



## Review article

## Empowering large language models to edge intelligence: A survey of edge efficient LLMs and techniques



Rui Wang \*, Zhiyong Gao, Liuyang Zhang, Shuaibing Yue, Ziyi Gao

*School of Computer and Communication Engineering, University of Science and Technology Beijing (USTB), Beijing, 100083, China*

## ARTICLE INFO

## Keywords:

Large language model  
Edge intelligence  
Small language model  
Model compression  
Efficient inference  
On-device LLM

## ABSTRACT

Large language models (LLMs) have showcased exceptional capabilities across various natural language processing (NLP) tasks in recent years, such as machine translation, text summarization, and question answering. Despite their impressive performance, the deployment of these models on edge devices, such as mobile phones, IoT devices, and edge computing nodes, is significantly hindered by their substantial computational and memory requirements. This survey provides a comprehensive overview of the state-of-the-art techniques and strategies for enabling efficient inference of LLMs on edge devices. We explore approaches including the development of small language models (SLMs), model compression techniques, inference optimization strategies, and dedicated frameworks for edge deployment. Our goal is to highlight the advancements and ongoing challenges in this field, offering valuable insights for researchers and practitioners striving to bring the power of LLMs to edge environments.

## Contents

1. Introduction .....	2
2. Small language model .....	3
2.1. Differences and similarities with large models .....	3
2.2. Off-the-shelf small models .....	3
2.2.1. Small variants of large models .....	4
2.2.2. Pre-trained small models .....	5
2.2.3. Compressed small models .....	7
2.3. Perspectives on edge performance .....	8
2.4. Prospects and applications .....	8
2.4.1. SLMs for specialized domains .....	8
2.4.2. Application in multimodal models .....	9
2.4.3. AI agents for edge .....	9
2.4.4. Mobile assistants .....	9
3. Model compression .....	9
3.1. Pruning .....	9
3.2. Knowledge distillation .....	10
3.3. Quantization .....	10
3.4. Low-rank decomposition .....	12
3.5. Discussion .....	12
4. Inference optimization .....	12
4.1. Speculative decoding .....	12
4.2. KV cache compression .....	13
4.3. Early exiting .....	14
4.4. Kernel optimization .....	15
4.5. Memory offloading .....	15
4.6. Discussion .....	15

\* Corresponding author.

E-mail address: [wangrui@ustb.edu.cn](mailto:wangrui@ustb.edu.cn) (R. Wang).

5. Deployment .....	15
5.1. On-device inference engines .....	16
5.2. Cloud-edge collaborative frameworks .....	16
5.3. Deployment suites .....	17
6. Open challenges and future directions .....	17
6.1. Resource constraints on edge devices .....	18
6.2. Dynamicity of edge environments .....	18
6.3. Heterogeneity of edge devices .....	18
6.4. Edge-cloud collaboration .....	18
6.5. Privacy and security concerns .....	18
6.6. Hallucination issues .....	19
6.7. Morality and law .....	19
6.8. Interpretability .....	19
6.9. User preferences and personalization .....	19
6.10. Green and sustainable development .....	19
7. Conclusion .....	19
Declaration of competing interest .....	19
Data availability .....	20
References .....	20

## 1. Introduction

Large language models (LLMs) have shown remarkable effectiveness in a wide range of natural language processing (NLP) tasks, such as machine translation, text summarization, and question answering. These models are typically trained on large corpora of text data, enabling them to produce coherent and contextually relevant text. Over recent years, the capabilities and sizes of these models have grown significantly, as seen in Fig. 1, which illustrates the increase in parameter sizes of various models from early versions like GPT-1 [1] and BERT [2] to more recent advancements like GPT-4 [3], BLOOM [4], LLaMA [5] and Llama2 [6].

However, the large parameter sizes of LLMs present significant challenges for their deployment on edge devices. Edge devices, including mobile phones, IoT devices, and edge computing nodes, typically possess limited computational and memory resources. For instance, Llama2-7B [6] inference requires at least 7 GB of CPU or GPU memory with INT4 quantization and only achieves 4.5 tokens per second on NVIDIA Jetson AGX Orin [7], which is often beyond the capabilities of most edge devices. Moreover, continuous inference can cause devices to heat up significantly, as some edge or mobile devices are typically passively cooled, which may have a significant impact on performance [8]. This high computational requirement and substantial memory footprint hinder the widespread adoption of LLMs in edge environments, where resources are constrained.

Specifically, the edge deployment of LLMs faces four major challenges due to the resource constraints inherent in edge environments:

1. The rapid growth in the size of LLMs is at odds with the limited memory resources of edge devices.
2. The high computational demands of LLMs clash with the restricted computational resources of edge devices.
3. The substantial energy consumption of LLMs conflicts with the finite energy supply of edge devices.
4. The large throughput requirements of LLMs are in contrast with the limited bandwidth of edge devices.

To address these challenges, existing research has aimed to alleviate resource pressures through: (1) introducing lightweight architectures to reduce model computational complexity and communication overhead; (2) utilizing model compression techniques to decrease the scale of model parameters; (3) optimizing inference system efficiency by designing effective inference strategies and algorithms.

Despite these challenges, deploying LLMs on edge devices presents distinct advantages in latency, privacy, personalization and so on [9]. Edge deployment can significantly reduce latency, as data processing

occurs closer to the source, thereby improving real-time responsiveness. It can also enhance data privacy and security, as sensitive information does not need to be transmitted to centralized servers for processing. Additionally, edge deployment can lead to more efficient use of network bandwidth and provide uninterrupted services even in areas with limited connectivity. Furthermore, edge-based LLMs can offer more personalized experiences by leveraging local data to tailor responses and services to individual users, thereby enhancing user satisfaction and engagement. According to Statista's forecast, the number of globally connected Internet of Things (IoT) devices is expected to reach 15.9 billion by 2023 and is estimated to escalate to 39.6 billion by 2033 [10]. The rapid expansion of edge IoT devices necessitates the exploration of redundant computational power at the edge and the utilization of edge advantages to provide services.

Existing surveys [11–16] have systematically summarized efficient inference methods for large language models. However, their focus predominantly lies in cloud environments, and there is a notable lack of investigation and discussion on models, technologies, and frameworks suitable for edge computing. In contrast, our survey offers a perspective on the development of LLMs at the edge, addressing the challenges of limited resources for LLMs in edge environments. It reviews recent developments in LLMs from three aspects: small-sized models, compression techniques and inference optimization technologies for edge. Furthermore, it identifies future research challenges in this field. Xu et al. [17] provides a comprehensive review of on-device LLMs, but our survey not only includes on-device scenarios but also considers relevant research progress in cloud-edge collaboration. Additionally, two recent related works, Lu et al. [18] and Wang et al. [19], have comprehensively surveyed recent advancements in small models and limited model optimization techniques. Our survey not only provides the latest research progress on small models but also covers a more comprehensive study on the optimization and deployment of small models at the edge.

The remainder of this survey is organized as follows: Section 2 provides a comprehensive summary and detailed investigation of recent notable small language models. Section 3 presents the latest advancements in LLM compression techniques. Section 4 delves into research progress on LLM inference optimization technologies for edge computing. Section 5 explores frameworks suitable for deploying LLMs on edge devices. Section 6 highlights the challenges associated with LLMs on the edge and discusses the future. Finally, we conclude the survey in Section 7. Table 1 lists the acronyms used throughout this survey. We hope this survey will serve as a valuable resource for researchers and practitioners working to bring the power of LLMs to edge environments.

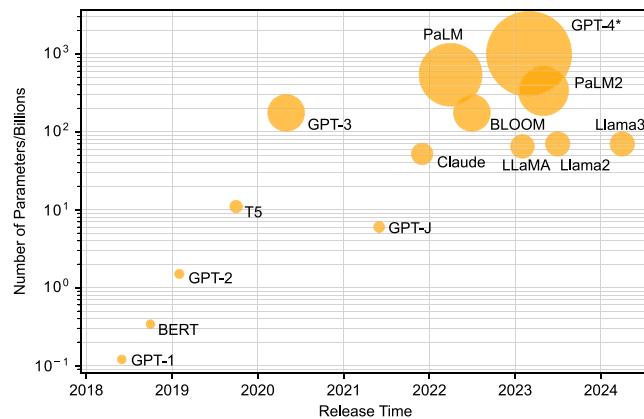


Fig. 1. Increase in parameter size of LLMs over time. GPT-4 is estimated to have more than 1 trillion parameters.

Table 1  
Abbreviation list.

Abbreviation	Full Form
LLM	Large Language Model
SLM	Small Language Model
NLP	Natural Language Processing
FFN	Feed Forward Network
MHA	Multi-Head Attention
MQA	Multi-Query Attention
GQA	Grouped-Query Attention
MLA	Multi-head Latent Attention
IoT	Internet of Things
KLD	Kullback–Leibler Divergence
MLLM	Multimodal Large Language Model
RAG	Retrieval-Augmented Generation
CoT	Chains-of-Thought
ICL	Incremental Context Learning
IF	Instruction Following
PTQ	Post-Training Quantization
QAT	Quantization-Aware Training
SIMD	Single Instruction Multiple Data
KV Cache	Key-Value Cache
SRAM	Static Random Access Memory
DRAM	Dynamic Random Access Memory
GEMM	General Matrix Multiplication
GEMV	General Matrix-Vector Multiplication
SpMM	Sparse Matrix Multiplication
FL	Federated Learning
HFL	Hierarchical Federated Learning
BSBODP	Bridge Sample Based Online Distillation Protocol
ASIC	Application-Specific Integrated Circuit
PEFT	Parameter-Efficient Fine-Tuning
LoRA	Low-Rank Adaptation

## 2. Small language model

The edge availability of large language models is essential for a wide range of applications. However, deploying LLMs on edge devices is challenging due to the high computational requirements of these models. Therefore, small language models (SLMs) have been developed in recent years to address this issue, as shown in Fig. 2. SLMs are typically smaller in size and have fewer parameters than their larger counterparts, making them suitable for edge devices with limited computational resources. In this survey, we define SLMs as language models with fewer than 4 billion parameters, which is a common threshold for distinguishing between large and small language models, and focus on transformer-based architectures.

We collected information on models submitted to Open LLM Leaderboard [20] and analyzed the distribution of model parameters, as shown in Fig. 3. The histogram reveals a significant boundary around 4-6B parameters, indicating that research on small models primarily

focuses on models with fewer than 4B parameters. Therefore, in this survey, we define small models as those with fewer than 4B parameters.

### 2.1. Differences and similarities with large models

Modern large language models are typically based on the transformer architecture, which has achieved significant success in the field of natural language processing. The SLMs discussed in this survey typically utilize a transformer architecture similar to LLMs, but differ in parameter size, training dataset, and training methodology. Additionally, to achieve efficient inference, some SLMs adopt special optimization strategies different from those of LLMs. Due to their smaller parameter sizes, SLMs may exhibit reduced capabilities in logical reasoning, long-context understanding, and multilingual processing compared to larger LLMs, which excel in these areas.

