastore delay, they propose a subset selection algorithm to optimize the datastore size and achieve balanced quality and efficiency. LinguaLinked [268] is a system to deploy LLMs on distributed mobile devices. The high memory and computational demands of these models typically exceed the capabilities of a single mobile device. It utilized load balancing and ring topology to optimize the delay of computation and communication while maintaining the original model structure. The results demonstrate improvements in inference throughput on mobile devices.

2) Semantic Communication and Model-Specific Optimization: Semantic communication enables much more efficient utilization of limited network resources at the edge by only transmitting the essential semantic information. However, the key challenge is the lack of closed-form expression for semantic reservation. To address this, [269] investigates multiple access designs to facilitate the coexistence of semantic and bit-based transmissions in future networks. The authors proposed a data regression method to approximate the semantics and designed a heterogeneous framework where an access point simultaneously sends semantic and bit streams to a semantics-interested user and a bit-interested user. Reference [270] presents a framework for semantic communications enabled by computing networks, aiming to provide sufficient computing for DL-enabled semantic functions. This framework introduces key techniques, such as semantic sampling and reconstruction, semantic-channel coding, and semantic-aware resource allocation and optimization based on cloud-edge-end computing coordination. To demonstrate the advantages of the proposed framework, authors provided two use cases, an end-cloud computing-enabled video transmission system, and a semantic-aware task offloading system.

Semantic communication proves particularly valuable for LLMs, where activation values during inference exhibit skewed distributions with inherent sparsity patterns [247]. It demonstrated that up to 50% of neurons in Transformer layers can be skipped without accuracy loss, suggesting significant potential for activation compression. Employing mixed-precision quantization schemes can preserve fidelity for semantically important activations while aggressively compressing others [271]. On the other hand, [272] pioneers direct activation communication between LLM agents, bypassing natural language decoding entirely by fusing intermediate layer activations through learned combination functions.

D. Integrated Frameworks

1) Optimization Coordination: Inference Phase Characteristics. The LLM inference process can be decomposed into three critical phases: a) The model loading phase requires loading parameters from storage, exhibiting memory-intensive characteristics; b) The prefill phase processes entire input sequences, exhibiting a computational complexity that scales quadratically with sequence length; c) The decoding phase generates tokens autoregressively while being memory-bound and possibly compute-bound at the low-end edge devices. Besides, the distributed decoding phase

TABLE VIII
INTEGRATED FRAMEWORKS

Framework	Features and Functions	Backend
LLaMA.cpp [261]	High-performance inference of LLaMA and other LLMs	CPU(x64, ARM), GPU(CUDA, ROCm, Metal), OpenCL, Vulkan
MLC-LLM [273]	Accelerate LLMs inference using Machine Learning Compilation	GPU(CUDA, ROCm, Metal, Vulkan, WebGPU), OpenCL
MNN-LLM [274]	Inference for Mobile Neural Network LLMs	CPU(x64, ARM), GPU(CUDA), OpenCL
FastChat [275]	Platform for both training and inference of LLM-based chatbots	CPU(x64), GPU(CUDA, ROCm, Metal)
DeepSpeed [217]	DeepSpeed-Inference integrates multiple parallelisms and custom kernels, communication, and memory optimizations.	CPU(x64, ARM), GPU(CUDA, ROCm)
OpenVINO [276]	Convert FMs and deploy them on Intel hardware	CPU(x64), GPU(OpenCL)
MLLM [277]	Mobile LLMs Inference	CPU(x86, ARM)
FP6 [278]	Mixed precision for LLMs	GPU(CUDA)
Colossal-AI [279]	Distributed training and inference using multiple parallelisms	GPU(CUDA)
Megatron-LM [280]	Efficient LLMs inference on GPUs with system optimizations	GPU(CUDA)
TensorRT-LLM [281]	NVIDIA TensorRT for LLMs inference	GPU(CUDA)

is also communication-bound. Each phase presents distinct bottlenecks.

Coordinated Optimization Methodology. Modern optimization approaches achieve global efficiency improvements through coordinated optimization for every phase. Cross-Phase optimizations such as a) Memory mapping (e.g., llama.cpp's mmap) simultaneously accelerates model loading and KV cache access during decoding, b) Quantization strategies enable memory optimization across all phases at the cost of de-quantization delay and precision reduction. c) Semantic communication accelerates both prefill and decoding phase. Phase-specific optimizations address particular bottlenecks: a) Parameter sharding with progressive de-quantization for loading phase, b) Computational kernel optimizations like FlashAttention for prefill phase, and c) Speculative decoding accelerates decoding phase. Sequentially, each inference phase can incorporate compatible optimization techniques. Specifically, quantization strategies effectively reduce memory consumption during the model loading phase; subsequently, FlashAttention accelerates the prefill phase while KV cache quantization minimizes cache memory requirements. Finally, memory-mapped KV cache access enhances Speculative decoding performance during the decoding phase. This coordinated implementation of compatible optimization methods yields comprehensive end-to-end efficiency improvements.

2) Implementation Frameworks: With the emergence of computational and memory optimization methods, numerous integrated frameworks have integrated these techniques, supporting various LLMs and lowering the barriers to LLM deployment. These frameworks leverage CPU and GPU backends, enabling the widespread local execution of LLMs.

The most popular framework to deploy LLMs at the edge is the llama.cpp [261], which pioneered the open-source implementation of LLM execution using only C++. We provide an overview of various open-source frameworks and their respective features in VIII. The frameworks can be categorized into heterogeneous and GPU-only backends, ranging from mobile and embedded devices to high-end computing systems. Different suppliers offer specialized development platforms and APIs for their hardware products (i.e., CPU and GPU). CUDA (Compute Unified Device Architecture) is a parallel computing platform developed for NVIDIA GPU. ROCm (Radeon Open Compute platform) is a software platform for high-performance computing using AMD GPU. Metal is a low-overhead hardware-accelerated 3D graphic and compute shader API developed by Apple for its own GPUs. Some frameworks like Vulkan and OpenCL (Open Computing Language) facilitate development across different hardware and operating systems. Vulkan is an API for cross-platform access to GPUs for graphics and computing. OpenCL is a framework for writing programs that execute across heterogeneous backends, including CPU, GPU, FPGA, and other hardware.

Recently, several novel frameworks have been specifically designed for the deployment of LLM-based applications/agents. LangChain is a framework aimed at simplifying the development of applications that interact with LLMs [282]. It offers a set of tools and abstractions that enable developers to construct complex, dynamic workflows by chaining various operations, such as querying a language model, retrieving documents, or processing data. SGLang is another LLM agent framework designed to efficiently execute complex language model programs [283]. It simplifies the programming of LLM applications by providing primitives for generation and parallelism control. Additionally, it accelerates the execution of these applications by reusing KC caches, enabling faster constrained decoding and designing API speculative execution.

E. Execution Optimization for Agents

1) Distinctions from Traditional DNN Optimization: The optimization of FMs diverges fundamentally from conventional DNN acceleration due to three paradigm-shifting characteristics of modern FMs: a) Architectural Complexity: Unlike DNNs with localized operations, Transformers employ global self-attention mechanisms with quadratic complexity for sequence length. This necessitates hardware-aware algorithms like FlashAttention [252] to optimize memory access patterns, contrasting with the static kernel optimizations for DNNs. b) Memory-Centric Workloads: While DNNs need less parameters, FM inference is bottlenecked by massive memory footprints and growing KV caches. This drives innovations like PagedAttention's memory management [251] and EdgeMoE's expert prefetching [257]. c) Dynamic Autoregressive Execution Patterns: Autoregressive generation introduces strict sequential dependencies absent in batch-oriented DNN inference. This prevents effective GPU saturation, necessitating speculative decoding [284]. FMs

exhibit input-dependent computation graphs, where contextual sparsity enables 30-50% neuron skipping and MoE routers dynamically activate experts [247], and opportunities for semantic communication [272].

These distinctions render traditional DNN optimizations insufficient. Modern FM acceleration requires co-design across algorithm-hardware-architecture: hardware-specific attention operators, dynamic memory management for agent contexts, and semantic-aware communication for distributed inference.

2) Agents Deployment: The deployment of agents powered by FMs at the edge introduces unique challenges that demand tailored optimization strategies. These challenges stem from three intrinsic characteristics of agent services:

High Computational Demand. Agents often require multi-step reasoning and real-time interaction, necessitating hardware-aware operator optimization. Techniques like FPGA-based systolic arrays and ASIC-accelerated sparse attention reduce latency for complex workflows. CPU-GPU hybrid frameworks (e.g., PowerInfer [244]) enable dynamic computation partitioning between prompt processing and token generation, crucial for handling agents' variable-length reasoning paths.

Memory-Intensive Workloads. Agents maintain long-term context and access external knowledge bases, requiring advanced memory management. KV cache page-wise management via PagedAttention [251] and activation offloading address the memory wall for persistent context.

Unreliable Edge Networks. The interactive nature of agents requires robust communication under fluctuating bandwidth. Semantic communication frameworks optimize transmission by prioritizing critical data (e.g., activation vectors, tool-calling parameters) over redundant tokens, thereby reducing payload size [272].

These optimization approaches collectively enable agents to overcome edge constraints: hardware-specific computational kernels enhance processing pipeline efficiency, memory management systems facilitate dynamic knowledge retrieval and caching, while semantic-aware networking ensures efficient communication. The convergence of these techniques establishes a paradigm in optimizations for FM-based agents.

F. Lessons Learned

The execution optimization of FM-based agents at the edge reveals several critical insights for researchers:

- Hardware-specific optimization is paramount. Diverse hardware architectures require distinct optimization approaches due to varying instruction sets, memory hierarchies, and programmability. Developing hardware-tailored strategies is necessary for efficiency. For example, FPGA acceleration requires customized dataflow.
- Phase-aware coordination improves efficiency. FM inference exhibits phase-dependent bottlenecks: compute-heavy prefill vs memory-bound decoding. Phase-specific FlashAttention accelerates prefill by I/O-aware taming of quadratic attention. Cross-phase mixed-precision quantization can benefit both phases.
- Resource constraint tradeoffs. Fixed edge resources necessitate trade-offs between efficiency and accuracy. Quantization reduces memory usage and improves computational efficiency, but incurs a loss in accuracy. Semantic-aware networking provides bandwidth-efficient communication at the cost of potential accuracy loss.

These lessons highlight that FM-based agent demands integrating optimization across hardware, algorithms, and system design rather than isolated improvements.

VII. Lesson Learned and Future Work

A. Overall Lesson Learned

1) Elastic Agent Serving System: A key lesson learned from examining the current serving systems for agents is the significant gap in elasticity at the agent layer. While substantial progress has been made to introduce elasticity at the levels of the resources, models, or execution tokens, there remains a notable lack of dynamic adaptability within the agent itself. The agent layer should be more flexible by incorporating adaptivity in API calls, external knowledge retrieval, reasoning capabilities, and multi-agent collaboration.