- Parameter Size.** Parameter size is the primary criterion for distinguishing between SLMs and LLMs in this survey. The smaller parameter size of SLMs is mainly due to reduced model layers, hidden dimensions, attention heads, and intermediate dimensions, resulting in faster inference speeds and lower memory consumption compared to larger models.
- Training Datasets.** SLMs typically require much smaller training datasets than LLMs, which may result in lower performance on some tasks compared to LLMs. Additionally, many SLMs are trained using synthetic data generated from larger models and high-quality data curated and filtered carefully, as inspired by the research of Phi-1 [21].
- Training Algorithms.** While most foundational LLMs are trained from scratch, some SLMs follow this approach as well. However, there are differences in hyperparameter settings, training data preferences at different stages, and other aspects. For example, MiniCPM [22] conducted detailed experiments to select the best hyperparameters. Some SLMs opt to distill or prune existing LLMs to obtain better performance than training from scratch.
- Model Architecture.** To achieve efficient inference, many SLMs employ efficient attention mechanisms such as GQA [23] and MQA [24], which reduce time and space complexity compared to MHA [25]. Another common technique in SLMs is parameter sharing, including embedding weight tying [26] and transformer weight sharing [27]. By reducing the model's parameter size, these techniques enhance both memory and computational efficiency. We also note that some SLMs prefer deeper and thinner architectures in smaller models, rather than reducing the number of layers. This viewpoint was proposed and verified by MobileLLM [27] and has been adopted by many other SLMs.
- Application Tasks.** Research on emergent abilities [28] indicates that certain capabilities are only present in LLMs with larger parameter sizes, particularly in tasks requiring complex reasoning or logic, where larger models often have an advantage. However, SLMs can also perform well in specific tasks, such as summarization, instruction following, rewriting, and code generation. Therefore, smaller models are more suitable for certain specific tasks and domain-specific tasks without complex reasoning requirements.

### 2.2. Off-the-shelf small models

SLMs can be categorized based on their development approaches into small variants of LLMs, pre-trained SLMs, and compressed SLMs. Small variants of LLMs refer to models that are smaller in size within a series, typically sharing the same architecture, corpus, and training methodology as their larger counterparts. Pre-trained SLMs are designed specifically as smaller LLMs, developed from scratch with architectural and training optimizations to further reduce model resource requirements and enhance performance. Compressed SLMs are

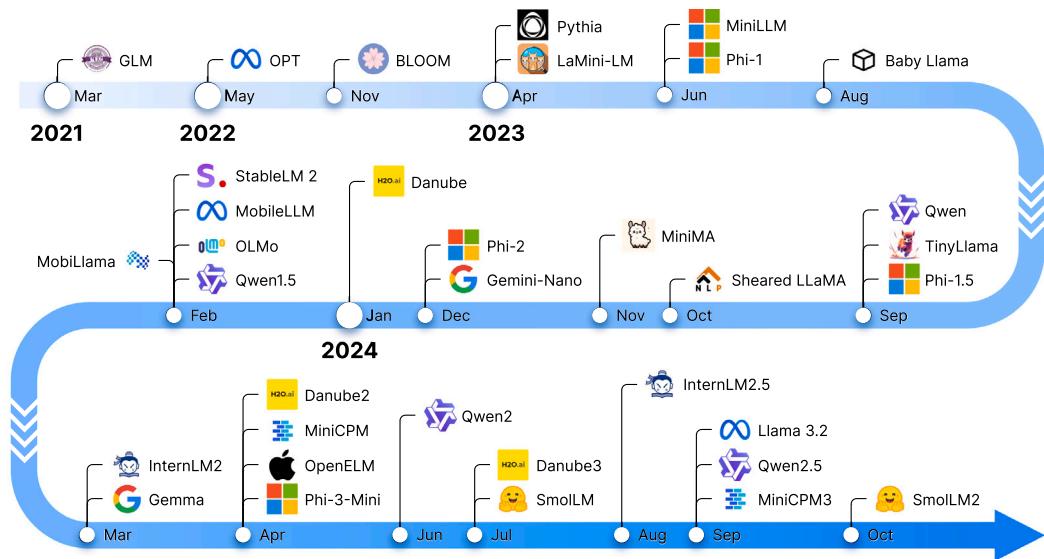


Fig. 2. Timeline of small language models from 2022 to 2024.

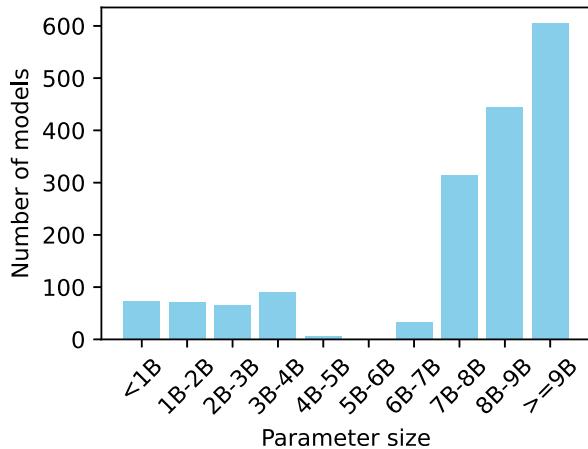


Fig. 3. Histogram of model parameters from Open LLM Leaderboard [20].

derived from larger LLMs through the process of pruning or knowledge distillation, which will be discussed in Section 3. This section will discuss each of these three types of SLMs in detail. We list the training details of some common SLMs in Table 2 and the architecture details in Table 3.

### 2.2.1. Small variants of large models

**OPT** [44]: OPT represents a series of open-source pre-trained models that replicate the performance and scale of the GPT-3 [45] model, with parameter sizes ranging from 125M to 175B. The authors provide a detailed discussion on the training process and evaluation results of the OPT models. The hyperparameter settings of the OPT models largely follow the design principles of GPT-3. To prevent underflow, OPT employs dynamic loss scaling during training described in [46].

**BLOOM** [4]: BLOOM is a multilingual open-source LLM that was trained on the ROOTS corpus [32]. To mitigate training instabilities, an additional layer normalization step was introduced after the first embedding layer. BLOOM is available in six versions, with parameter sizes ranging from 560M to 176B. The authors provide a comprehensive discussion on the creation of the ROOTS training dataset, as well as the architecture and design of the tokenizer used in BLOOM.

**GLM** [47]: GLM introduces a General Language Model (GLM) based on autoregressive blank infilling. This architecture modifies the masking mechanism of the causal decoder to enable bidirectional attention for prefix tokens while maintaining unidirectional attention for generated tokens. The released pre-trained model is a bilingual LLM. Additionally, another project, ChatGLM-6B [48], is a small-scale popular bilingual dialogue model based on the GLM architecture. By integrating model quantization techniques, it allows for local deployment on consumer-grade graphics cards.

#### InternLM Series [49]:

InternLM2 [49] is a highly capable open-source LLM, available in various sizes including 1.8B, 7B, and 20B, trained on a diverse dataset and optimized for long-context modeling and robustness. It demonstrates significant advantages in modeling long contexts and conducting open-ended subjective evaluations. The pre-training dataset of InternLM2 encompasses a diverse range of data types, including text, code, and long-context data. To better support long contexts, InternLM2 employs GQA [23] to reduce inference costs and has been additionally trained to accept up to 32k tokens of context.

InternLM2.5 [50] is the next-generation iteration of InternLM2, trained on a large amount of synthetic data. It supports a better long-context and tool usage capability, and achieves advanced levels of mathematical reasoning.

#### Qwen Series [51–53]:

The architecture of the Qwen [51] model is similar to that of the LLaMA [5] series. Qwen family encompasses a diverse array of models, including general-purpose, multimodal, and fine-tuned models, extensively trained on up to 3 trillion tokens from a wide range of texts and code, covering a broad spectrum of domains. Notably, the Qwen series excels as a multilingual model, particularly demonstrating outstanding proficiency in both Chinese and English. The Qwen-1.8B model, a smaller variant within the series, delivers competitive performance across various tasks, occasionally surpassing larger models in specific scenarios.

Qwen1.5 [52] significantly enhances the model's capabilities in multilingual processing, alignment with human preferences, and understanding of long contexts. The smaller variants of Qwen1.5 include models with 0.5B, 1.8B, and 4B parameters, along with several different quantized versions. Compared to leading small models in the industry, Qwen1.5 demonstrates strong competitive performance. Starting from this generation, the smaller Qwen models utilize embedding tying techniques to reduce the parameter size.

**Table 2**

An overview of training details for selected small language models.

Model	Date	Precision	Tokens	Trained Tokens	Hardware	Datasets	Language	Institute
GLM	2021-03	–	–	–	–	Pile	en	THU
OPT	2022-05	–	180B	–	–	RoBERTa, Pile, PushShift.io Reddit	en	Meta
Bloom	2022-11	fp16	341B	410B	384 A100 80GB	ROOTS	48 languages	BigScience
Phi-1	2023-06	fp16	7B	51B	8 A100	CodeTextbook, CodeExercises	en	Microsoft
Gemini-Nano	2023-12	–	–	–	TPUv4 and TPUv5e	–	multilingual	Google
Phi-2	2023-12	fp16	250B	1.4T	96 A100 80G	–	en	Microsoft
TinyLlama	2023-09	bf16	950B	3T	16 A100 40G	SlimPajama, StarCoder	en, zh <sup>a</sup>	SUTD
Phi-1.5	2023-09	fp16	30B	150B	32 A100 40G	CodeTextbook, synthetic data	en	Microsoft
Qwen	2023-09	bf16	–	2.2T	–	–	en, zh	Alibaba
Danube	2024-01	fp8	–	1T	8 H100 80G	–	en	H2O.ai
MobileLLM	2024-02	–	1T	–	32 A100 80G	–	en	Meta
Qwen1.5	2024-02	bf16	–	2.4T	–	–	12 languages	Alibaba
OLMo	2024-02	bf16	3.0T	at least 2T	1024 MI250X 128GB, 216 A100 40G	Dolma	en	AI2
StableLM 2	2024-02	bf16	–	2T	512 A100 40G	Restruct-v1	en, de, es, fr, it, nl, pt	Stability AI
Gemma	2024-03	–	3T	–	512 TPUv5e	–	en	Google
InternLM2	2024-03	–	2T	–	–	–	en, zh	Shanghai AI Lab
Phi-3-Mini	2024-04	bf16	3.3T	–	512 H100 80G	–	en	Microsoft
OpenELM	2024-04	–	1.5T	–	128 A100/H100 80G	RefinedWeb, RedPajama, Pile, Dolma	en	Apple
Danube2	2024-04	fp8	–	3T	8 H100 80G	–	en	H2O.ai
MiniCPM	2024-04	–	1.1T	–	–	CommonCrawl, Dolma, C4, Pile, etc.	en, zh	OpenBMB
Qwen2	2024-06	bf16	7T (12T on 0.5B)	–	–	–	27 languages	Alibaba
SmollM	2024-07	bf16	–	1T	64 H100 80G	SmollM-Corpus	en	HuggingFace
Danube3	2024-07	–	–	6T (4T on 500M)	–	–	en	H2O.ai
InternLM2.5	2024-08	–	–	–	–	–	en, zh	Shanghai AI Lab
MiniCPM3	2024-09	–	–	–	–	–	en, zh	OpenBMB
Qwen2.5	2024-09	bf16	18T	–	–	–	29 languages	Alibaba
SmollM2	2024-10	bf16	–	11T	256 H100 80G	SmollM-Corpus	en	HuggingFace

Here, “Date” refers to the release date of the corresponding paper or article. “Precision” denotes the numerical precision used during training, where “bf16” stands for bfloat16 and “fp16” stands for float16. “Tokens” represents the number of tokens in datasets, and “Trained Tokens” denotes the number of tokens used for training. “Hardware” refers to the hardware used for training. “Datasets” lists the datasets used for training. Mentioned datasets are RoBERTa [29], Pile [30], PushShift.io Reddit [31], ROOTS [32], SlimPajama [33], StarCoder [34], CodeTextbook/CodeExercises [21], Restruct-v1 [35], RefinedWeb [36], RedPajama [37], Dolma [38], CommonCrawl [39], C4 [40], WuDuo [41], Cbook [42], SmollM-Corpus [43]. “Language” indicates the language(s) of the datasets, some models’ language support is too extensive to list here. “Institute” refers to the institution that developed the model.