- Firstly, an adaptive agent could decide when it needs to access external databases or APIs for more specialized information, or when it should rely on internal reasoning capabilities to make decisions autonomously.
- Additionally, instead of hardcoded processes for knowledge integration, agents could be equipped with mechanisms to interact with evolving knowledge repositories and search from different levels of knowledge databases.
- Besides, agents could be adaptive in reasoning and multi-agent collaboration. Current systems tend to employ static reasoning approaches, where the agent either follows a predefined logic or relies entirely on a single model's output. In contrast, a more flexible agent could dynamically choose between different reasoning strategies depending on the task complexities, and potentially break down a problem into subtasks to be handled by specialized agents within a collaborative network. This multi-agent coordination could allow the system to generate more scalable and robust solutions.

Without this level of flexibility, LLM agents remain limited in their ability to handle a broad spectrum of tasks. As more complex and specialized applications arise, the lack of agent-layer flexibility becomes a bottleneck, preventing these systems from fully realizing their potential. Thus, future work in this field should prioritize developing more flexible, adaptable agents capable of handling diverse, evolving tasks in real-world environments.

2) Workflow-Aware Resource Scheduling for Agent Deployment: In deploying AI agents, it's crucial to recognize that their operation encompasses various workflows, such as reasoning, retrieval-augmented generation (RAG), and multi-agent interactions. Each of these workflows has distinct model interactions and resource requirements. However, current resource scheduling frameworks often lack awareness of these specific workflows, leading to suboptimal global resource

allocation and scheduling. For instance, without insight into an agent's workflow, the system may fail to proactively allocate resources, load necessary models, or manage cache storage efficiently. This oversight can result in increased latency and inefficient resource utilization. By integrating workflow awareness into resource scheduling, systems can optimize global scheduling decisions. Understanding the agent's workflow allows for proactive resource scheduling. This approach enhances efficiency and responsiveness.

Incorporating workflow awareness into resource scheduling not only optimizes performance but also enhances the system's ability to handle complex, dynamic tasks. As AI agents become more integral to various applications, adopting such advanced scheduling strategies will be essential for achieving scalable and efficient deployments.

3) Development of Multimodal Agents: The development of multimodal agents represents a significant advancement beyond traditional text-based AI systems. Currently, many agents operate primarily within the text modality, and even those that incorporate multimodal elements often convert inputs into text before applying text-based reasoning or RAG techniques. This approach, while effective to a degree, limits the potential of truly multimodal interactions.

A genuine multimodal agent would seamlessly integrate various data types, including images, videos, and audio, enabling it to process and generate content across these diverse formats. This integration would allow for reasoning capabilities that are inherently based on different modalities, rather than relying solely on textual representations. For instance, such an agent could analyze a video to extract contextual information, interpret audio cues to understand emotional tones, and combine these insights with textual data to provide more comprehensive and nuanced responses.

Despite these prospects, developing truly multimodal agents presents challenges. Integrating and processing diverse data types require sophisticated models capable of understanding and reasoning across different modalities. Additionally, ensuring seamless interaction between these modalities to produce coherent outputs is a complex task that necessitates further research and development.

4) Heterogeneous Edge Computing for FM Inference: Each edge computing device is uniquely designed to meet specific requirements. Consequently, different accelerators exhibit trade-offs in flexibility, memory capacity, and computational efficiency, influencing their suitability for FM inference in edge scenarios.

- ASICs, while optimal for fixed model architectures in FMs, are immutable post-production, limiting their adaptability to new models. FPGAs offer greater flexibility through programmability, supporting various model sizes. However, they are primarily suited for linear computations, necessitating additional design considerations for non-linear operators prevalent in transformers.
- TPUs have limited memory for FMs and constrained precision and performance due to their integer operation support, making them suboptimal for direct FM inference.
- CPUs, while significantly less powerful than other accelerators, offer larger memory. This characteristic enables offloading some computation workload and part of model caches to CPUs, facilitating collaborative inference with other accelerators.
- GPUs benefit from a mature CUDA ecosystem, making them suitable for out-of-the-box FM inference. Consequently, numerous research efforts focus on accelerating GPU inference in data centers. However, their high cost and energy consumption hinder widespread adoption in edge scenarios. This limitation has spurred interest in edge GPU devices, exemplified by NVIDIA Jetson, which has limited computing capacity and enough memory.

In conclusion, optimizing heterogeneous edge computing for various types of FMs remains an under-explored field, as most existing designs primarily focus on LLMs. Researchers have generally assumed a fixed transformer architecture for FMs, overlooking hybrid models such as ViT-LLM, which are increasingly popular in multi-modal FMs. This narrow focus has led to solutions that are highly tailored to specific models but suboptimal for others, limiting their adaptability in real-world deployment scenarios.

Moreover, most existing approaches do not address inter-accelerator optimization, such as parallelism strategies and communication efficiency between heterogeneous devices. Instead, they focus on hardware-specific optimizations. However, in LLM-based agent deployment, leveraging the complementary strengths of different accelerators—such as offloading memory-intensive tasks to CPUs, utilizing GPUs for parallel processing, and incorporating FPGAs or TPUs for low-latency inference—can significantly enhance overall system efficiency. A well-designed optimization framework that considers workload distribution, synchronization, and cross-device communication can lead to more effective and scalable FM inference on edge devices.

B. Future Directions

1) Enhancing Elasticity in Agent Serving Systems: To address the current limitations in agent-layer elasticity, future research should focus on developing agents capable of dynamic adaptability. This includes:

- Adaptive API Utilization: Designing agents that can intelligently decide when to access external databases or APIs for specialized information versus relying on internal reasoning capabilities.
- Dynamic Knowledge Integration: Equipping agents with mechanisms to interact with evolving knowledge repositories, enabling them to search and retrieve information from various levels of knowledge databases as needed.
- Flexible Reasoning Strategies: Implementing agents that can dynamically choose between different reasoning approaches based on task complexity, enhancing their problem-solving efficiency.
- Multi-Agent Collaboration: Facilitating systems where agents can break down complex problems into subtasks and collaborate with specialized agents within a network, leading to more scalable and robust solutions.

2) Implementing Workflow-Aware Resource Scheduling for Agent Deployment: To optimize resource allocation and scheduling in AI agent deployment, it is essential to integrate workflow awareness into resource management frameworks. Future efforts should concentrate on:

- Proactive Resource Allocation: Developing systems that understand an agent's specific workflow, allowing for the proactive scheduling of resources, loading of necessary models, and efficient cache management.
- Dynamic Resource Management: Creating frameworks that can adjust resource distribution in real-time based on the agent's workflow demands, reducing latency and improving performance.

By adopting workflow-aware resource scheduling, AI agent systems can achieve enhanced efficiency, responsiveness, and scalability in handling complex, dynamic tasks.

3) Advancing the Development of Multimodal Agents: To move beyond the limitations of text-based AI systems, future research should focus on creating truly multimodal agents capable of seamless integration and processing of diverse data types. Key areas of development include:

- Integrated Multimodal Processing: Designing agents that can concurrently process and generate content across various formats, including text, images, videos, and audio, without defaulting to text-based representations.
- Modality-Specific Reasoning: Implementing reasoning capabilities that are inherently based on different modalities, allowing agents to analyze and interpret data in a native form, such as extracting contextual information from videos or understanding emotional tones from audios.
- Seamless Modality Interaction: Ensuring that agents can fluidly combine insights from multiple data types to provide comprehensive and nuanced responses, enhancing their applicability in complex, real-world scenarios.

By focusing on these areas, the development of multimodal agents can lead to more versatile AI systems capable of handling a wide range of tasks across different data formats.

4) Efficiently Deploying FM-Based Agents on Heterogeneous Devices: Efficiently deploying FM-based agents across heterogeneous edge-cloud environments requires a strategic combination of edge computing and cloud resources to enhance performance, scalability, and cost-effectiveness. Current research primarily focuses on optimizing small models for edge devices or deploying large models in cloud environments. However, efficiently deploying large FMs on heterogeneous edge-cloud infrastructures remains an open challenge.

To address these issues, advanced model compression techniques, resource scaling strategies, and dynamic workload distribution methods must be explored. Additionally, leveraging various types of hardware accelerators, such as GPUs from NVIDIA and Ascend chips from Huawei, can optimize performance across diverse infrastructures.

C. Challenges for Future Directions

Deploying FM-based agents on heterogeneous edge-cloud environments introduces a range of challenges that must be addressed to enhance efficiency, scalability, and adaptability. One of the key issues is the heterogeneous nature of edge devices, which vary significantly in terms of computational power, memory, and energy constraints. Agents running on these devices must be dynamically allocated based on available resources, ensuring an optimal balance between computation, communication, and energy efficiency. Furthermore, real-time adaptability is crucial for agent-based applications, requiring low-latency inference and decision-making mechanisms to function effectively in dynamic environments.

Another major challenge is the complexity of multi-modal and MoE models. Multi-modal models, which integrate diverse input types, often require specialized encoders and decoders that must be efficiently scheduled to avoid computational bottlenecks. Similarly, MoE models dynamically activate expert modules based on input characteristics, making real-time scheduling and resource allocation critical. Serving systems must be designed to handle these dynamic workloads while maintaining high performance and reliability.

Agents operating in distributed settings must also contend with communication overhead and data synchronization issues. In multi-agent systems, coordination between agents is essential for collaborative decision-making. However, weak communication links and limited bandwidth at the edge introduce latency and potential data inconsistencies. Optimizing communication protocols and introducing decentralized coordination mechanisms can mitigate these issues.

Security and privacy considerations further complicate agent deployment. Many FM-based agents process sensitive user data, necessitating robust privacy-preserving mechanisms, such as encrypted inference. Additionally, the deployment of agents on battery-powered edge devices raises concerns about energy consumption. Energy-efficient inference techniques, such as model quantization and selective activation of model components, are essential for prolonging device longevity.

Finally, serving architectures must be reimagined to support agent-specific execution patterns. Unlike traditional FM inference, agents operate with diverse planning strategies, memory retrieval mechanisms, and execution styles. Designing adaptable, scalable serving systems capable of supporting varied agent behaviors is imperative for the next generation of intelligent applications. Addressing these challenges will pave the way for more efficient, responsive, and capable AI agents in heterogeneous edge-cloud environments.

D. Discussion

1) Delving Into the Heterogeneity: Resource Heterogeneity. Edge computing systems exhibit fundamental execution heterogeneity across three interdependent dimensions: computational architectures, memory access patterns, and communication constraints. At the hardware level, processing units diverge in their core capabilities: GPUs excel at massive parallel computation but face memory bandwidth limitations; FPGAs enable custom linear algebra acceleration but are inefficient for attention mechanisms with non-linear operations. This architectural diversity demands algorithm-architecture co-design, such as approximating attention mechanisms on FPGAs using matrix multiplication primitives.

Memory heterogeneity manifests through model-driven access patterns. Different LLMs have diverse dimensions and require adaptive memory management of model and KV cache, such as CPU-GPU collaborative inference for MoE models, where CPUs manage less-activated experts while GPUs compute others and get dense outputs, reducing memory overhead. Communication heterogeneity arises from both physical-layer variability and model-specific semantic activation patterns. To address this issue, methods such as sparsified activation can be utilized by allocating semantic-important activations to reliable physical links for fast and reliable serving LLMs.

Model & Agent Heterogeneity. In terms of model architecture, although most LLMs now use a decoder-only transformer architecture, there are still different architectures like BERT's encoder-only transformer, T5's encoder-decoder transformer, and Gemini's MoE architecture. Additionally, models coexist with parameter counts ranging from billions (e.g., LLaMA-13B) to trillions (e.g., GPT-4). Training methods are also diverse: GPT-3 relies on self-supervised learning, InstructGPT is optimized through reinforcement learning from human feedback, and DeepMind's Gopher incorporates curriculum learning strategies. In terms of training data, the Bloom model covers 46 languages to showcase multilingual advantages, while Baidu's ERNIE-Bot focuses on optimizing the Chinese context; BioBERT targets the biomedical field, and Meta's Galactica specializes in scientific literature understanding. Fine-tuning strategies are also selective: Stanford's Alpaca uses full parameter fine-tuning, Google's FLAN-T5 employs adapter modules, and QLoRA achieves efficient fine-tuning through quantization techniques.

Agent Heterogeneity refers to the diverse characteristics and behaviors exhibited by agents based on LLMs. The existence of heterogeneous agents allows for a more comprehensive approach to solve problem. For instance, in urban planning, an agent with knowledge of architecture can work together with an agent proficient in environmental science and another with expertise in transportation planning. Each agent contributes its unique perspective and capabilities, leading to more well-rounded and effective solutions. However, agents may have different internal representations, communication protocols, and task-specific terminologies, developing a universal communication framework that enables effective information exchange is difficult. Determining which agents can work together effectively and ensuring their compatibility is not straightforward. The integration of agents from different sources or with different design philosophies may lead to conflicts in functionality or resource usage. Training and cooperating heterogeneous agents requires a complex approach.

Application Heterogeneity. At the application level, heterogeneity also exists, encompassing query load, user demands, and user inputs. The query load varies over time, requiring the system to efficiently handle bursty loads while avoiding excessive hardware resource consumption. User demands are diverse; some applications prioritize low latency, while others require high accuracy, necessitating a more adaptive system response. Therefore, an elastic agent serving system should dynamically adjust resources, apply different parallelization strategies, and leverage various acceleration techniques based on system and application characteristics, as detailed in Sections V and IV. Additionally, user inputs vary in length, modality, and task type, requiring the system to adaptively batch requests and invoke different agents or models.

2) Optimizing Costs for Agent Applications: We elaborate on cost-reduction techniques from both a system framework and an algorithm-system co-design perspective. System Framework Optimization:

- Hierarchical computing and storage scheduling: A multi-tier computing and storage scheduling strategy can significantly optimize resource utilization. For instance, GPU-CPU collaborative inference can be implemented by assigning different computational components to appropriate hardware. Specifically, computationally expensive components like feed-forward networks can be offloaded to GPUs, while attention mechanisms, which involve sequential dependencies and memory-intensive operations, can be processed on CPUs [285].
- Elastic resource allocation: Dynamically allocating GPU resources based on system load and user demand can improve cost efficiency. By employing adaptive scheduling strategies, the system can allocate resources only when necessary, reducing idle GPU usage and optimizing inference efficiency under varying workloads.
- Efficient parallelization for multi-node execution: Optimizing multi-GPU and multi-node parallel execution can further reduce computational overhead. Strategies such as pipeline parallelism and tensor parallelism can enhance throughput while maintaining computational efficiency.

Algorithm-System Co-Design for Cost Optimization:

- Optimized model architectures: Leveraging more efficient model architectures can significantly reduce computational costs. Techniques such as Mixture of Experts (MoE), Multi-Head Latent Attention, and sparse activation neurons help distribute computation more efficiently while maintaining model accuracy.
- Precision reduction and quantization: Mixed-precision computing, such as using FP8, can reduce memory consumption and computational cost.
- KV cache reuse and token pruning: By reusing key-value caches across multiple inference steps, redundant computations can be avoided, leading to faster inference times. Additionally, token pruning techniques can dynamically remove tokens that contribute minimally to the final output, further reducing unnecessary computation.

These strategies are described in this survey and collectively contribute to reducing the operational cost of LLM-based agents while maintaining service quality.

VIII. Conclusion

FM-powered agent services are widely recognized as a crucial way of achieving AGI and are expected to spearhead development in this field over the next decade. Constructing a robust infrastructure for these agent services within an edge-cloud environment is of great importance and has a substantial impact on the user experience. In this survey, we

have presented a unified framework for a thorough literature review on diverse techniques for deploying FM-powered agent services. We have introduced a series of low-level optimization methods for the model execution, including computation, communication, and I/O optimization. Subsequently, we have discussed the parallelism methods and resource allocation schemes designed to optimize resource utilization. We've also highlighted several popular FMs and introduced two lightweight methods, namely model compression and token reduction, to expedite the inference processes. Furthermore, we have also reviewed the latest research endeavors focusing on the development of an effective multi-agent framework and explored several recent applications. Finally, the future research directions for serving multi-modal agents on heterogeneous devices are outlined.

This survey is anticipated to significantly advance the development of large-scale model applications in both academia and industry. The proposed framework, along with the referenced research works, provides a comprehensive perspective on deploying FM-powered agent services. It showcases the latest advancements and encourages more researchers to contribute to this practical and compelling field. The technologies discussed in the paper collectively contribute to the development of a reliable and flexible serving system, enabling low-latency agent services that enhance users' daily experiences with increased intelligence. The integrated optimization of system architecture and AI algorithms will expedite the deployment of large-scale model applications for a diverse range of users, thereby promoting societal progress. Looking ahead, we are dedicated to advancing research in large model applications and aim to expand our investigation to include a broader range of models, execution environments, and practical applications.

REFERENCES

[1] R. Bommasani et al. "On the opportunities and risks of foundation models." 2022. [Online]. Available: https://arxiv.org/abs/2108.07258

[2] Z. Xi et al., "The rise and potential of large language model based agents: A survey," *Sci. China Inf. Sci.*, vol. 68, no. 2, 2025, Art. no. 121101.

[3] Nerdynav. "107 up-to-date ChatGPT statistics & user numbers." 2024. Accessed: Apr. 24, 2024. [Online]. Available: https://nerdynav.com/chatgpt-statistics/

[4] C. Kachris, "A survey on hardware accelerators for large language models," *Appl. Sci.*, vol. 15, no. 2, p. 586, 2025.

[5] S. Tang et al., "A survey on scheduling techniques in computing and network convergence," *IEEE Commun. Surveys Tuts.*, vol. 26, no. 1, pp. 160–195, 1st Quart., 2024.

[6] X. Miao et al. "Towards efficient generative large language model serving: A survey from algorithms to systems." 2023. [Online]. Available: https://arxiv.org/abs/2312.15234

[7] M. Xu et al., "Unleashing the power of edge-cloud generative AI in mobile networks: A survey of AIGC services," *IEEE Commun. Surveys Tuts.*, vol. 26, no. 2, pp. 1127–1170, 2nd Quart., 2024.

[8] H. Djigal, J. Xu, L. Liu, and Y. Zhang, "Machine and deep learning for resource allocation in multi-access edge computing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 4, pp. 2449–2494, 4th Quart., 2022.

[9] Y. Chang et al., "A survey on evaluation of large language models," *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 3, pp. 1–45, Mar. 2024. [Online]. Available: https://doi.org/10.1145/3641289

[10] S. Yin et al., "A survey on multimodal large language models," *Nat. Sci. Rev.*, vol. 11, no. 12, Nov. 2024, Art. no. nwae403. [Online]. Available: https://doi.org/10.1093/nsr/nwae403

[11] X. Zhu, J. Li, Y. Liu, C. Ma, and W. Wang, "A survey on model compression for large language models," *Trans. Assoc. Comput. Linguist.*, vol. 12, pp. 1556–1577, Aug. 2024. [Online]. Available: https://aclanthology.org/2024.tacl-1.85/

[12] Y. Xu, C. Song, Y. Sun, and S. Yu, "Advances and challenges in large model compression: A survey," in *Proc. Guangdong–Hong Kong–Macao Greater Bay Area Int. Conf. Digit. Econ. Artif. Intell. (DEAI)*, 2024, pp. 421–426. [Online]. Available: https://doi.org/10.1145/3675417.3675487

[13] X. Xu et al., "A survey on knowledge distillation of large language models," 2024, *arXiv:2402.13116*.

[14] L. Wang et al., "A survey on large language model based autonomous agents," *Front. Comput. Sci.*, vol. 18, no. 6, pp. 1–26, 2024.

[15] T. Guo et al., "Large language model based multi-agents: A survey of progress and challenges," in *Proc. 33rd Int. Joint Conf. Artif. Intell. (IJCAI)*, Aug. 2024, pp. 8048–8057. [Online]. Available: https://doi.org/10.24963/ijcai.2024/890

[16] OpenAI et al. "OpenAI o1 system card." 2024. [Online]. Available: https://arxiv.org/abs/2412.16720

[17] D. Guo et al., "DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning," 2025, *arXiv:2501.12948*.

[18] "OpenAI ChatGPT." Accessed: May 12, 2024. [Online]. Available: https://chat.openai.com

[19] Baidu. "Wenxin Yiyan." 2024. [Online]. Available: https://yiyan.baidu.com/

[20] "Github copilot: Your AI pair programmer." 2023. [Online]. Available: https://github.com/features/copilot

[21] OpenAI. "OpenAI codex." 2021. [Online]. Available: https://openai.com/index/openai-codex/

[22] "Midjourney." 2023. [Online]. Available: https://www.midjourney.com/

[23] OpenAI. "Sora." 2024. [Online]. Available: https://openai.com/index/sora/

[24] Runway. "Gen-2 by runway." 2023. [Online]. Available: https://research.runwayml.com/gen2

[25] T. B. Richards et al. "Auto-GPT." 2023. [Online]. Available: https://news.agpt.co/.