<sup>a</sup> These models are monolingual, but have multilingual versions.

Qwen2 [53] introduces two smaller-sized versions: 0.5B and 1.5B. The training dataset of Qwen has been expanded to include 27 additional languages beyond English and Chinese. Qwen2 features enhanced support for longer context lengths and exhibits significant improvements in coding and mathematical capabilities. Across a spectrum of competencies including natural language understanding, knowledge, coding, mathematics, and multilingual processing, Qwen2 significantly surpasses its predecessors.

Qwen2.5 [54] demonstrates significant improvements in instruction following, long-text generation, long-context understanding, structured data understanding, and structured output generation.

### 2.2.2. Pre-trained small models

**Phi Series** [21,55–57]: The Phi series exemplifies a strategic evolution in small language model development through data-centric scaling

and architectural refinement. Phi-1 [21] pioneered the quality-over-quantity paradigm, demonstrating coding proficiency with < 7B tokens of curated Python textbooks and GPT-3.5-synthesized examples, though its specialized training limited generalizability and revealed prompt sensitivity. Building on these insights, Phi-1.5 [55] expanded into common sense reasoning using 150B tokens of textbook-style synthetic data, achieving toxicity reduction through domain-focused training while maintaining a 1.3B parameter scale. Subsequent architectural innovations in Phi-2 [56] introduced RoPE positional encodings and GeLU activations, enabling the 2.7B parameter model to match 13B counterparts through hybrid training on filtered web/synthetic tokens. The latest Phi-3-Mini [57] adopts Llama2-inspired block structures with SwiGLU activations and RMSNorm, scaling training to 3.3T tokens while implementing rigorous data filtering to achieve reasoning parity with Mixtral 8x7B and GPT-3.5, despite persistent challenges in factual knowledge retention and safety alignment inherent to its compact

**Table 3**

Architecture details for selected small language models.

Model	Size	Arch.	Atten.	Vocab.	Tokenizer	Norm	PE	Act.	Bias	L	H	d	CL
OPT	125M	–	MHA	50k	BPE	pre LayerNorm	Learned	ReLU	yes	12	12	768	2k
	350M									24	16	1024	
	1.3B									24	32	2048	
	2.7B									32	32	2560	
Bloom	560M	–	MHA	250k	BPE	pre LayerNorm	ALiBi	GeLU	yes	24	16	1024	2k
	1.1B									24	16	1536	
	1.7B									24	16	2048	
	3B									30	32	2560	
GLM	2B	GPT2	MHA	50k	BPE	post LayerNorm	Learned	GeLU	yes	36	32	2048	1k
InternLM2	1.5B	–	GQA	93k	BPE	pre RMSNorm	RoPE	SwiGLU	no	24	16	2048	32k
InternLM2.5	1.8B	–	GQA	93k	BPE	pre RMSNorm	RoPE	SwiGLU	no	24	16	2048	32k
Gemini-Nano	1.8B 3.25B	–	–	–	SentencePiece	–	–	–	–	–	–	–	–
TinyLlama	1.1B	Llama	GQA	32k	BPE	pre RMSNorm	RoPE	SwiGLU	no	22	32	2048	2k
Qwen	1.8B	Llama	MHA	152k	BPE	pre RMSNorm	RoPE	SwiGLU	qkv	24	16	2048	8k
Qwen1.5	0.5B	Llama	MHA	152k	BPE	pre RMSNorm	RoPE	SwiGLU	qkv	24	16	1024	32k
	1.8B									24	16	2048	
	4B									40	20	2560	
Qwen2	0.5B 1.5B	Llama	GQA	152k	BPE	pre RMSNorm	RoPE	SwiGLU	qkv	24	14	896	32k
Qwen2.5	0.5B	Llama	GQA	152k	BPE	pre RMSNorm	RoPE	SwiGLU	qkv	24	14	896	32k
	1.5B									28	12	1536	
	3B									36	16	2048	
Gemma	2B	–	MQA	256k	SentencePiece	pre RMSNorm	RoPE	GeGLU	–	18	8	2048	8k
Phi-1	1.3B	–	MHA	51k	BPE	pre LayerNorm	RoPE	GeLU	yes	24	32	2048	2k
Phi-1.5	1.3B	–	MHA	51k	BPE	pre LayerNorm	RoPE	GeLU	yes	24	32	2048	2k
Phi-2	2.7B	–	MHA	51k	BPE	pre LayerNorm	RoPE	GeLU	yes	32	32	2560	2k
Phi-3-Mini	3.8B	–	MHA	32k	BPE	pre RMSNorm	RoPE	SwiGLU	no	32	32	3072	4k, 128k <sup>b</sup>
StableLM 2	1.6B	–	MHA	100k	BPE	pre LayerNorm	RoPE	SwiGLU	qkv	24	32	2048	4k
OpenELM	270M-3B	–	GQA	32k	BPE	pre RMSNorm	RoPE	SwiGLU	no	16–36	– <sup>a</sup>	1280	2k
OLMo	1B	–	MHA	50k	BPE	pre non-parametric LayerNorm	RoPE	SwiGLU	no	16	16	2048	2k
MiniCPM	1.2B 2.4B	–	GQA MHA	123k	BPE	pre RMSNorm	RoPE	SwiGLU	no	52 40	8 36	1536 2304	4k, 128k <sup>b</sup>
MiniCPM3	4B	–	MLA	73k	BPE	pre RMSNorm	RoPE	SwiGLU	no	62	40	2560	32k
MobileLLM	125M 350M	Llama	GQA	32k	BPE	pre RMSNorm	RoPE	SwiGLU	no	30 32	9 15	576 960	2k
SmolLM & SmolLM2	135M	Llama	GQA GQA MHA	49k	BPE	pre RMSNorm	RoPE	SwiGLU	no	30	9	576	2k (SmolLM) 8k (SmolLM2)
	360M									32	15	960	
	1.7B									24	32	2048	
Danube	1.8B	Llama2	GQA	32k	BPE	pre RMSNorm	RoPE	SwiGLU	no	24	32	2560	16k
Danube2	1.8B	Llama2	GQA	32k	BPE	pre RMSNorm	RoPE	SwiGLU	no	24	32	2560	8k
Danube3	500M 4B	Llama	GQA	32k	BPE	pre RMSNorm	RoPE	SwiGLU	no	16 24	16 32	1536 3840	8k
Sheared LLaMA	1.23B 3.21B	Llama	GQA	128k	BPE	pre RMSNorm	RoPE	SwiGLU	no	16 28	32 24	2048 3072	128k
Llama 3.2	1.3B 2.7B	Llama	MHA	32k	BPE	pre RMSNorm	RoPE	SwiGLU	no	24 32	16 20	2048 2560	4k

Here, “Size” refers to the number of parameters in the model. “Arch.” denotes the basic architecture used in the model. “Atten.” represents the attention mechanism used in the model, where “MHA” stands for multi-head attention, “GQA” stands for generalized query attention and “MQA” stands for multi-query attention. “Vocab.” denotes the size of the vocabulary. “Tokenizer” refers to the tokenizer used for the model. “Norm” denotes the normalization layer used in the model. “PE” represents the positional embedding used in the model. “Act.” denotes the activation function used in the model. “Bias” indicates whether the model uses bias in the attention mechanism. “qkv” means only the QKV layer of attention have bias. “L” represents the number of layers in the model. “H” denotes the number of attention heads. “d” represents the hidden dimension of the model. “CL” denotes the context length of the model.

<sup>a</sup> OpenELM has different numbers of attention heads for each layer.

<sup>b</sup> Models with long context support are another independent model.

3.8B parameter design. The series highlights diminishing returns of data scaling without quality control, though fundamental limitations in knowledge density for sub-4B models remain an open research challenge.

**Gemma** [58]: Gemma, a lightweight model released by Google’s DeepMind team. These models demonstrate robust text understanding

and reasoning capabilities. The 2B version is trained on a dataset of 3 trillion tokens, which primarily consists of English data from web documents, mathematics, and code. Evaluations reveal that Gemma not only excels in dialogue and reasoning tasks but also outperforms competitors on six standard safety benchmarks and in human side-by-side evaluations.

**TinyLlama** [59]: TinyLlama employs the same architecture and tokenizer as Llama2 [6] and is pre-trained on a mixed dataset of 30 trillion tokens of natural language and code data. Despite its relatively small size, TinyLlama surpasses OPT-1.3B [44] and Pythia-1.4B [60] in various downstream tasks. Its compactness makes it suitable for end-user applications on mobile devices and for various applications or research studies that require limited computation and memory usage.

**MobileLLM** [27]: In the pursuit of energy and memory efficiency, MobileLLM investigates use cases for LLMs with fewer than one billion parameters on-device. The study emphasizes the considerable importance of model architecture for LLMs of this scale. It demonstrates that for smaller LLMs, depth is more crucial than width, leading MobileLLM to adopt a deeper and thinner architecture rather than a shallower and wider one. MobileLLM employs GQA [23] and a cache-efficient weight sharing strategy where duplicating the decoding block twice can enhance accuracy. Furthermore, it reuses the input embedding weights as the output fully connected layer weights, resulting in a more efficient and compact model architecture.

**MobiLlama** [61]: MobiLlama presents accurate yet efficient small language models of 0.5B and 0.8B parameters, optimized for resource-constrained devices. In a significant architectural decision, MobiLlama employs a shared FFN design for all blocks in the model, resulting in a 60% reduction in the total number of trainable parameters compared to non-shared models. Despite these reductions, MobiLlama achieves performance comparable to OLMo-1.17B [62] and TinyLlama-1.1B [59]. Remarkably, it does so with significantly less training data and training time, and substantially reduced resource requirements.

**MiniCPM Series** [22]: This evolving family demonstrates efficient small language models through architectural innovations and training optimizations. The original MiniCPM [22] (2.4B/1.2B) employs shared input-output embeddings, GQA, and a deep-and-thin structure, enhanced by a three-phase learning rate scheduler that prioritizes high-quality SFT data in the final decay stage. Its successor MiniCPM3-4B [63] introduces MLA to blend GQA's memory efficiency with MHA's expressiveness through learned attention head grouping, achieving capabilities comparable to 7B-9B models. Key advancements include native 32 K context support with full accuracy in needle-in-haystack tests, and function calling capabilities outperforming many 7B-9B models and GPT-3.5-Turbo. The series shows exceptional cross-domain performance in mathematical reasoning, Chinese instruction following, and tool invocation.

**OpenELM** [64]: OpenELM is a compact language model trained and evaluated on public datasets. The core of OpenELM is a layer-wise scaling strategy, which employs smaller hidden dimensions in the attention and feed-forward networks of the earlier layers, gradually increasing the hidden dimensions in the later layers. As a result, each layer in the model has a different number of parameters. This strategy allows OpenELM to make better use of the available parameter budget, achieving higher accuracy. With a parameter budget of approximately one billion, OpenELM's accuracy improves by 2.36% compared to OLMo [62], while requiring half the pre-training tokens.

**OLMo** [62]: OLMo introduces a comprehensive framework, encompassing everything from the model, data, training, to evaluation tools, along with detailed metrics collected during training runs. Architecturally, OLMo employs a non-parametric layer norm to achieve faster speed than RMSNorm [65]. It utilizes their open-source pre-training dataset, Dolma [38], and data processing and analysis tools, training on at least 2 trillion tokens using a linear learning rate decay schedule. Remarkably, both the 1B and 7B models surpass their same-scale competitors in zero-shot scores across multiple tasks.