[26] J. S. Park, J. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," in *Proc. 36th Annu. ACM Symp. User Interface Softw. Technol.*, 2023, pp. 1–22.

[27] G. Wang et al., "Voyager: An open-ended embodied agent with large language models," *Trans. Mach. Learn. Res.*, pp. 1–44, Mar. 2023.

[28] C. Qian et al., "Communicative agents for software development," 2023, *arXiv:2307.07924*.

[29] M. R. Morris et al., "Position: Levels of AGI for operationalizing progress on the path to AGI," in *Proc. 41st Int. Conf. Mach. Learn.*, 2024, pp. 1–9.

[30] L.-b. Ning et al., "CheatAgent: Attacking LLM-empowered recommender systems via LLM agent," in *Proc. 30th ACM SIGKDD Conf. Knowl. Disc. Data Min. (KDD)*, 2024, pp. 2284–2295. [Online]. Available: https://doi.org/10.1145/3637528.3671837

[31] W. Tao, Y. Zhou, Y. Wang, W. Zhang, H. Zhang, and Y. Cheng, "MAGIS: LLM-based multi-agent framework for GitHub issue resolution," in *Proc. Adv. Neural Inf. Process. Syst.*, 2024, pp. 51963–51993. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/file/5d1f02132ef51602adf07000ca5b6138-Paper-Conference.pdf

[32] D. Yang et al., "How2Comm: Communication-efficient and collaboration-pragmatic multi-agent perception," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, pp. 25151–25164. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/4f31327e046913c7238d5b671f5d820e-Paper-Conference.pdf

[33] A. Smit, N. Grinsztajn, P. Duckworth, T. D. Barrett, and A. Pretorius, "Should we be going MAD? A look at multi-agent debate strategies for LLMs," in *Proc. 41st Int. Conf. Mach. Learn. (ICML)*, 2024, pp. 1–8.

[34] A. Salutari, *Harmonizing Users' and System's Requirements in Complex and Resource Intensive Application Domains by a Distributed Hybrid Approach*, Università degli Studi dell'Aquila, L'Aquila, Italy, 2024.

[35] S. Agashe, Y. Fan, and X. E. Wang, "Evaluating multi-agent coordination abilities in large language models," 2023, *arXiv:2310.03903*.

[36] W. Huang et al., "Grounded decoding: Guiding text generation with grounded models for embodied agents," in *Proc. Conf. Neural Inf. Process. Syst.*, 2023, pp. 59636–59661.

[37] A. Omidvar and A. An, "Empowering conversational agents using semantic in-context learning," in *Proc. Annu. Meeting Assoc. Comput. Linguist.*, 2023, pp. 1–8.

[38] Y. Talebirad and A. Nadiri, "Multi-agent collaboration: Harnessing the power of intelligent LLM agents," 2023, *arXiv:2306.03314*.

[39] X. Yao et al., "Socialized learning: Making each other better through multi-agent collaboration," in *Proc. 41st Int. Conf. Mach. Learn.*, 2024, pp. 1–12.

[40] O. Friha, M. A. Ferrag, B. Kantarci, B. Cakmak, A. Ozgun, and N. Ghoualmi-Zine, "LLM-based edge intelligence: A comprehensive survey on architectures, applications, security and trustworthiness," *IEEE Open J. Commun. Soc.*, vol. 5, pp. 5799–5856, 2024.

[41] Z. Liu, Y. Zhang, P. Li, Y. Liu, and D. Yang, "Dynamic LLM-agent network: An LLM-agent collaboration framework with agent team optimization," 2023, *arXiv:2310.02170*.

[42] W. Chen et al., "AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors," in *Proc. 12th Int. Conf. Learn. Rep.*, 2024, pp. 1–9. [Online]. Available: https://openreview.net/forum?id=EHg5GDnyq1

[43] X. Liu et al., "AgentBench: Evaluating LLMs as agents," in *Proc. 12th Int. Conf. Learn. Rep.*, 2024, p. 58. [Online]. Available: https://openreview.net/forum?id=zAdUB0aCTQ

[44] H. Zhang et al., "Building cooperative embodied agents modularly with large language models," in *Proc. 12th Int. Conf. Learn. Rep.*, 2024, pp. 1–9. [Online]. Available: https://openreview.net/forum?id=EnXJfQqy0K

[45] K. Li and Y. Zhang, "Planning first, question second: An LLM-guided method for controllable question generation," in *Proc. Assoc. Comput. Linguist. (ACL)*, Aug. 2024, pp. 4715–4729. [Online]. Available: https://aclanthology.org/2024.findings-acl.280/

[46] Z. Yang, G. Chen, X. Li, W. Wang, and Y. Yang, "DoraemonGPT: Toward understanding dynamic scenes with large language models (exemplified as a video agent)," in *Proc. 41st Int. Conf. Mach. Learn. (ICML)*, 2024, pp. 1–9.

[47] R. Gong et al., "Mindagent: Emergent gaming interaction," in *Proc. Findings Assoc. Comput. Linguist. (NAACL)*, Jun. 2024, pp. 3154–3183. [Online]. Available: https://aclanthology.org/2024.findings-naacl.200/

[48] Z. Wang, S. Cai, G. Chen, A. Liu, X. S. Ma, and Y. Liang, "Describe, explain, plan and select: Interactive planning with LLMs enables open-world multi-task agents," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 1–9.

[49] Z. Zhao, W. S. Lee, and D. Hsu, "Large language models as commonsense knowledge for large-scale task planning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 31967–31987.

[50] L. Guan, K. Valmeekam, S. Sreedharan, and S. Kambhampati, "Leveraging pre-trained large language models to construct and utilize world models for model-based task planning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 79081–79094.

[51] S. Wang et al., "Avalon's game of thoughts: Battle against deception through recursive contemplation," 2023, *arXiv:2310.01320*.

[52] G. Lee, V. Hartmann, J. Park, D. Papailiopoulos, and K. Lee, "Prompted LLMs as Chatbot modules for long open-domain conversation," in *Proc. Annu. Meeting Assoc. Comput. Linguist.*, 2023, pp. 4536–4554.

[53] Y. Wang et al., "MEMORYLLM: Towards self-updatable large language models," in *Proc. 41st Int. Conf. Mach. Learn.*, Jul. 2024, pp. 50453–50466. [Online]. Available: https://proceedings.mlr.press/v235/wang24s.html

[54] D. Zhang, L. Chen, S. Zhang, H. Xu, Z. Zhao, and K. Yu, "Large language models are semi-parametric reinforcement learning agents," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 1–9.

[55] X. Li and X. Qiu, "MOT: Memory-of-thought enables ChatGPT to self-improve," in *Proc. Conf. Empirical Methods Nat. Lang. Process.*, 2023, pp. 6354–6374.

[56] N. Liu, L. Chen, X. Tian, W. Zou, K. Chen, and M. Cui, "From LLM to conversational agent: A memory enhanced architecture with fine-tuning of large language models," 2024, *arXiv:2401.02777*.

[57] S. Xu, W. Hua, and Y. Zhang, "OpenP5: An open-source platform for developing, training, and evaluating LLM-based recommender systems," in *Proc. 47th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval (SIGIR)*, 2024, pp. 386–394. [Online]. Available: https://doi.org/10.1145/3626772.3657883

[58] S. Zhang et al., "Offline training of language model agents with functions as learnable weights," in *Proc. 41st Int. Conf. Mach. Learn.*, vol. 235, Jul. 2024, pp. 60315–60335. [Online]. Available: https://proceedings.mlr.press/v235/zhang24cd.html

[59] T. Schick et al., "Toolformer: Language models can teach themselves to use tools," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 1–14.

[60] R. Yang et al., "Gpt4tools: Teaching large language model to use tools via self-instruction," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 1–4.

[61] Y. Kong et al., "TPTU-V2: Boosting task planning and tool usage of large language model-based agents in real-world industry systems," in *Proc. Conf. Empirical Methods Nat. Lang. Process. Ind. Track*, Nov. 2024, pp. 371–385. [Online]. Available: https://aclanthology.org/2024.emnlp-industry.27/

[62] H. Wei, C. Liu, and Y. Shi, "A robust distributed MPC framework for multi-agent consensus with communication delays," *IEEE Trans. Autom. Control*, vol. 69, no. 11, pp. 7418–7432, Nov. 2024.

[63] J. Liu, T. Yang, and J. Neville, "CliqueParcel: An approach for batching LLM prompts that jointly optimizes efficiency and faithfulness," 2024, *arXiv:2402.14833*.

[64] K. Basu et al., "NestFul: A benchmark for evaluating LLMs on nested sequences of API calls," 2024, *arXiv:2409.03797*.

[65] Y. Zhang, H. Cai, X. Song, Y. Chen, R. Sun, and J. Zheng, "Reverse chain: A generic-rule for LLMs to master multi-API planning," in *Proc. Findings Assoc. Comput. Linguist. (NAACL)*, Jun. 2024, pp. 302–325. [Online]. Available: https://aclanthology.org/2024.findings-naacl.22/

[66] G. Ramírez, M. Lindemann, A. Birch, and I. Titov, "Cache & distil: Optimizing API calls to large language models," 2023, *arXiv:2310.13561*.

[67] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–9.

[68] C. Raffel et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.

[69] A. Radford et al., "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[70] T. Brown et al., "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1877–1901.

[71] W. Zeng et al. "PanGu-α: Large-scale autoregressive pretrained Chinese language models with auto-parallel computation." 2021. [Online]. Available: https://arxiv.org/abs/2104.12369

[72] Y. Sun et al., "ERNIE 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation," 2021, *arXiv:2107.02137*.

[73] J. Hoffmann et al., "Training compute-optimal large language models," 2022, *arXiv:2203.15556*.

[74] R. Thoppilan et al., "Lamda: Language models for dialog applications," 2022, *arXiv:2201.08239*.

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

[76] J. W. Rae et al., "Scaling language models: Methods, analysis & insights from training gopher," 2021, *arXiv:2112.11446*.

[77] S. Zhang et al., "OPT: Open pre-trained transformer language models," 2022, *arXiv:2205.01068*.

[78] R. Taylor et al., "Galactica: A large language model for science," 2022, *arXiv:2211.09085*.

[79] T. L. Scao et al., "BLOOM: A 176B-parameter open-access multilingual language model," 2022, *arXiv:2211.05100*.

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

[81] H. Touvron et al., "Llama 2: Open foundation and fine-tuned chat models," 2023, *arXiv:2307.09288*.