**Stable LM 2** [35]: The study focuses on training a small-scale large language model in a reproducible manner, providing a complete and transparent set of principles for designing pre-training datasets and their sources. Experimental results reveal that Stable LM 2 [35] significantly outperforms other similarly-sized open-source models across various tasks and languages. Additional measurements of the model's

throughput performance on edge devices show that it achieves approximately 50+ tokens per second on their consumer-grade products. Notably, using lower precision can double the throughput.

**Pythia** [60]: Pythia, a suite of models primarily designed for scientific research, encompasses 12 models of varying sizes to better understand model behavior in terms of training and scaling. All models are pre-trained on the same dataset in the same order. Despite the use of several techniques that could potentially impair the performance of smaller models, Pythia's smaller models still achieve performance comparable to OPT [44]. Additionally, the study provides three case analyses summarizing the impact of data bias on learned behaviors, the influence of training order on memorization, and the effect of pretraining term frequencies on task performance.

#### SmollM Series [43,66]:

SmollM [43] is an excellent small model released by the HuggingFace. The small-sized model adopts a similar architecture to MobileLLM [27]. They curated and constructed a high-quality dataset called SmollM-Corpus, which consists of training data from Mixtral [67] synthetic data, educational web content, and filtered Python code. Evaluation results of the model demonstrate advanced performance in common sense reasoning and world knowledge, with the 1.7B model showing strong Python coding performance. SmollM proves through experiments that small models can achieve good performance as long as they are trained sufficiently and the data quality is good. SmollM2 [66] builds upon its predecessor by adopting a more diverse dataset, resulting in significant improvements in instruction following, knowledge, reasoning, and mathematics.

#### Danube Series [68,69]

H2O-Danube-1.8B [68] is the foundational model trained by H2O.ai, adopting an architecture similar to Llama2 [6] and delivering performance comparable to Qwen and Stable LM 2 [35]. H2O-Danube2-1.8B [68] inherits the architecture and weights from H2O-Danube-1.8B, continuing training on 2T tokens to achieve higher accuracy, better performance, and long-context support. H2O-Danube3-4B [69] is trained from scratch on a curated 6T token dataset. In addition to its compact size and improved performance, this model outperforms competitors across various academic benchmarks.

#### 2.2.3. Compressed small models

Contrary to training a SLM from scratch, model compression methods transfer the capabilities of pre-trained large models to more compact and efficient small architectures for deployment in resource-constrained environments. The most common techniques include pruning and distillation, both of which can preserve specific performance of the original model to a certain extent, and even achieve better compression results by combining multiple methods. There have been many practices generating small models using such methods, and this section will introduce these models.

**Gemini-Nano** [70]: Gemini-Nano, a member of Google's Gemini Family, is designed as a series of small models for on-device applications. Available in two sizes, 1.8B and 3.25B parameters, these models are specifically aimed at devices with low and high memory capacities, respectively. The training process involves distillation from larger models within the Gemini series. Impressively, Gemini-Nano models exhibit remarkable capabilities across a variety of tasks, including reasoning, STEM, coding, multimodal, and multilingual tasks.

**LaMini-LM** [71]: LaMini-LM employs Instruction Following distillation to offer a spectrum of SLMs varying in size and architecture. To accomplish this, a large-scale offline-distillation instruction dataset comprising 2.58 million examples was created, utilizing GPT-3.5-Turbo to generate responses for each instruction. In a comprehensive evaluation, including automatic assessments of downstream NLP tasks as well as human evaluations of general usage, hallucinations, and toxicity, these models achieved superior performance compared to previous methods.

**Baby Llama** [72]: Baby Llama, a contestant in the BabyLM challenge [73], employs Instruction Following distillation to train a compact teacher model on a dataset comprising 10 million words. This process culminates in the distillation of an even smaller LLaMA model with 58 million parameters. Remarkably, on a majority of zero-shot tasks, this distilled model outperforms its teacher model. The findings of this study suggest that distillation, when the teacher model is trained on a sufficiently small dataset, not only preserves the full performance of the teacher model but can also lead to enhanced performance.

**MiniLLM** [74]: MiniLLM introduces a white-box knowledge distillation method for LLMs that minimizes the reverse Kullback–Leibler Divergence (KLD) to prevent the student model from overestimating low-probability regions in the teacher model’s distribution, thereby enhancing the quality of generated samples. Experiments conducted on multiple small-sized variants from three different architectures serving as student models demonstrate that MiniLLM achieves lower exposure bias, better calibration, and higher performance in long text generation, along with commendable generative diversity. The student models range in size from 120M to 7B parameters.

**MiniMA** [75]: MiniMA reveals a linear correlation between the optimal teacher model size and the student model size across different model architectures and data scales, with the optimal student model size being 0.4 times that of the teacher model. Leveraging this law, a highly computationally efficient 3B model was developed through distillation from the LLaMA2 7B [6] model. Experiments demonstrate that MiniMA sets a new computational performance Pareto frontier among existing 3B models on common benchmarks. Its instruction-finetuned version outperforms 3B competitors in the GPT-4 evaluation and is even comparable to several 7B chat models.

**Sheared LLaMA** [76]: Sheared LLaMA proposes an economically efficient structured pruning method to develop small yet competitive language models. This method first prunes from the LLaMA2-7B model and then trains using only 50 billion tokens. It also introduces dynamic batch loading technology to dynamically adjust the proportion of data from different domains, achieving effective accuracy recovery across different domains in the recovery phase after pruning.

**Llama 3.2** [77]: Llama 3.2 introduces lightweight text models suitable for edge and mobile devices, with sizes of 1B and 3B. The Llama Team employs pruning and distillation methods to generate these small models. Initially, structured pruning is used to create 1B and 3B initial models from the Llama 3.1 8B model. Subsequently, knowledge distillation is applied, with the Llama 3.1 8B and 70B models serving as teacher models, to restore the performance of the pruned models. In the post-training stage, the team also conducts synthetic data and long-context training on the models. Llama 3.2 also provides quantized versions of these models, with quantization schemes optimized for Arm CPUs, making the models inherently suitable for Arm-based edge hardware devices.

### 2.3. Perspectives on edge performance

The limited resources of edge devices pose significant challenges for LLM inference. Compared to LLMs, SLMs have lower computational complexity and storage requirements, making LLM inference on edge devices feasible. However, edge devices exhibit significant heterogeneity, with large differences across architectures and models. Therefore, selecting appropriate devices based on workloads is critical for the successful deployment of SLMs on edge devices.

Lu et al. [18] conducted a detailed investigation and experiments on the Capabilities and Runtime Cost of SLMs on edge devices. They highlighted that, in addition to the number of model parameters, the model architecture is also a critical factor influencing performance. Quantization can benefit the decoding phase of model inference by reducing memory access overhead, while hardware with higher parallelism tends to gain greater advantages during the prefill phase. They also identified that GEMV computation is the most time-consuming operation in SLMs,

**Table 4**

Specifications of Representative Edge Devices [78].

Device	CPU Core	CPU Frequency	Memory	Disk
RaspberryPi 5B	4	2.4 GHz	4 GB	128 GB
Jetson AGX Orin	12	2.2 GHz	32 GB	64 GB
Mac Mini	8	3.23 GHz	16 GB	494.38 GB

and when performing long-context inference, the KV cache occupies the majority of memory. Islam et al. [78] also conducted extensive experiments to evaluate the performance of multiple SLMs on typical edge hardware platforms. Details of the three devices used are provided in Table 4. The experiments were conducted using the Llama.cpp framework with fp16 and int4 quantized weight, and employed the WikiText-2 [79] dataset. Model performance was assessed by fixing the same request tokens and output token lengths. The focus was on two key metrics: throughput and perplexity. Throughput indicates the speed of inference on the device, with higher values being better. Perplexity indicates the language capabilities of the model, with lower values being better. The results are presented in Table 5.

The comparison of throughput and perplexity before and after quantization reveals that while quantization slightly increases perplexity, it significantly boosts throughput. This demonstrates the effectiveness of quantization strategies in enhancing processing speed, particularly in resource-constrained environments like Raspberry Pi.

It is important to note that throughput and perplexity alone do not fully capture the challenges of edge devices. Different environments may have varying requirements. For examples:

1. Thermal and Power Constraints: Prolonged high workloads can lead to heat accumulation and significant battery drain, which can be costly for mobile devices.
2. System Resource Sharing: During inference, users may perform other parallel tasks, and the inference process should not monopolize all system resources.

These challenges highlight potential future research directions for deploying SLMs on edge devices. Therefore, the selection of both edge devices and models must be carefully analyzed based on specific scenarios and requirements.

## 2.4. Prospects and applications

### 2.4.1. SLMs for specialized domains

Even though SLMs do not possess the powerful generalization capabilities of LLMs, they often exhibit comparable performance in certain downstream tasks or specialized domains. Given specific task requirements, SLMs offer advantages such as low resource consumption, high throughput, low latency, low power consumption, privacy protection, and personalization. Therefore, they hold promising application prospects in scenarios involving mobile devices, IoT devices, and edge computing devices.

Similar to the application of LLMs in specialized domains, these domain specialized LLMs are typically fine-tuned from base models. The most common application scenarios in specialized domains are code generation and mathematical reasoning. We can already see model variants fine-tuned for these two domains in many pre-trained models, such as DeepSeek-Coder [80] CodeGemma [81], Orca-Math [82], etc. In addition, some models are specifically trained for these domains as well [55,83]. In other domains, some general small models also demonstrate their advantages in specific areas, such as law and economics [84]. However, whether smaller models have competitive application prospects in these domains remains to be further researched, as most domain specialized LLMs, even at their smallest sizes, reach 7/8B parameters.

**Table 5**  
Performance of SLMs/LLMs on Representative Edge Devices [78].

Model Name	Parameter Size	Weight Bit-width	Device	Throughput (tokens/s)	Perplexity
TinyLlama	1.1B	16 bit	RaspberryPi 5B	22.85	8.4444
		16 bit	Jetson AGX Orin	40.74	8.4444
		16 bit	Mac Mini	694.69	8.4444
Phi-3	3.8B	16 bit	RaspberryPi 5B	4.77	7.0836
		16 bit	Jetson AGX Orin	13.31	7.0836
		16 bit	Mac Mini	211.88	7.0836
TinyLlama	1.1B	4 bit	RaspberryPi 5B	27.61	8.762
		4 bit	Jetson AGX Orin	46.52	8.762
		4 bit	Mac Mini	604.94	8.7438
Phi-3	3.8B	4 bit	RaspberryPi 5B	7.22	7.4731
		4 bit	Jetson AGX Orin	13.73	7.4731
		4 bit	Mac Mini	176.83	7.376

#### 2.4.2. Application in multimodal models

Due to the efficiency of small text base models, they often play a core component role in some multimodal large language models (MLLM) to achieve efficient inference on mobile SoCs, such as MobileVLM [85], MobileVLM V2 [86], TinyLLaVA [87], LLaVA-Phi [83], Bunny [88], DeepSeek-VL [89], Vary-toy [90], MiniCPM-V [91], etc. For these types of models, researchers only need to train the mapping relationship between the text representation space and different modality representation spaces (such as vision, audio, video, etc.). Since the SLM contributes the majority of MLLM parameters, its selection is closely related to the lightweight characteristics of these MLLMs [92].