[82] A. Grattafiori et al. "The llama 3 herd of models." 2024. [Online]. Available: https://arxiv.org/abs/2407.21783

[83] BaiChuan-Inc. "Baichuan-7B: About: A large-scale 7B pretraining language model developed by BaiChuan-Inc." 2024. Accessed: Apr. 7, 2024. [Online]. Available: https://github.com/baichuan-inc/Baichuan-7B/tree/main

[84] Baichuan Intelligent Technology. "A 13B large language model developed by Baichuan intelligent technology." 2024. Accessed: Apr. 7, 2024. [Online]. Available: https://github.com/baichuan-inc/Baichuan-13B/tree/main

[85] J. Bai et al., "Qwen technical report," 2023, *arXiv:2309.16609*.

[86] C. An et al. "Training-free long-context scaling of large language models." 2024. [Online]. Available: https://arxiv.org/abs/2402.17463

[87] A. Yang et al. "Qwen2 technical report." 2024. [Online]. Available: https://arxiv.org/abs/2407.10671

[88] Qwen et al. "Qwen2.5 technical report." 2025. [Online]. Available: https://arxiv.org/abs/2412.15115

[89] T. Wei et al., "Skywork: A more open bilingual foundation model," 2023, *arXiv:2310.19341*.
[90] E. Almazrouei et al., "The falcon series of open language models," 2023, *arXiv:2311.16867*.
[91] R. Li et al., "Starcoder: May the source be with you!" 2023, *arXiv:2305.06161*.
[92] J. Wei et al. "Finetuned language models are zero-shot learners." 2022. [Online]. Available: https://arxiv.org/abs/2109.01652
[93] L. Xue et al., "mT5: A massively multilingual pre-trained text-to-text transformer,' 2020, *arXiv:2010.11934*.
[94] S. Shen et al. "Flan-MOE: Scaling instruction-finetuned language models with sparse mixture of experts." 2023. [Online]. Available: https://arxiv.org/abs/2210.11416
[95] H. W. Chung et al. "Scaling instruction-finetuned language models." 2022. [Online]. Available: https://arxiv.org/abs/2210.11416
[96] R. Taori et al. "Alpaca." 2023. [Online]. Available: https://crfm.stanford.edu/2023/03/13/alpaca.html
[97] A. Kumar et al. "Training language models to self-correct via reinforcement learning." 2024. [Online]. Available: https://arxiv.org/abs/2409.12917
[98] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 24824–24837. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf
[99] DeepSeek-AI et al. "DeepSeek-V3 technical report." 2024. [Online]. Available: https://arxiv.org/abs/2412.19437
[100] Z. Shao et al., "Deepseekmath: Pushing the limits of mathematical reasoning in open language models," 2024, *arXiv:2402.03300*.
[101] Y. Wang et al., "MMLU-pro: A more robust and challenging multi-task language understanding benchmark," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 37, 2025, pp. 95266–95290.
[102] D. Rein et al., "GPQA: A graduate-level Google-proof Q&A benchmark," 2023, *arXiv:2311.12022*.
[103] N. Chowdhury et al. "Introducing SWE-bench verified." 2025. Accessed: Apr. 28, 2025. [Online]. Available: https://openai.com/index/introducing-swe-bench-verified/
[104] Q. Team. "QwQ-32B: Embracing the power of reinforcement learning." Mar. 2025. [Online]. Available: https://qwenlm.github.io/blog/qwq-32b/
[105] Y. Cheng et al. "Exploring large language model based intelligent agents: Definitions, methods, and prospects." 2024. [Online]. Available: https://arxiv.org/abs/2401.03428
[106] J.-B. Alayrac et al., "Flamingo: A visual language model for few-shot learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 23716–23736.
[107] J. Achiam et al., "GPT-4 technical report," 2023, *arXiv:2303.08774*.
[108] D. Zhu, J. Chen, X. Shen, X. Li, and M. Elhoseiny, "MiniGPT-4: Enhancing vision-language understanding with advanced large language models," 2023, *arXiv:2304.10592*.
[109] W. Wang et al., "CogVLM: Visual expert for pretrained language models," 2023, *arXiv:2311.03079*.
[110] R. Girdhar et al., "ImageBind: One embedding space to bind them all," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 15180–15190.
[111] Y. Su, T. Lan, H. Li, J. Xu, Y. Wang, and D. Cai, "PandaGPT: One model to instruction-follow them all," 2023, *arXiv:2305.16355*.
[112] S. Wu, H. Fei, L. Qu, W. Ji, and T.-S. Chua, "Next-GPT: Any-to-any multimodal LLM," 2023, *arXiv:2309.05519*.
[113] J. Han et al., "OneLLM: One framework to align all modalities with language," 2023, *arXiv:2312.03700*.
[114] G. Team et al., "Gemini: A family of highly capable multimodal models," 2023, *arXiv:2312.11805*.
[115] M. Reid et al., "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," 2024, *arXiv:2403.05530*.
[116] S. Bai et al. "Qwen2.5-VL technical report." 2025. [Online]. Available: https://arxiv.org/abs/2502.13923
[117] K. Chen et al., "Shikra: Unleashing multimodal LLM's referential dialogue magic," 2023, *arXiv:2306.15195*.
[118] H. You et al., "Ferret: Refer and ground anything anywhere at any granularity," 2023, *arXiv:2310.07704*.
[119] T. Chen, S. Saxena, L. Li, D. J. Fleet, and G. Hinton. "Pix2seq: A language modeling framework for object detection." 2022. [Online]. Available: https://arxiv.org/abs/2109.10852
[120] A. Zhang et al., "Next-chat: An LMM for chat, detection and segmentation," 2023, *arXiv:2311.04498*.
[121] Meta AI. "Llama-4 multimodal intelligence." 2025. Accessed: Apr. 228, 2025. [Online]. Available: https://ai.meta.com/blog/llama-4-multimodal-intelligence/
[122] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter, "A simple and effective pruning approach for large language models," 2023, *arXiv:2306.11695*.
[123] 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, 2024, pp. 1–9.
[124] M. Zhang et al. "LoRAPrune: Structured pruning meets low-rank parameter-efficient fine-tuning." 2024. [Online]. Available: https://arxiv.org/abs/2305.18403
[125] T. Chen, T. Ding, B. Yadav, I. Zharkov, and L. Liang, "LoRaShear: Efficient large language model structured pruning and knowledge recovery," 2023, *arXiv:2310.18356*.
[126] Y. An, X. Zhao, T. Yu, M. Tang, and J. Wang, "Fluctuation-based adaptive structured pruning for large language models," 2023, *arXiv:2312.11983*.
[127] H. Shao, B. Liu, and Y. Qian, "One-shot sensitivity-aware mixed sparsity pruning for large language models," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2024, pp. 11296–11300.
[128] S. Guo, J. Xu, L. L. Zhang, and M. Yang, "Compresso: Structured pruning with collaborative prompting learns compact large language models," 2023, *arXiv:2310.05015*.
[129] M. Xia, T. Gao, Z. Zeng, and D. Chen, "Sheared llama: Accelerating language model pre-training via structured pruning," 2023, *arXiv:2310.06694*.
[130] Y. Ji, Y. Cao, and J. Liu, "Pruning large language models via accuracy predictor," 2023, *arXiv:2309.09507*.
[131] R. J. Das, L. Ma, and Z. Shen, "Beyond size: How gradients shape pruning decisions in large language models," 2023, *arXiv:2311.04902*.
[132] E. Kurtić, E. Frantar, and D. Alistarh, "ZipLM: Inference-aware structured pruning of language models," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 1–7.
[133] W. Ye, Q. Wu, W. Lin, and Y. Zhou, "Fit and prune: Fast and training-free visual token pruning for multi-modal large language models," in *Proc. AAAI Conf. Artif. Intell.*, vol. 39, 2025, pp. 22128–22136.
[134] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "SmoothQuant: Accurate and efficient post-training quantization for large language models," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 38087–38099.
[135] Z. Yuan et al., "RPTQ: Reorder-based post-training quantization for large language models," 2023, *arXiv:2304.01089*.
[136] Y. Li et al., "LoFTQ: LoRa-fine-tuning-aware quantization for large language models," 2023, *arXiv:2310.08659*.
[137] X. Wei et al., "Outlier suppression+: Accurate quantization of large language models by equivalent and effective shifting and scaling," in *Proc. Conf. Empirical Methods Nat. Lang. Process.*, 2023, pp. 1648–1665.
[138] Q. Li et al., "FPTQ: Fine-grained post-training quantization for large language models," 2023, *arXiv:2308.15987*.
[139] C. Lee, J. Jin, T. Kim, H. Kim, and E. Park, "OWQ: Lessons learned from activation outliers for weight quantization in large language models," 2023, *arXiv:2306.02272*.
[140] J. Lin et al., "AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration," in *Proc. Mach. Learn. Syst.*, vol. 6, 2024, pp. 87–100.
[141] Y. Zhang et al., "Integer or floating point? New outlooks for low-bit quantization on large language models," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, 2024, pp. 1–6.
[142] W. Shao et al., "OmniQuant: Omnidirectionally calibrated quantization for large language models," 2023, *arXiv:2308.13137*.
[143] R. Liu et al., "IntactKV: Improving large language model Quantization by keeping pivot tokens intact," 2024, *arXiv:2403.01241*.
[144] J. Kim et al., "Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2023, pp. 36187–36207.
[145] J. Liu, R. Gong, X. Wei, Z. Dong, J. Cai, and B. Zhuang, "QLLM: Accurate and efficient low-bitwidth quantization for large language models," 2023, *arXiv:2310.08041*.
[146] X. Hu et al., "OstQuant: Refining large language model Quantization with orthogonal and scaling transformations for better distribution fitting," 2025, *arXiv:2501.13987*.
[147] C.-Y. Hsieh et al., "Distilling step-by-step! Outperforming larger language models with less training data and smaller model sizes," 2023, *arXiv:2305.02301*.
[148] L. Tunstall et al., "ZePHYR: Direct distillation of LM alignment," 2023, *arXiv:2310.16944*.

[149] Y. Jiang, C. Chan, M. Chen, and W. Wang, "LION: Adversarial distillation of closed-source large language model," 2023, *arXiv:2305.12870*.

[150] X. Zhu, B. Qi, K. Zhang, X. Long, and B. Zhou, "PAD: Program-aided distillation specializes large models in reasoning," 2023, *arXiv:2305.13888*.

[151] Y. Zhou et al., "DistillSpec: Improving speculative decoding via knowledge distillation," 2023, *arXiv:2310.08461*.

[152] C. Liang, S. Zuo, Q. Zhang, P. He, W. Chen, and T. Zhao, "Less is more: Task-aware layer-wise distillation for language model compression," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 20852–20867.

[153] C. Liang, H. Jiang, Z. Li, X. Tang, B. Yin, and T. Zhao, "HomoDistil: Homotopic task-agnostic distillation of pre-trained transformers," 2023, *arXiv:2302.09632*.