#### 2.4.3. AI agents for edge

AI agents refer to systems that can interact with environment on behalf of a user, automate the utilizing of tools, and information collecting. The ability to make function calls is essential for implementing AI agents. In this direction, Octopus has gone further. The Octopus series is a specialized AI agent model tailored for edge devices. Unlike chat models, these models focus more on providing API call capabilities, which are crucial for integrating LLMs externally. To this end, Octopus [93] has designed a dataset from software API documents and condition masking techniques to enhance the effectiveness of LLMs in API interactions. Octopus v2 [94] is specifically trained for Android APIs, facilitating edge computing devices by introducing functional tokens rather than RAG-based methods. Octopus v3 [95] equips the series with multimodal capabilities and compact parameters of less than 1B, enabling good performance on resource-constrained devices such as Raspberry Pi. Octopus v4 [96], the latest version in the series, offers query redirection functionality. With this, the Octopus team has devised a concept of models that redirect different queries to different vertical domains, achieving optimal performance through multi-node reasoning.

#### 2.4.4. Mobile assistants

More and more mobile device manufacturers have been embedding small language models into mobile smartphones and other devices. Recent practices include Apple's Apple Intelligent [97] and Google's Gemini [70]. Other companies like Huawei, Xiaomi, and OPPO have also been actively exploring the integration of small language models into their mobile devices.

As a case study, Apple Intelligence [97] implemented various optimization strategies to adapt their models for their devices. To improve the performance of small models on specific tasks, Apple's team fine-tuned smaller models using PEFT methods and developed a runtime-swappable adapter architecture. This approach allows a single base model to specialize in dozens of such tasks. By dynamically loading low-cost adapters, the base model can handle specific tasks with greater specialization while maintaining the responsiveness of the operating system. Additionally, to fit AFM within the limited memory budgets of edge devices and reduce inference costs, Apple employed model quantization techniques. Specifically, the base model uses mixed-precision

quantization, where each layer is quantized with different bit-widths, compressing the model in production to an average of approximately 3.7 bits per weight. To address performance degradation caused by quantization, Apple trained a set of LoRA adapters to efficiently recover performance. This approach achieved 7%–18% accuracy recovery on AlpacaEval and 5%–10% accuracy improvement on GSM8K. Apple's design effectively enables efficient switching of the base model across different downstream tasks while significantly reducing the model size without substantially compromising performance.

In addition to industrial practices, many research works are also exploring how to deeply integrate these language models into mobile devices to provide more convenience. AppAgent [98] has developed a multimodal intelligent framework based on LLMs, enabling agents to mimic human interaction styles and operate smartphone applications. MM-Navigator [99] has designed an intelligent system based on GPT-4V, focusing on smartphone graphical user interface navigation tasks, featuring advanced screen understanding and precise action localization capabilities. AutoDroid [100] has crafted a task automation system for Android mobile devices driven by large language models, which can handle any task on any Android application by integrating LLMs with application-specific knowledge, eliminating the need for manual operation. GptVoiceTasker [101] is a virtual assistant capable of comprehending user commands and executing corresponding device interactions, enhancing the user experience and task efficiency on mobile devices, and continuously learning from historical user commands to further improve execution efficiency.

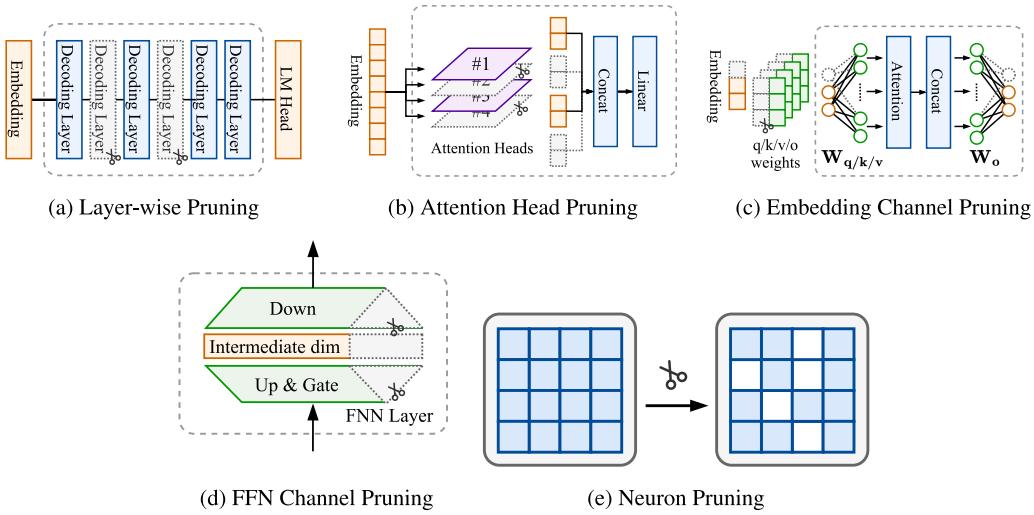
However, the flexibility and scalability of current mobile intelligent assistants remain limited. Their intelligence is far from sufficient, particularly in understanding user intent, reasoning, and task execution. Research on LLM-based agents is still in its early stages, with relatively weak task execution capabilities and a narrow range of supported functions. Moreover, ensuring the efficiency, reliability, and usability of personal agents requires addressing numerous critical performance and security challenges. LLMs rely on large-scale parameters to deliver better service quality, which inherently conflicts with the resource, privacy, and security constraints of personal agents.

### 3. Model compression

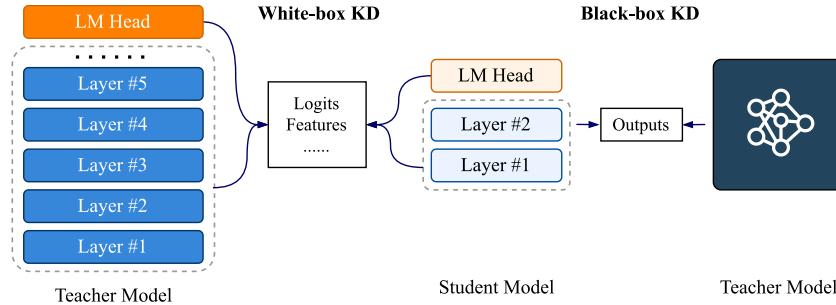
Model compression reduces model size by removing redundant information. This can be achieved through pruning, quantization, distillation, or low-rank decomposition. In this section, we provide an overview of these techniques.

#### 3.1. Pruning

Pruning is a technique for model compression that eliminates redundant structures or weights, thereby compressing the model. Depending on the granularity, existing techniques for pruning LLMs can be categorized into structured and unstructured pruning. We illustrate five granularities of pruning in Fig. 4. Structured pruning [76,102–107]



**Fig. 4.** Five granularities of pruning for large language models. Figs. 4(b), 4(c), 4(d), and 4(e) indicate structured pruning, while Fig. 4(a) indicates unstructured pruning.



**Fig. 5.** Illustration of knowledge distillation techniques.

involves removing structured components, such as multiple channels or blocks from the parameter matrices of LLMs. Some studies [76,103, 105,106] have combined fine-tuning, quantization, or even training to mitigate the loss of accuracy and enhance compression efficiency. Conversely, unstructured pruning [108–110] does not consider the internal structure of LLMs. It induces sparsity in the weight matrix by pruning at the neuron level, which necessitates specialized hardware for accelerating tensor operations. After pruning, the performance of the model is typically restored through fine-tuning. However, due to the huge computational cost associated with full-parameter fine-tuning of LLMs, current large model pruning techniques often forego the fine-tuning step [108,111] or incorporate parameter-efficient fine-tuning to mitigate these costs [103,105]. Given that pruning inevitably leads to a loss in model performance and full-parameter fine-tuning of the model is challenging, the practicality of such methods for LLMs remains to be enhanced.

### 3.2. Knowledge distillation

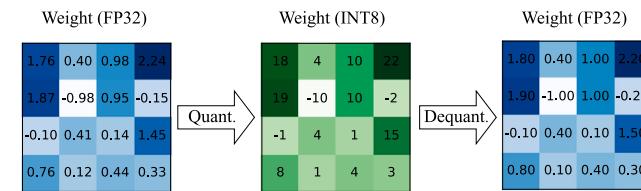
Knowledge distillation techniques treat LLMs as teacher models, using the supervisory information of the teacher model to train a smaller student model. Existing research on LLMs can be classified into white-box and black-box distillation, as shown in Fig. 5. White-box distillation methods [74,75,112,113] use both the internal information and output of the large model to train the student model, whereas black-box distillation methods [114–120] assume that the internal structure of the large model is invisible and only use the output of the teacher model to train the student model. Unlike the knowledge distillation of general neural networks, LLM distillation focuses more on the transfer of knowledge rather than compression of the architecture [121]. When the parameter size of the large model reaches a certain level, it exhibits

“emergent abilities”, i.e., astonishing performance in handling complex tasks. This feature can help small models learn to cope with complex tasks, thereby giving rise to black-box distillation methods based on Chains-of-Thought (CoT) [114–118], Incremental Context Learning (ICL) [122], and Instruction Following (IF) [71,119,120] capabilities. The knowledge distillation of LLMs is usually used to distill domain knowledge into small models that edge devices can bear, for specific downstream tasks [112]. However, the expressive power of small models is limited compared to LLMs, and users need to further balance between model capability and model size.

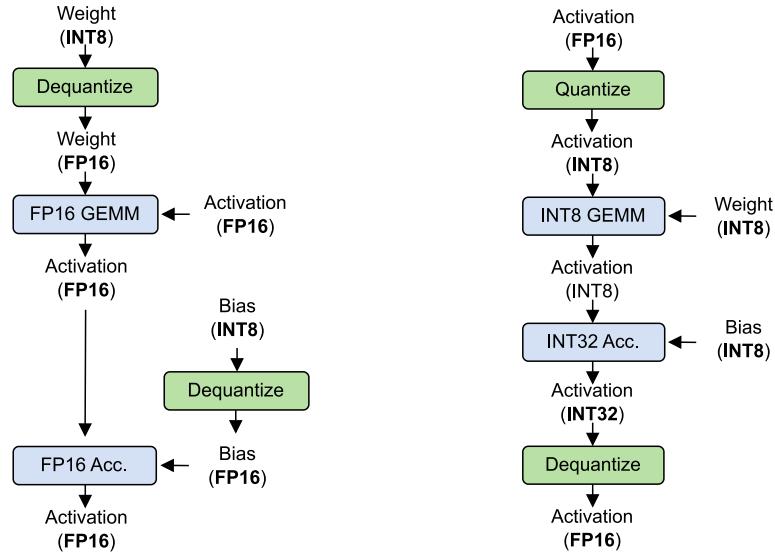
### 3.3. Quantization

Model quantization methods convert the floating-point representation of weights or activation values into lower-precision numerical representations, making full use of the numerical representation space while minimizing errors. The mainstream quantization schemes are Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT). PTQ directly converts the weights of the trained model into a low-precision format without modifying the model architecture or retraining, offering simplicity and efficiency compared to QAT [123]. QAT integrates the quantization process into the model training process, enabling the model to adapt to the low-precision storage format and achieve lower precision loss. Several representative quantization methods are shown in Table 6. Recent works like LLM-QAT [124], EfficientQAT [125] have made efforts to minimize the burden of retraining through knowledge distillation and partial retraining. Another work, EdgeQAT [126], has designed a QAT method for lightweight LLMs on edge devices which is guided by entropy and distribution.