[154] L. H. Li, J. Hessel, Y. Yu, X. Ren, K.-W. Chang, and Y. Choi, "Symbolic chain-of-thought distillation: Small models can also 'think' step-by-step," 2023, *arXiv:2306.14050*.

[155] C. Liu et al., "Evolving knowledge distillation with large language models and active learning," 2024, *arXiv:2403.06414*.

[156] C. Zhang et al., "Minimal distillation schedule for extreme language model compression," in *Proc. Findings Assoc. Comput. Linguist. (EACL)*, 2024, pp. 1378–1394.

[157] V. Kontonis, F. Iliopoulos, K. Trinh, C. Baykal, G. Menghani, and E. Vee, "SLAM: Student-label mixing for distillation with unlabeled examples," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 1–28.

[158] W. Zhou, S. Zhang, Y. Gu, M. Chen, and H. Poon, "Universalner: Targeted distillation from large language models for open named entity recognition," 2023, *arXiv:2308.03279*.

[159] P. Wang, Z. Wang, Z. Li, Y. Gao, B. Yin, and X. Ren, "Scott: Self-consistent chain-of-thought distillation," 2023, *arXiv:2305.01879*.

[160] D. Li, Z. Tan, T. Chen, and H. Liu, "Contextualization distillation from large language model for knowledge graph completion," 2024, *arXiv:2402.01729*.

[161] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "{*INFaaS*}: Automated model-less inference serving," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2021, pp. 397–411.

[162] K. Zhao et al., "EdgeAdaptor: Online configuration adaption, model selection and resource provisioning for edge DNN inference serving at scale," *IEEE Trans. Mobile Comput.*, vol. 22, no. 10, pp. 5870–5886, Oct. 2023.

[163] Y. Wu, M. Lentz, D. Zhuo, and Y. Lu, "Serving and optimizing machine learning workflows on heterogeneous infrastructures," *Proc. VLDB Endow.*, vol. 16, no. 3, pp. 406–419, 2022.

[164] L. Guo, W. Choe, and F. X. Lin, "STI: Turbocharge NLP inference at the edge via elastic pipelining," in *Proc. 28th ACM Int. Conf. Architect. Support Program. Lang. Oper. Syst.*, 2023, pp. 791–803.

[165] Y. Wang, K. Chen, H. Tan, and K. Guo, "TABI: An efficient multi-level inference system for large language models," in *Proc. 18th Eur. Conf. Comput. Syst.*, 2023, pp. 233–248.

[166] T. Schuster, A. Fisch, T. Jaakkola, and R. Barzilay, "Consistent accelerated inference via confident adaptive transformers," in *Proc. Conf. Empirical Methods Nat. Lang. Process.*, 2021, pp. 4962–4979.

[167] T. R. D. Oliveira et al., "Virtual reality solutions employing artificial intelligence methods: A systematic literature review," *ACM Comput. Surveys*, vol. 55, no. 10, pp. 1–29, 2023.

[168] P. Esmaeilzadeh, "Use of AI-based tools for healthcare purposes: A survey study from consumers' perspectives," *BMC Med. Inform. Decis. Making*, vol. 20, pp. 1–19, Jul. 2020.

[169] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," in *Proc. 40th Int. Conf. Mach. Learn. (ICML)*, 2023, pp. 1–8.

[170] D. Xu et al., "LLMCAD: Fast and scalable on-device large language model inference," 2023, *arXiv:2309.04255*.

[171] X. Miao et al., "SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification," in *Proc. 29th ACM Int. Conf. Architect. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2024, pp. 932–949. [Online]. Available: https://doi.org/10.1145/3620666.3651335

[172] Z. Chen et al., "Sequoia: Scalable and robust speculative decoding," in *Proc. 38th Annu. Conf. Neural Inf. Process. Syst.*, 2024, pp. 1–8. [Online]. Available: https://openreview.net/forum?id=rk2L9YGDi2

[173] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Comput. Surveys*, vol. 55, no. 6, pp. 1–28, Dec. 2022.

[174] J. B. Haurum, S. Escalera, G. W. Taylor, and T. B. Moeslund, "Which tokens to use? Investigating token reduction in vision transformers," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2023, pp. 773–783.

[175] D. Bolya, C.-Y. Fu, X. Dai, P. Zhang, C. Feichtenhofer, and J. Hoffman, "Token merging: Your ViT but faster," in *Proc. 11th Int. Conf. Learn. Rep.*, 2022, pp. 1–2.

[176] Y. Li, B. Dong, F. Guerin, and C. Lin, "Compressing context to enhance inference efficiency of large language models," in *Proc. Conf. Empirical Methods Nat. Lang. Process.*, 2023, pp. 6342–6353.

[177] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu, "LLMLingua: Compressing prompts for accelerated inference of large language models," in *Proc. Conf. Empirical Methods Nat. Lang. Process.*, 2023, pp. 13358–13376.

[178] Z. Zhang et al., "H_2O: Heavy-hitter oracle for efficient generative inference of large language models," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 1–8.

[179] S. Ge, Y. Zhang, L. Liu, M. Zhang, J. Han, and J. Gao, "Model tells you what to discard: Adaptive KV cache compression for LLMs," in *Proc. 12th Int. Conf. Learn. Rep.*, 2023, pp. 1–14.

[180] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," in *Proc. 12th Int. Conf. Learn. Rep.*, 2023, pp. 1–21.

[181] Y. Xu et al., "ThinK: Thinner key cache by query-driven pruning," in *Proc. 13th Int. Conf. Learn. Rep.*, 2025, pp. 1–5. [Online]. Available: https://openreview.net/forum?id=n0OtGl6VGb

[182] G. Xiao et al., "DuoAttention: Efficient long-context LLM inference with retrieval and streaming heads," in *Proc. 13th Int. Conf. Learn. Rep.*, 2025, pp. 1–6. [Online]. Available: https://openreview.net/forum?id=cFu7ze7xUm

[183] Z. Chen et al., "MagicPIG: LSH sampling for efficient LLM generation," in *Proc. 13th Int. Conf. Learn. Rep.*, 2025, p. 5. [Online]. Available: https://openreview.net/forum?id=ALzTQUgW8a

[184] A. Chevalier, A. Wettig, A. Ajith, and D. Chen, "Adapting language models to compress contexts," in *Proc. Conf. Empirical Methods Nat. Lang. Process.*, 2023, pp. 3829–3846.

[185] T. Ge, H. Jing, L. Wang, X. Wang, S.-Q. Chen, and F. Wei, "In-context autoencoder for context compression in a large language model," in *Proc. 12th Int. Conf. Learn. Rep.*, 2023, pp. 1–9.

[186] J. Mu, X. Li, and N. Goodman, "Learning to compress prompts with gist tokens," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 19327–19352.

[187] P. Dong et al., "HeatVIT: Hardware-efficient adaptive token pruning for vision transformers," in *Proc. IEEE Int. Symp. High Perform. Comput. Architect. (HPCA)*, 2023, pp. 442–455.

[188] Y. Zhang, L. Wei, and N. Freris, "Synergistic patch pruning for vision transformer: Unifying intra-& inter-layer patch importance," in *Proc. 12th Int. Conf. Learn. Rep.*, 2024, pp. 1–9.

[189] Y. Liang, G. Chongjian, Z. Tong, Y. Song, J. Wang, and P. Xie, "EViT: Expediting vision transformers via token reorganizations," in *Proc. Int. Conf. Learn. Rep.*, 2021, p. 9.

[190] S. Long, Z. Zhao, J. Pi, S. Wang, and J. Wang, "Beyond attentive tokens: Incorporating token importance and diversity for efficient vision transformers," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2023, pp. 10334–10343.

[191] S. Wei, T. Ye, S. Zhang, Y. Tang, and J. Liang, "Joint token pruning and squeezing towards more aggressive compression of vision transformers," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 2092–2101.

[192] L. Chen et al., "An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models," in *Proc. 18th Eur. Conf. Comput. Vis. (ECCV)*, 2024, pp. 19–35.

[193] Z. Wan et al., "LOOK-M: Look-once optimization in KV cache for efficient multimodal long-context inference," in *Proc. Findings Assoc. Comput. Linguist. (EMNLP)*, Nov. 2024, pp. 4065–4078. [Online]. Available: https://aclanthology.org/2024.findings-emnlp.235/

[194] J. Chen et al., "OTAS: An elastic transformer serving system via token adaptation," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2024, pp. 1021–1030.

[195] F. Xue, V. Likhosherstov, A. Arnab, N. Houlsby, M. Dehghani, and Y. You, "Adaptive computation with elastic input sequence," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 38971–38988.

[196] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 613–627.

[197] C. Zhang, M. Yu, W. Wang, and F. Yan, "*MArk*: Exploiting cloud services for cost-effective, SLO-aware machine learning inference serving," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2019, pp. 1049–1062.

[198] H. Shen et al., "Nexus: A GPU cluster engine for accelerating DNN-based video analysis," in *Proc. 27th ACM Symp. Oper. Syst. Principles*, 2019, pp. 322–337.

[199] D. Crankshaw et al., "InferLine: Latency-aware provisioning and scaling for prediction serving pipelines," in *Proc. 11th ACM Symp. Cloud Comput.*, 2020, pp. 477–491.

[200] A. Gujarati et al., "Serving *DNNs* like clockwork: Performance predictability from the bottom up," in *Proc. 14th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2020, pp. 443–462.

[201] J. R. Gunasekaran et al., "CockTail: A multidimensional optimization for model serving in cloud," in *Proc. 19th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2022, pp. 1041–1057.

[202] B. Li, S. Samsi, V. Gadepally, and D. Tiwari, "Kairos: Building cost-efficient machine learning inference systems with heterogeneous cloud resources," in *Proc. 32nd Int. Symp. High Perform. Parallel Distrib. Comput.*, 2023, pp. 3–16.

[203] H. Zhang, Y. Tang, A. Khandelwal, and I. Stoica, "*SHEPHERD*: Serving *DNNs* in the wild," in *Proc. 20th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2023, pp. 787–808.

[204] X. Miao et al., "SpotServe: Serving generative large language models on preemptible instances," in *Proc. 29th ACM Int. Conf. Architect. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2024, pp. 1112–1127. [Online]. Available: https://doi.org/10.1145/3620665.3640411

[205] Z. Tong, X. Deng, F. Ye, S. Basodi, X. Xiao, and Y. Pan, "Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment," *Inf. Sci.*, vol. 537, pp. 116–131, Oct. 2020.

[206] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in IoT edge computing," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1133–1146, Jun. 2020.