Despite the efforts of the aforementioned studies, the cost of retraining LLMs remains substantial. Consequently, PTQ has emerged as



(a) Quantization on Weights



(b) Weight-only Quantization

(c) Weight-Activation Quantization

Fig. 6. An illustration of quantization before and after, as well as the inference schema for weight-only quantization and weight-activation quantization.

the mainstream technique for quantizing LLMs [127]. The application of PTQ quantization technology in LLMs includes two mainstream directions: weight-only quantization and weight activation value quantization, Fig. 6 illustrates their main differences during inference.

**Weight-only Quantization.** Weight-only Quantization quantizes only the weights of the model. The low-precision weights are then dequantized to full precision to compute with activation values. Although weight-only quantization does not accelerate LLM inference computationally, it reduces the storage size of weights, thereby reducing memory bandwidth requirements during matrix loading and improving LLM inference speed. Many works [127–131] are related to weight-only quantization, some of them were inspired by several insights into compensating for the quantization errors, like outlier separation [132], second-order approximation compensation [128], and distribution smoothing [133] etc. These methods are not always mutually exclusive, and can be combined. For example, SpQR [129] proposed further optimization strategies for the quantization scheme of GPTQ [128], separating outliers and using sparse matrix storage, and adopting a mixed-precision double-layer quantization strategy for non-outlier weights, further reducing the performance loss of LLMs after quantization. AWQ [127], based on the viewpoint of the imbalance of weight importance in LLM, selects important weights according to activation values and introduces a smoothing factor to reduce the quantization error of important weights, ultimately achieving excellent quantization schemes suitable for various LLMs. OWQ [130] theoretically analyzed the amplifying effect of outlier activation values on weight quantization error, and introduced a mixed-precision quantization scheme for the weight matrix based on AWQ. In addition to the aforementioned methods, AQLM [131] combines the advantages of vector quantization and multi-codebook quantization. It represents vectors as the sum of multiple codebook vectors, a process known as

Additive Quantization. This allows for a more precise approximation of the original data. AQLM is an asymmetric, non-linear vector quantization method, and its codebooks require a calibration process to learn the optimal solution.

**Weight-Activation Quantization.** Weight-activation quantization quantizes both weight matrices and activation values. This method reduces memory bandwidth requirements and improves the inference speed of LLMs by replacing high-precision operators with low-precision operators. For some representative works, LLM.int8() [132] discovered the emergence of outlier features and proposed mixed-precision method to preserve high-precision outliers. ZeroQuant [134] proposed a fine-grained hardware-friendly quantization scheme, which uses different quantization granularities for weights and activation values, and employs layer-wise knowledge distillation to mitigate the accuracy loss after quantization. SmoothQuant [133] smooths the distribution of activation values, shifting the difficulty of activation value quantization to model weight quantization, and on this basis, implements a W8A8 quantization scheme for LLMs. Outlier Suppression+ [135], based on Outlier Suppression [136], combines the characteristics of asymmetric outlier distribution and concentration in specific channels, and uses channel-level transformation and scaling to alleviate the errors caused by asymmetric outliers. OliVe [137] adopts outlier-victim pair quantization, considering that outliers are more important than normal values, and handles local outliers with methods that minimize hardware overhead. QLLM [138] proposed an adaptive channel reorganization method to effectively handle outliers in activation values, and uses calibration data to offset quantization errors. FPTQ [139] designed a novel W4A8 post-training quantization method, combining the advantages of W8A8 and W4A16, and integrating fine-grained weight quantization with layer-wise activation quantization strategies, further maintaining the original performance of the model. Agile-Quant [140]

**Table 6**  
Comparison of several representative LLM quantization methods.

Method	Weight	Activation	Uniform	Symmetric	Calibration	Mixed Precision
LLM-QAT	✓	✓	✓	✓	-	✗
EfficientQAT	✓	✗	✓	✗	-	✗
EdgeQAT	✓	✓	✓	✓	-	✓
GPTQ	✓	✗	✓	✗	✓	✗
SpQR	✓	✗	✓	✗	✓	✓
AWQ	✓	✗	✓	✗	✓	✗
OWQ	✓	✗	✓	✗	✓	✓
AQLM	✓	✗	✗	✗	✓	✗
LLM.int8()	✓	✓	✓	✗	✗	✓
ZeroQuant	✓	✓	✓	✓	✗	✗
SmoothQuant	✓	✓	✓	✓	✗	✗
Outlier Suppression	✓	✓	✓	-	✓	✓
PB-LLM	✓	✓	-	-	✓	✓
BiLLM	✓	✓	-	-	✓	✗
BitNet	✓	✓	-	-	-	✗

introduces a new quantization framework for LLMs on edge devices, incorporating an activation-aware token pruning technique to mitigate outliers and adverse effects on attention. Agile-Quant achieves end-to-end acceleration across multiple edge devices by leveraging a SIMD-based 4-bit multiplier and an efficient TRIP matrix multiplier.

**Binary Quantization.** In addition to the aforementioned low-precision quantization schemes, some studies focus on improving the performance of binary neural networks in LLMs. For instance, PB-LLM [141] employs partial binarization to avoid performance degradation caused by salient weight binarization, and restores the performance by employing PTQ and QAT. BiLLM [142], on the other hand, preserves the residual binary matrix of weights to reduce the quantization error of salient weights. BitNet [143] introduces a new binary Linear layer and trains the binary LLM by QAT method. Building on this, BitNet b1.58 [144] introduces a new negative value, exploring the performance of LLMs in a ternary system.

### 3.4. Low-rank decomposition

Low-rank decomposition leverages the low-rank characteristics of the model weight matrix, approximating the matrix as two or more smaller matrices to save on the number of parameters. This technique has been widely used for efficient parameter fine-tuning of LLMs [145], but recent work has shown that it can also be used for model compression [146–148], with excellent compression results. For example, TensorGPT [148] uses low-rank tensor compression for the embedding layer, reducing the spatial complexity of LLMs and making them usable on edge devices. LoSparse [147] approximates the weight matrix as the sum of a low-rank matrix and a sparse matrix, combining the advantages of low-rank approximation and structured pruning, achieving substantial memory savings.

### 3.5. Discussion

We compared and summarized the key differences between various model compression techniques, providing a concise overview in Table 7. The table includes a comparison of advantages and disadvantages, along with recommendations for usage.

Model compression remains a pivotal research direction in AI, particularly as the growing parameter counts of large-scale models amplify the advantages of efficient compression strategies. Interestingly, these models exhibit greater sparsity compared to their smaller counterparts, creating unique opportunities for compression. However, traditional compute-intensive compression methods face scalability challenges due to prohibitive computational costs, driving a paradigm shift toward efficiency-aware algorithms. Recent studies highlight the critical role of PEFT methods in bridging the performance gap introduced by compression while maintaining computational tractability [103–105,108,111,

149]. A key insight centers on identifying and preserving structurally significant dimensions or weights that underpin model capabilities, prompting novel research into saliency-aware compression frameworks. This underscores the importance of developing optimized sparse patterns tailored to large models, advanced importance metrics for parameter retention, and resource-conscious compression algorithms as foundational research priorities.

While existing compression techniques demonstrate orthogonality with combined approaches achieving superior compression ratios, each method inevitably introduces non-trivial performance degradation. Current literature predominantly explores pairwise combinations [103, 106,108,110,129,146], whereas investigations into three or more complementary techniques remain sparse. This raises an open research question: Can sophisticated multi-technique integration push compression boundaries while maintaining acceptable accuracy thresholds? The field awaits systematic studies to quantify cumulative error propagation and establish principled frameworks for hybrid compression pipelines.

## 4. Inference optimization

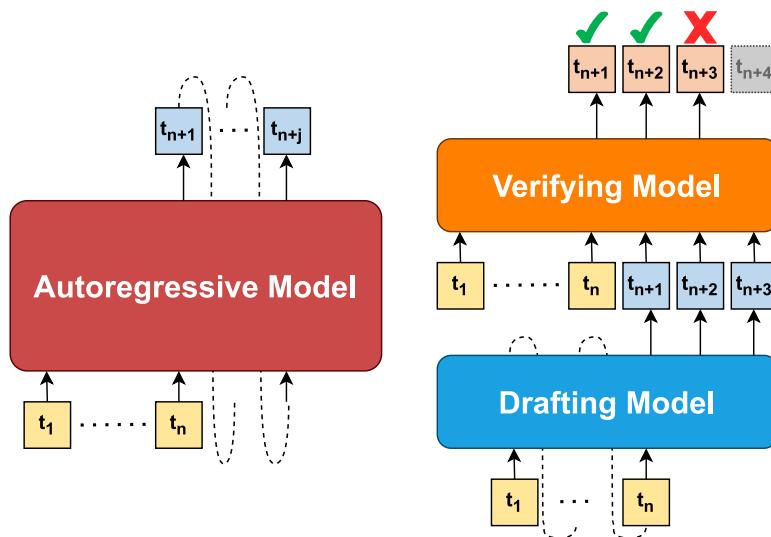
Inference optimization primarily involves enhancing the optimization of the forward process of the model, rather than merely modifying the model's weights or structure. Current large language models generally employ autoregressive decoding based on the decoder-only transformer architecture. Inference optimization aims to fully utilize software and hardware resources by optimizing repeated computations in the forward process, enhancing the efficiency of attention and linear layer operations, and even modifying the model decoding strategy. The main methods for inference optimization include speculative decoding, KV cache optimization, early exiting, kernel optimization, and memory offloading.

### 4.1. Speculative decoding

The autoregressive decoding mechanism results in inefficiencies in the generation capabilities of LLMs. Speculative decoding is a type of non-autoregressive decoding algorithm that divides the LLM decoding process into two stages: Drafting and Verifying. First, candidate tokens are generated by a drafting model. Then, the verifying model checks the correctness of candidate tokens and corrects them through a single forward propagation. Typically, the drafting model is smaller, while the verifying model is larger. We give a brief comparison of autoregressive decoding and speculative decoding in Fig. 7. The efficiency improvement in speculative decoding primarily stems from employing auxiliary models with lower resource requirements to generate candidate tokens. Verifying model only needs to parallelly judge the correctness, thereby reducing the end-to-end inference latency. The more candidate tokens accepted by the drafting model and the faster the verifying process, the

**Table 7**  
Conclusion and Comparison of Model Compression Methods.

Method	Pruning	Knowledge Distillation	Quantization	Low-Rank Decomposition
Purpose	Removes redundant weights.	Transfer knowledge from a large model to a smaller one.	Reduce the precision of weights/activations.	Decompose weights into smaller matrices.
Implementation	Removes structured components or unstructured parameters by importance and recovers performance by retraining.	Trains a smaller student model using teacher model outputs.	Apply fixed-point or lower-bit-width arithmetic.	Approximate weight matrices by low-rank matrices.
Advantages	Reduces model size. Improves inference speed.	Preserves model capacity. Gain smaller models.	Improves inference speed. Storage/Memory saving.	High compression rate. Standard matrix operators.
Disadvantages	Requires retraining. Need Hardware support for unstructured pruning.	Requires teacher model. Need to train student model.	Accuracy drop. Specialized operators required.	Accuracy drop. Sensitive to decomposition rank.
Impact of Performance	Moderate, depends on pruning strategy.	Often improves performance of smaller models.	Small accuracy degradation on proper methods.	Moderate, depends on decomposition rank and method.
Impact of Efficiency	Significant improvement.	Depending on student model size, often improves efficiency.	Significant improvement.	Moderate improvement.
Cost	Moderate, requires retraining/finetuning.	High, requires teacher model.	Low, may require retraining/finetuning.	Moderate, requires decomposition process.
Usage Advice	Use for reducing model size or hardware supporting sparse operations.	Use when need to deploy a smaller model with similar performance to a larger model.	Recommended for edge inference with limited resources.	Use for large models with high redundancy.



**Fig. 7.** Illustration of autoregressive decoding (left) and speculative decoding (right).

higher the efficiency of the algorithm. Therefore, efficient drafting and verifying are key research points of speculative decoding.

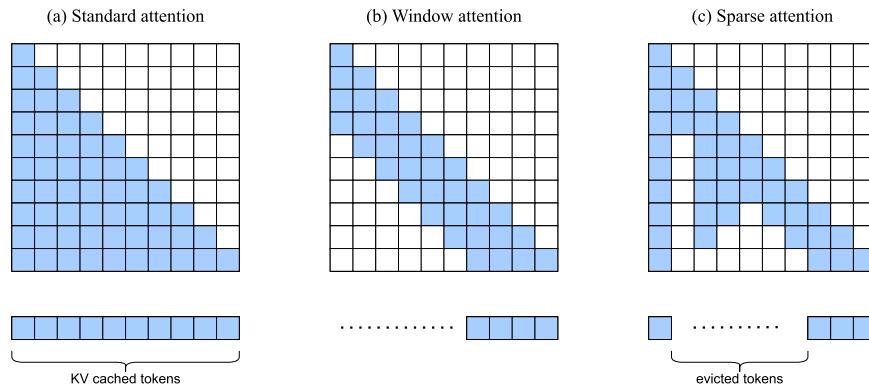
Studies [150,151] first proposed Speculative Sampling, which uses existing pre-trained models of different sizes and a modified sampling schema to achieve accurate decoding, resulting in more than a 2x inference acceleration. BiLD [152] proposed a fallback strategy and rollback strategy based on the prediction distribution to address the issue of when to stop the prediction of a small model and how to verify and correct the predicted tokens. SpecInfer [153] proposed a tree-based decoding mechanism and token tree verification mechanism, reducing memory accesses to parameters and the end-to-end inference latency. EAGLE [154] predicts the next feature at the feature level, uses a trained lightweight autoregressive head to generate tokens, and achieves higher efficiency through tree attention. Medusa [155] adopts multiple decoding heads on top of the last hidden state of an LLM, as well as tree attention, and predicts multiple subsequent tokens in parallel without the need for auxiliary models. Lookahead [156]

views autoregressive decoding as solving a nonlinear equation and uses Jacobi iteration method for parallel decoding.

In addition to the above studies, some research has begun applying this technology to edge devices to alleviate the memory and computational pressure of LLMs on these devices. LLMCad [157] designed a speculative decoding engine based on tree attention and a dynamic fallback strategy, significantly improving the generation speed of LLMs on IoT devices and smartphones. SpecExec [158] generates longer accepted token sequences through an improved token tree structure and designs an LLM parameter offloading system to support interactive LLMs on consumer devices.

#### 4.2. KV cache compression

KV cache represents a significant optimization technique in the inference process of LLMs, reducing redundant computations during the decoding stage by retaining previous keys and values in attention heads. The size of the KV cache linearly increases with the length of

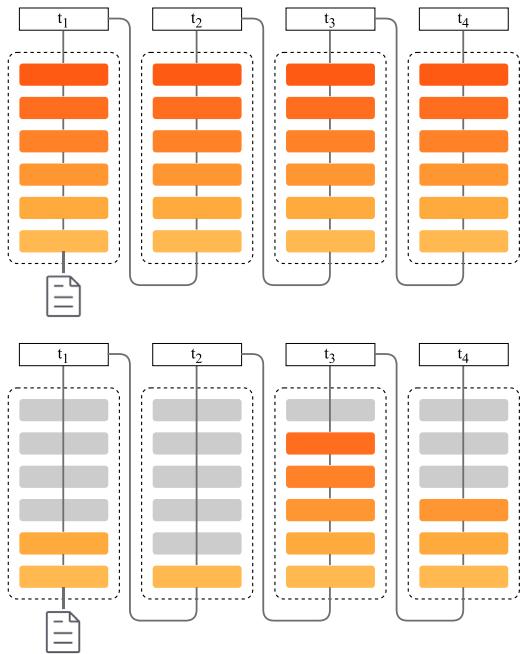


**Fig. 8.** Illustration of KV cache sparsity. (a) shows the full attention and key-value cache, while (b) and (c) depict the sparse attention patterns and corresponding key-value cache.

generated tokens, making memory a bottleneck in the inference process for long text generation. Therefore, researching KV cache compression methods is crucial to enabling efficient long-text inference [159]. Current KV cache compression can be classified into KV cache sparsity and KV cache quantization.

**KV Cache Sparsity** leverages sparse attention mechanisms to retain only the essential keys and values, thereby reducing the size of the KV cache. Fig. 8 illustrates the concept of KV cache sparsity in different attention patterns. FastGen [160] observed that attention heads in different layers exhibit different attention structures and applied different compression strategies based on these patterns. StreamingLLM [161] identifies that in autoregressive LLMs, a large amount of attention score is allocated to the initial few tokens, termed an “attention sink”. By preserving the initial tokens’ KV and employing windowed attention, it enables LLMs trained with a finite attention window to process text of infinite length. Scissorhands [162] proposed the “Persistence of Importance Hypothesis”, indicating that different tokens have varying importance for future predictions. Leveraging this observation, they proposed an inference algorithm using compressed KV cache, reducing memory usage of KV cache up to 5 $\times$  without degradation on model quality. H2O [163] introduced a KV cache eviction strategy using attention scores, evicting the key–value pair with the lowest cumulative attention score at each decoding step, thus retaining a balance between recent and important tokens.

**KV Cache Quantization** reduces the memory usage by quantizing keys and values, similar to weight quantization. However, directly quantizing the KV cache to low precision can lead to significant perplexity degradation [159]. Additionally, the streaming nature of KV cache may introduce additional computational overhead in calculating quantization parameters [164]. To address this issue, KIVI [164] suggests that the key cache should be quantized per-channel, while the value cache should be quantized per-token. Based on this conclusion, they proposed a hardware-friendly 2-bit quantization method, reducing the peak memory usage by 2.6 $\times$  while maintaining inference quality. KVQuant [159] conducted a more detailed evaluation of KV cache quantization performance, proposing a non-uniform ultra-low precision quantization method for KV cache. They calibrated the quantization parameters of the key cache using an offline method and those of the value cache using an online method. QAQ [165] proposes that key caches and value caches exhibit different sensitivities to quantization and designs a non-uniform mixed quantization strategy to preserve the precision of outlier attention values. IntactKV [166] observes that outliers in attention scores occur at specific tokens, referred to as pivot tokens. It proposes a quantization method using offline calibration and supports treating quantization parameters as additional trainable parameters for further calibration.



**Fig. 9.** Illustration of complete process (top) and early exiting process (bottom).

#### 4.3. Early exiting

Early exiting is a conditional computation technique that terminates generation early in the decoding process to reduce computation. Fig. 9 illustrates the complete and early exiting processes. CALM [167] introduces an early exit classifier in each layer of the decoder, predicting the confidence score of the current layer. Terminate the decoding early based on whether the confidence score exceeds the threshold. ConsistentEE [168] employs a policy network to make early exiting decisions, while also designing a “Memory Layer” metric incorporated into the reward function, allowing for a balance between accuracy and acceleration based on individual hardness. Unfortunately, there are conflicts between early exiting and KV cache, making it difficult for layers after the exit point to accelerate attention calculations [169]. A feasible method is to directly copy the hidden states of the current token at the exiting layer to all subsequent layers. However, this may lead to deviation and degrade output quality [170]. Several recent works aim to mitigate this issue, such as FREE [171], which introduces an early exiting framework incorporating a shallow-deep module and synchronized parallel decoding. It leverages parallel computing capabilities to calculate the KV cache for layers previously exited when

encountering a non-exited token, reducing redundant computations for subsequent tokens. Similarly, EE-LLM [170] proposes a pipeline parallelism relying on multiple devices, thereby filling the KV cache for layers that were early exited. SkipDecode [169] assumes that subsequent tokens are easier to predict, skipping intermediate layers starting from a fixed point. It ensures that the number of skipped layers for preceding tokens is not less than that for subsequent tokens, preventing the recalculation of KV cache. In addition to the above methods, some studies combine early exiting with speculative decoding to further enhance inference efficiency. For instance, LayerSkip [172] employs first few layers of an LLM as a drafting model and the complete model as a verifying model, combining early exiting and speculative decoding strategies. To encourage early exiting, LayerSkip applies layer dropout during training, with the dropout rate increasing as the model depth increases, thereby reducing the model's dependency on later layers.

#### 4.4. Kernel optimization

The main operators of transformers are attention and linear. Optimizing these two operators is crucial for improving the inference efficiency of LLMs. Common optimization methods involve reducing computation and memory access by leveraging software and hardware resources or reducing startup overhead through kernel fusion. FlashAttention [173] proposed an attention tile algorithm that utilizes the high-speed SRAM on GPUs, avoiding communication overhead caused by multiple memory copies through kernel fusion. FlashAttention-2 [174] improves upon the original by optimizing work partitioning among GPU threads, significantly reducing shared memory reads and writes. It further enhances parallelism by distributing attention computation across thread blocks and warps, thereby increasing occupancy and efficiency. FlashDecoding [175] introduces a novel parallelization dimension along the keys/values sequence length, fully utilizing the GPU even with small batch sizes and long contexts. It employs an iterative log-sum-exp reduction to maintain efficient memory usage, significantly reducing attention runtime regardless of sequence length. FlashDecoding++[176] introduces an asynchronous softmax with a unified max value to eliminate synchronization overheads, significantly enhancing attention computation efficiency. It optimizes flat GEMM operations with double buffering, improving computation utilization and reducing memory latency. Additionally, FlashDecoding++ implements a heuristic dataflow that adapts to hardware resources dynamically, achieving substantial speedups on both NVIDIA and AMD GPUs. PagedAttention [177] leverages an innovative attention algorithm inspired by virtual memory and paging, enabling efficient non-contiguous storage of key-value cache. This method reduces internal and external fragmentation, significantly improving memory utilization. Implemented in the vLLM serving system, PagedAttention achieves 2-4x throughput improvements over state-of-the-art systems by allowing flexible memory management and sharing of KV cache across requests. Flash-LLM [111] proposes a general Load-as-Sparse and Compute-as-Dense methodology for unstructured sparse matrix multiplication (SpMM) to address memory bandwidth bottlenecks in LLM inference. It also includes a pipeline design for SpMM that overlaps sparse data extraction, dense data loading and matrix computation. T-MAC [178] proposed a mixed precision matrix multiplication (mpGEMM) algorithm for weight-quantized LLMs. It reduces multiplications and additions by bit-wise table lookup, achieving higher throughput on lower-end edge devices.

#### 4.5. Memory offloading

Edge or consumer devices typically have limited memory, making it difficult to load the large parameters of LLMs into the RAM of these devices, rendering LLM inference infeasible. An effective method is to “offload” weights to external storage, loading partial weights into memory when needed, enabling devices to run models larger than their memory capacity. Some platforms and inference engines already

provide this feature, such as HuggingFace [179], Llama.cpp [180], etc. However, frequent memory swapping introduces significant communication overhead. To mitigate this drawback, recent studies have optimized memory offloading and loading strategies.