[207] Z. Zhou, S. Yu, W. Chen, and X. Chen, "CE-IoT: Cost-effective cloud-edge resource provisioning for heterogeneous IoT applications," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8600–8614, Sep. 2020.

[208] Z. Chang, L. Liu, X. Guo, and Q. Sheng, "Dynamic resource allocation and computation offloading for IoT fog computing system," *IEEE Trans. Ind. Informat.*, vol. 17, no. 5, pp. 3348–3357, May 2021.

[209] Y. He, J. Fang, F. R. Yu, and V. C. Leung, "Large language models (LLMs) inference offloading and resource allocation in cloud-edge computing: An active inference approach," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 11253–11264, Dec. 2024.

[210] M. Xu, D. Niyato, and C. G. Brinton, "Serving long-context LLMs at the mobile edge: Test-time reinforcement learning-based model caching and inference offloading," 2025, *arXiv:2501.14205*.

[211] Y. Fu et al., "ServerlessLLM: Low-latency serverless inference for large language models," in *Proc. 18th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2024, pp. 135–153.

[212] S. Ye et al., "Galaxy: A resource-efficient collaborative edge AI system for in-situ transformer inference," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, 2024, pp. 1001–1010.

[213] B. Wu, S. Liu, Y. Zhong, P. Sun, X. Liu, and X. Jin, "LoongServe: Efficiently serving long-context large language models with elastic sequence parallelism," in *Proc. ACM SIGOPS 30th Symp. Oper. Syst. Principle (SOSP)*, 2024, pp. 640–654.

[214] NVIDIA Corporation. "NVIDIA triton inference server." Accessed: Apr. 17, 2024. [Online]. Available: https://developer.nvidia.com/nvidia-triton-inference-server

[215] Google LLC. "TensorFlow serving." Accessed: Apr. 17, 2024. [Online]. Available: https://www.tensorflow.org/tfx/guide/serving

[216] 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*.

[217] R. Y. Aminabadi et al., "Deepspeed-inference: Enabling efficient inference of transformer models at unprecedented scale," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal. (SC)*, 2022, pp. 1–15.

[218] D. Li et al., "LightSeq:: Sequence level parallelism for distributed training of long context transformers," in *Proc. Workshop Adv. Neural Netw. Training Comput. Efficiency Scalability Resource Optim. (WANT@NeurIPS)*, 2023, pp. 1–8.

[219] A. Borzunov et al., "Distributed inference and fine-tuning of large language models over the Internet," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 12312–12331.

[220] A. Agrawal et al., "Taming throughput-latency tradeoff in LLM inference with Sarathi-serve," in *Proc. 18th USENIX Conf. Oper. Syst. Design Implement. (OSDI)*, 2024, pp. 1–9.

[221] Y. Zhong et al., "DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving," in *Proc. 18th USENIX Conf. Oper. Syst. Design Implement. (OSDI)*, 2024, pp. 193–210.

[222] H. Wang, P. Zheng, X. Han, W. Xu, R. Li, and T. Zhang, "FedNLR: Federated learning with neuron-wise learning rates," in *Proc. 30th ACM SIGKDD Conf. Knowl. Disc. Data Min.*, 2024, pp. 3069–3080.

[223] Y. Jiang, R. Yan, X. Yao, Y. Zhou, B. Chen, and B. Yuan, "HEXGEN: Generative inference of large language model over heterogeneous environment," in *Proc. 41st Int. Conf. Mach. Learn. (ICML)*, 2024, pp. 1–6.

[224] Y. Mei, Y. Zhuang, X. Miao, J. Yang, Z. Jia, and R. Vinayak, "Helix: Distributed serving of large language models via max-flow on heterogeneous GPUs," in *Proc. ASPLOS Conf.*, 2025, pp. 1–15.

[225] H. Wang et al., "FedDSE: Distribution-aware sub-model extraction for federated learning over resource-constrained devices," in *Proc. ACM Web Conf.*, 2024, pp. 2902–2913.

[226] S. Lu, M. Wang, S. Liang, J. Lin, and Z. Wang, "Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer," in *Proc. IEEE 33rd Int. Syst. Chip Conf. (SOCC)*, 2020, pp. 84–89.

[227] H. Jang, J. Kim, J.-E. Jo, J. Lee, and J. Kim, "MNNFast: A fast and scalable system architecture for memory-augmented neural networks," in *Proc. 46th Int. Symp. Comput. Architect.*, 2019, pp. 250–263.

[228] Y. Bai, H. Zhou, K. Zhao, J. Chen, J. Yu, and K. Wang, "Transformer-OPU: An FPGA-based overlay processor for transformer networks," in *Proc. IEEE 31st Annu. Int. Symp. Field Program. Custom Comput. Mach. (FCCM)*, 2023, pp. 221–221.

[229] I. Okubo, K. Sugiura, and H. Matsutani, "A cost-efficient FPGA implementation of tiny transformer model using neural ODE," 2024, *arXiv:2401.02721*.

[230] S. Zeng et al., "FlightLLM: Efficient large language model inference with a complete mapping flow on FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2024, pp. 223–234.

[231] T. J. Ham et al., "A$\hat{3}$: Accelerating attention mechanisms in neural networks with approximation," in *Proc. IEEE Int. Symp. High Perform. Comput. Architect. (HPCA)*, 2020, pp. 328–341.

[232] T. J. Ham et al., "ELSA: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Architect. (ISCA)*, 2021, pp. 692–705.

[233] H. Wang, Z. Zhang, and S. Han, "SpatTen: Efficient sparse attention architecture with cascade token and head pruning," in *Proc. IEEE Int. Symp. High-Perform. Comput. Architect. (HPCA)*, 2021, pp. 97–110.

[234] L. Lu et al., "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchitect. (MICRO)*, 2021, pp. 977–991.

[235] Z. Zhou, J. Liu, Z. Gu, and G. Sun, "Energon: Toward efficient acceleration of transformers using dynamic sparse attention," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 1, pp. 136–149, Jan. 2023.

[236] H. Guo, L. Peng, J. Zhang, Q. Chen, and T. D. LeCompte, "ATT: A fault-tolerant ReRAM accelerator for attention-based neural networks," in *Proc. IEEE 38th Int. Conf. Comput. Design (ICCD)*, 2020, pp. 213–221.

[237] A. F. Laguna, A. Kazemi, M. Niemier, and X. S. Hu, "In-memory computing based accelerator for transformer networks for long sequences," in *Proc. IEEE Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2021, pp. 1839–1844.

[238] B. Reidy, M. Mohammadi, M. Elbtity, H. Smith, and Z. Ramtin, "Work in progress: Real-time transformer inference on edge AI accelerators," in *Proc. IEEE 29th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, 2023, pp. 341–344.

[239] B. He and T. Hofmann, "Simplifying transformer blocks," in *Proc. 12th Int. Conf. Learn. Rep.*, 2024, pp. 1–29. [Online]. Available: https://openreview.net/forum?id=RtDok9eS3s

[240] J. Choi, H. Li, B. Kim, S. Hwang, and J. H. Ahn, "Accelerating transformer networks through recomposing softmax layers," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, 2022, pp. 92–103.

[241] N. Yang et al., "Inference with reference: Lossless acceleration of large language models," 2023, *arXiv:2304.04487*.

[242] P. Belcak and R. Wattenhofer. "Exponentially faster language modeling.' 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2311.10770

[243] H. Shen, H. Chang, B. Dong, Y. Luo, and H. Meng. "Efficient LLM inference on CPUs." 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2311.00502

[244] Y. Song, Z. Mi, H. Xie, and H. Chen, "PowerInfer: Fast large language model serving with a consumer-grade GPU," in *Proc. ACM SIGOPS 30th Symp. Oper. Syst. Principle (SOSP)*, 2024, pp. 590–606. [Online]. Available: https://doi.org/10.1145/3694715.3695964

[245] X. Zhao, B. Jia, H. Zhou, Z. Liu, S. Cheng, and Y. You, "HeteGen: Efficient heterogeneous parallel inference for large language models on resource-constrained devices," in *Proc. Mach. Learn. Syst.*, 2024, pp. 162–172. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2024/file/5431dca75a8d2abc1fb51e89e8324f10-Paper-Conference.pdf

[246] Y. Sheng et al., "FlexGEN: High-throughput generative inference of large language models with a single GPU," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 31094–31116.

[247] Z. Liu et al., "Deja Vu: Contextual sparsity for efficient LLMs at inference time," in *Proc. 40th Int. Conf. Mach. Learn. (ICML)*, 2023, pp. 1–8.

[248] K. Alizadeh et al., "LLM in a flash: Efficient large language model inference with limited memory," in *Proc. 62nd Annu. Meeting Assoc. Comput. Linguist.*, 2024, pp. 12562–12584.

[249] N. Shazeer, "Fast transformer decoding: One write-head is all you need," 2019, *arXiv:1911.02150*.

[250] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebron, and S. Sanghai, "GQA: Training generalized multi-query transformer models from multi-head checkpoints," in *Proc. Conf. Empirical Methods Nat. Lang. Process.*, 2023, pp. 4895–4901.

[251] W. Kwon et al., "Efficient memory management for large language model serving with pagedattention," in *Proc. 29th Symp. Oper. Syst. Principle*, 2023, pp. 611–626.

[252] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "FLASHATTENTION: Fast and memory-efficient exact attention with IO-awareness," in *Proc. 36th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 2022, pp. 1–8.

[253] K. Hong et al., "FlashDecoding++: Faster large language model inference with asynchronization, flat GEMM optimization, and heuristics," in *Proc. Mach. Learn. Syst.*, vol. 6, 2024, pp. 148–161. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2024/file/5321b1dabcd2be188d796c21b733e8c7-Paper-Conference.pdf

[254] P. Patel et al., "SplitWise: Efficient generative LLM inference using phase splitting," in *Proc. ACM/IEEE 51st Annu. Int. Symp. Comput. Architect. (ISCA)*, 2024, pp. 118–132.

[255] B. Wu, Y. Zhong, Z. Zhang, G. Huang, X. Liu, and X. Jin, "Fast distributed inference serving for large language models," 2023, *arXiv:2305.05920*.

[256] S. Rajbhandari et al., "DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 18332–18346.

[257] R. Yi, L. Guo, S. Wei, A. Zhou, S. Wang, and M. Xu, "EdgeMOE: Fast on-device inference of MOE-based large language models," 2023, *arXiv:2308.14352*.

[258] A. Eliseev and D. Mazur, "Fast inference of mixture-of-experts language models with offloading," 2023, *arXiv:2312.17238*.

[259] L. Xue, Y. Fu, Z. Lu, L. Mai, and M. Marina, "MoE-infinity: Activation-aware expert offloading for efficient MoE serving," 2024, *arXiv:2401.14361*.

[260] K. Kamahori, Y. Gu, K. Zhu, and B. Kasikci, "Fiddler: CPU-GPU orchestration for fast inference of mixture-of-experts models," in *Proc. 13th Int. Conf. Learn. Rep.*, 2025, p. 6.

[261] G. Gerganov. "Ggerganov/llama.cpp: Port of Facebook's llama model in C/C++." 2023. [Online]. Available: https://github.com/ggerganov/llama.cpp

[262] B. Tadych. "Distributed llama." 2024. [Online]. Available: https://github.com/b4rtaz/distributed-llama

[263] Z. Li, W. Feng, M. Guizani, and H. Yu, "TPI-LLM: Serving 70B-scale LLMs efficiently on low-resource edge devices," 2024, *arXiv:2410.00531*.

[264] Z. Hong et al., "Intelligence-endogenous management platform for computing and network convergence," *IEEE Netw.*, vol. 38, no. 4, pp. 166–173, Jul. 2024.

[265] C. Liu and J. Zhao, "Resource allocation in large language model integrated 6G vehicular networks," in *Proc. IEEE 99th Veh. Technol. Conf. (VTC-Spring)*, 2024, pp. 1–6.

[266] M. Zhang, X. Shen, J. Cao, Z. Cui, and S. Jiang, "Edgeshard: Efficient LLM inference via collaborative edge computing," *IEEE Internet Things J.*, vol. 12, no. 10, pp. 13119–13131, May 2025.

[267] Y. Ding, C. Niu, F. Wu, S. Tang, C. Lyu, and G. Chen, "Enhancing on-device LLM inference with historical cloud-based LLM interactions," in *Proc. 30th ACM SIGKDD Conf. Knowl. Disc. Data Min.*, 2024, pp. 597–608.

[268] J. Zhao, Y. Song, S. Liu, I. G. Harris, and S. Abdu Jyothi, "LinguaLinked: Distributed large language model inference on mobile devices," in *Proc. 62nd Annu. Meeting Assoc. Comput. Linguist.*, Aug. 2024, pp. 160–171. [Online]. Available: https://aclanthology.org/2024.acl-demos.16/

[269] X. Mu, Y. Liu, L. Guo, and N. Al-Dhahir, "Heterogeneous semantic and bit communications: A semi-NOMA scheme," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 1, pp. 155–169, Jan. 2023.

[270] Z. Qin, J. Ying, D. Yang, H. Wang, and X. Tao, "Computing networks enabled semantic communications," *IEEE Netw.*, vol. 38, no. 2, pp. 122–131, Mar. 2024.

[271] H. Dong, T. Johnson, M. Cho, and E. Soroush, "Towards low-bit communication for tensor parallel LLM inference," in *Proc. NeurIPS Workshop*, 2024, p. 5. [Online]. Available: https://arxiv.org/abs/2411.07942

[272] V. Ramesh and K. Li, "Communicating activations between language model agents," 2025, *arXiv:2501.14082*.

[273] MLC Team. "MLC-LLM: Efficient and portable deployment of large language models." 2023. Accessed: Feb. 28, 2025. [Online]. Available: https://github.com/mlc-ai/mlc-llm

[274] MNN LLM, "MNN-LLM: LLM deploy project based MNN." 2023. [Online]. Available: https://github.com/wangzhaode/mnn-llm

[275] L. Zheng et al., "Judging LLM-as-a-judge with MT-bench and Chatbot arena," in *Proc. 37th Conf. Neural Inf. Process. Syst. Datasets Benchmarks Track*, 2023, p. 14. [Online]. Available: https://openreview.net/forum?id=uccHPGDlao

[276] Y. Gorbachev, M. Fedorov, I. Slavutin, A. Tugarev, M. Fatekhov, and Y. Tarkan, "OpenVINO deep learning workbench: Comprehensive analysis and tuning of neural networks inference," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshops*, 2019, pp. 783–787.

[277] MLLM. "MLLM is a fast and lightweight multimodal LLM inference engine for mobile and edge devices." 2025. [Online]. Available: https://github.com/UbiquitousLearning/mllm, 2023.

[278] H. Xia et al. "FP6-LLM: Efficiently serving large language models through FP6-centric algorithm-system co-design." 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2401.14112

[279] S. Li et al., "Colossal-AI: A unified deep learning system for large-scale parallel training," in *Proc. 52nd Int. Conf. Parallel Process.*, 2023, pp. 766–775.

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

[281] Tensorrtllm. "A TensorRT toolbox for Optimized large language model inference." 2023. [Online]. Available: https://github.com/NVIDIA/TensorRT-LLM

[282] Harrison Chase. "LangChain." 2024. Accessed: Apr. 7, 2024. [Online]. Available: https://github.com/langchain-ai/langchain

[283] L. Zheng et al., "SGLang: Efficient execution of structured language model programs," in *Proc. 38th Annu. Conf. Neural Inf. Process. Syst.*, 2024, pp. 1–8. [Online]. Available: https://openreview.net/forum?id=VqkAKQibpq

[284] Z. Wang et al., "Speculative RAG: Enhancing retrieval augmented generation through drafting," in *Proc. 13th Int. Conf. Learn. Rep.*, 2025, pp. 1–8. [Online]. Available: https://openreview.net/forum?id=xgQfWbV6Ey

[285] J. He and J. Zhai, "FastDecode: High-throughput GPU-efficient LLM serving using heterogeneous pipelines," 2024, *arXiv:2403.11421*.

Wenchao Xu received the B.E. and M.E. degrees from Zhejiang University, Hangzhou, China, in 2008 and 2011, respectively, and the Ph.D. degree from the University of Waterloo, Canada, in 2018. He is an Assistant Professor with the Division of Integrative Systems and Design, Hong Kong University of Science and Technology, Hong Kong. In 2011, he joined Alcatel Lucent Shanghai Bell Company Ltd., where he was a Software Engineer for telecom virtualization. He has also been an Assistant Professor with the School of Computing and Information Sciences, Caritas Institute of Higher Education, Hong Kong. His research interests includes wireless communication, Internet of Things, distributed computing, and AI enabled networking.

Jinyu Chen received the Bachelor of Engineering degree in software engineering from Xidian University, Xi'an, China, in 2019, and the Master of Science degree in computer science and technology from Sun Yat-sen University, Guangzhou, China, in 2022. He is currently pursuing the Ph.D. degree with the Department of Computing, The Hong Kong Polytechnic University.

His research interests encompass cloud and edge computing, multimodal learning, and AI systems.

Peirong Zheng received the B.S. degree from the University of Electronic Science and Technology of China in 2022, and the M.S. degree from Nanyang Technological University in 2023. He is currently pursuing the Ph.D. degree with the Department of Computing, The Hong Kong Polytechnic University. His research interests focus on machine learning system, large language model, and edge/cloud computing.

Xiaoquan Yi received the B.E. degree from the School of Computer Science and Technology, Wuhan University of Technology. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, Huazhong University of Science and Technology. His main research direction is large model generated content detection.

Tianyi Tian is currently pursuing the B.E. degree in electronic information engineering with the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications. His research interests include natural language processing and affective computing.

Wenhui Zhu is currently pursuing the B.E. degree in communication engineering with the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications. Her research interests include wireless communication and foundation models.

Quan Wan is currently pursuing the B.E. degree in software engineering with the School of Computer Science, Beijing University of Posts and Telecommunications, China. His research interests include distributed systems, and edge–cloud collaboration.

Haozhao Wang (Member, IEEE) received the B.S. degree in computer science from the University of Electronic Science and Technology in 2016, and the Ph.D. degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, where he is currently an Assistant Professor and also he was a Postdoctoral Researcher. He was also a Research Fellow with SLab, Nanyang Technical University. His research interests include distributed machine learning and multimodal learning.

Yunfeng Fan received the B.S. degree from the Beijing Institute of Technology, Beijing, China, in 2019, and the M.S. degree from the School of Automation, Beijing Institute of Technology in 2022. He is currently pursuing the Ph.D. degree with the Department of Computing, The Hong Kong Polytechnic University. His current research interests include multimodal learning and federated learning.

Qinliang Su received the Ph.D. degree from the University of Hong Kong in 2014, followed by a Postdoctoral Training with the ECE department, Duke University, USA, from 2015 to 2017. He is an Associate Professor with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. His research interests cover the general areas of machine learning and its applications, including generative models, weakly-supervised learning, representation learning, statistical inference, NLP, and anomaly detection. His work on semantic hashing won the Best Paper Mention Award on the Prestigious Natural Language Processing Conference ACL 2018. He has published over 70 papers on leading machine learning and AI conferences and journals like ICML, NeurIPS, AAAI, IJCAI, ACL, WWW, and TKDE, and is constantly serve as the Area Chairs or the Senior Program Committee on top-tier conferences.

Xuemin (Sherman) Shen (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990. He is currently a University Professor with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research interests include network resource management, wireless network security, the Internet of Things, 5G and beyond, and vehicular ad hoc and sensor networks. He was a recipient of the Canadian Award for Telecommunications Research from the Canadian Society of Information Theory in 2021; the R. A. Fessenden Award from IEEE, Canada, in 2019; the Award of Merit from the Federation of Chinese Canadian Professionals (Ontario) in 2019; the James Evans Avant Garde Award from the IEEE Vehicular Technology Society in 2018; the Joseph LoCicero Award and Education Award from the IEEE Communications Society in 2015 and 2017, respectively; and the Technical Recognition Award from Wireless Communications Technical Committee in 2019 and the AHSN Technical Committee in 2013. He was also a recipient of the Excellent Graduate Supervision Award from the University of Waterloo in 2006; and the Premier's Research Excellence Award from the Province of Ontario, Canada, in 2003. He was the Technical Program Committee Chair/Co-Chair of IEEE Globecom 2016, IEEE Infocom 2014, IEEE VTC 2010 Fall, and IEEE Globecom 2007; and the Chair of IEEE Communications Society Technical Committee on Wireless Communications. He is the President of the IEEE Communications Society. He was the Vice President of Technical and Educational Activities, the Vice President of Publications, the Member-at-Large on the Board of Governors, the Chair of the Distinguished Lecturer Selection Committee, and a member of IEEE Fellow Selection Committee of the ComSoc. He was an Editor-in-Chief of IEEE INTERNET OF THINGS JOURNAL, IEEE NETWORK, and *IET Communications*. He is a Registered Professional Engineer of Ontario, Canada; a Fellow of the Engineering Institute of Canada, Canadian Academy of Engineering, and the Royal Society of Canada; a Foreign Member of Chinese Academy of Engineering; and a Distinguished Lecturer of IEEE Vehicular Technology Society and Communications Society.