FlexGen [181] developed a linear programming-based search algorithm to optimize throughput within the search space, aiming to identify the optimal offloading strategy. Furthermore, it compresses both weights and attention caches to 4 bits, significantly enhancing the maximum throughput during LLM inference. Chen et al. [182] introduced the concept of attention offloading, which involves partitioning the attention mechanism into multiple subtasks, utilizing memory-optimized devices to compute the attention operator and computation-optimized devices for processing the remaining components of the model. Recent studies have demonstrated significant sparsity in neuron activations during LLM inference [183]. Leveraging this characteristic, some research has designed a memory offloading strategy based on sparsity. PowerInfer [184] discovered that LLM inference exhibits high locality, with some neurons, termed “hot-activated neurons”, being frequently activated. Based on this insight, PowerInfer designed a neuron-aware offloading strategy and inference engine, utilizing both GPU and CPU to store weights. It preloads the weights of frequently activated neurons for the GPU, while the weights of less active neurons are retained on the CPU. Addressing the challenge of designing memory offloading strategies for devices with limited memory, LLM in a flash [185] proposes a memory offloading strategy that utilizes DRAM and flash memory. Specifically, LLM in a flash leverages the concept of contextual sparsity introduced by DejaVu [183] to dynamically load neurons predicted to be active. It employs a sliding window technique to cache recently activated tokens and utilizes static memory pre-allocation to minimize loading latency. EdgeMoE [186] developed a memory offloading strategy specifically designed for MoE models. Leveraging the sparsity of the MoE architecture, non-expert weights are stored in memory, while expert weights are loaded from external storage only when activated, achieving memory savings on edge devices.

#### 4.6. Discussion

Inference optimization addresses runtime cost challenges by enhancing the forward propagation efficiency of models through systematic improvements. Current approaches, including speculative decoding, KV cache optimization, early exiting, kernel optimization, and sparsity exploitation, offer distinct tradeoffs between computational efficiency and accuracy, with their applicability depending on specific deployment constraints. These methods fall into two categories: lossless and lossy optimization. Lossless techniques (e.g., speculative decoding, kernel tuning, memory offloading) maximize hardware parallelism, optimize compute/memory access patterns, and reduce memory footprint without compromising model outputs. In contrast, lossy approaches like KV cache compression and early exiting trade minor accuracy degradation for significant reductions in FLOPs and memory usage.

Recent studies emphasize two strategic priorities: (1) hardware-aware algorithm design that leverages device-specific features, such as multi-core parallelism and memory hierarchies, and (2) dynamic sparsity exploitation during inference to reduce computational complexity. Emerging research also explores hybrid optimization frameworks, for instance, combining memory offloading with runtime sparsity [185], demonstrating how orthogonal techniques can complement each other. Future research directions may focus on exploring the synergies and upper bounds of such integrated optimization strategies.

### 5. Deployment

In this section, we explore frameworks designed for deploying and optimizing LLMs on edge devices. We categorize these frameworks into: on-device inference engines, cloud-edge collaborative frameworks and deployment suites. On-device inference engines refer to software

**Table 8**  
Frameworks designed for deploying and optimizing LLMs on edge devices.

Category	Features	Works
On-device Inference Engines	A suite of on-device optimizations, multi-platform compatibility, lightweight inference runtime, convenient development tools, and hardware acceleration capabilities	[157,184,191–196]
Cloud-edge Collaborative Frameworks	Combine cloud and edge computing, boost performance for complex tasks, leverage the strengths of both	[197–204]
Deployment Suites	Encompass a variety of LLM inference optimization functionalities and offer convenient development tools	[180,205–207]

infrastructures optimized for executing LLM inference tasks on edge devices. Cloud-edge collaborative frameworks aim to make full use of the resources of edge devices and the cloud for LLM. Deployment suites are more oriented toward providing software support and enhancing performance for LLM applications. We summarize the main features of these frameworks in Table 8. It is noteworthy that some well-known LLM inference frameworks [177,187–190] are primarily utilized for LLM serving. These have not been included in the discussion of this section, despite their capabilities for execution on edge devices.

### 5.1. On-device inference engines

An inference engine refers to software that combines inference optimization techniques to optimize resources for LLM inference. This category of frameworks includes some general-purpose deep learning inference frameworks for mobile and edge devices, such as ExecuTorch [191], TFLite [192], MNN [193], and NCNN [194]. These engines typically feature a suite of on-device optimizations, multi-platform compatibility, lightweight inference runtime, convenient development tools, and hardware acceleration capabilities. While these engines do not typically involve specialized optimizations for transformer architectures, their generality and flexibility make them suitable for a wide range of LLM models.

Another category of inference engines includes frameworks specifically designed for LLM inference. These frameworks are optimized for LLM inference tasks, incorporating one or more optimization techniques mentioned in Section 4, as well as other on-device optimizations, such as pipeline and efficient kernels. In addition to utilizing tree-based speculative decoding, LLMcad [157] employs a computing-loading pipeline to avoid contention for small and large LLM memory by profiling or user defining a memory upper bound. PowerInfer [184] employs the principle of locality in activations, comprising two stages: offline profiling and online inference. During the online phase, it also designs a sparse matrix multiplication operator tailored for its dynamic scenarios. Based on PowerInfer, PowerInfer-2 [195] introduces a dedicated inference framework for smartphones, leveraging the heterogeneous computational, memory, and I/O resources by decomposing traditional matrix computations into fine-grained computations of neuron clusters. Additionally, it incorporates a fine-grained pipeline to conceal I/O overhead. However, PowerInfer-2 necessitates models to exhibit strong predictable sparsity, which may limit its applicability to current mainstream LLMs. Transformer-Lite [196] is an LLM inference engine specifically designed for smartphones, proposing symbolic expression-based dynamic shape inference, low-precision matrix multiplication operators, and optimization of KV cache copying. The schematic representations of the four inference engine architectures are illustrated in Fig. 10.

### 5.2. Cloud-edge collaborative frameworks

Besides the aforementioned on-device LLM inference engines, another strategy is cloud-edge collaborative inference, which combines

the advantages of powerful cloud computing and edge-side computation. This approach holds promise as a significant pathway for LLM edge intelligence. For tasks that exceed the capabilities of edge devices, better performance can be achieved through edge-cloud collaboration. Previous studies [197–200] have employed model partitioning and offloading techniques to distribute computation between the cloud and edge, thereby reducing the computational burden on edge devices. However, due to the autoregressive nature of LLMs, direct model partitioning results in significant communication overhead, making it difficult to apply in LLM applications. Recent studies consider the trade-off between cost and quality for edge-cloud collaboration in resource-constrained scenarios. The comparative analysis of core features and application scenarios across representative cloud-edge collaborative frameworks is systematically presented in Table 9.

Hybrid LLM [201] involves a hybrid inference strategy that considers the trade-off between cost and quality. A router is utilized to allocate queries between a large model (deployed in the cloud) and a small model (suitable for edge devices). The router's decision is based on the predicted query difficulty and the desired quality level, which can be dynamically adjusted at test time. This allows for seamless adaptation to different scenarios and requirements. Different router designs, such as deterministic and probabilistic routers with and without data transformation, are explored to handle various situations, especially when dealing with the disparity in performance between the cloud-based large model and the edge-based small model.

Tabi [202] is an efficient multi-level inference system. Tabi features a multi-level inference engine that serves queries using small models and optional LLMs for demanding applications. It decides whether to use small models or LLMs based on calibrated confidence scores. For queries routed to LLMs, it employs attention-based word pruning, using the attention weights from small models to reduce input data size and offset latency overhead. It also uses a weighted ensemble technique to combine predictions from different levels and improve accuracy.

Edge-LLM [203] is a collaborative framework for large language model (LLM) serving in edge computing. The framework aims to address the challenges of efficient deployment and fine-tuning of adapter models for LLM tasks in resource-constrained scenarios. It employs a server-node collaboration approach, where the LLM backbone is deployed on the edge server and the adapter on the edge nodes. This design reduces the computational pressure on the edge server and enables real-time inference and online fine-tuning on the edge nodes.

FedAgg [204] is a framework for End-Edge-Cloud Collaboration in Federated Learning (FL). FedAgg overcomes the model scale limitations of prior Hierarchical Federated Learning (HFL) methods by using a customized Bridge Sample Based Online Distillation Protocol (BSBODP). In BSBODP, each computing node (except leaf nodes with an added encoder) has a pre-trained decoder and a model. Bridge samples, generated from embeddings related to private data, facilitate online distillation between parent-child nodes, enabling knowledge transfer and growth of models in size and generalization ability from end to cloud, while maintaining privacy and flexibility.

Dynamic token-level edge-cloud collaboration framework [208] involves identifying “harder” tokens through various indicators such

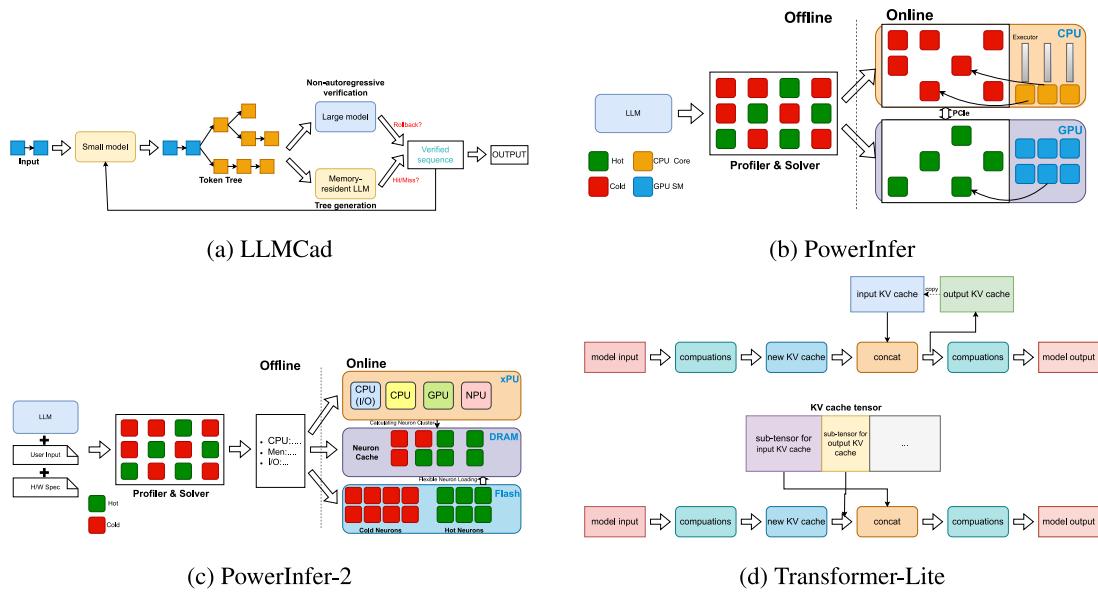


Fig. 10. The framework diagrams of four on-device Inference Engines.

**Table 9**

Frameworks designed for Cloud-Edge collaborative frameworks.

Framework	Core Features	Application Scenarios
Hybrid LLM	Routers allocate queries to large models or small models based on query difficulty and desired quality level to balance quality and cost.	It focuses on dynamically allocating models according to query difficulty and quality requirements in natural language processing tasks, optimizing reasoning costs.
Tabi	A multi-layer reasoning engine is designed. It uses calibrated confidence scores to decide whether to quickly return the accurate results of the small model or re-route to the LLM.	It is mainly applied to text processing tasks with high requirements for accuracy and latency, such as text classification and natural language inference.
Edge-LLM	An adaptive quantization strategy, an FM caching mechanism, and a VDF scheduling algorithm are implemented.	It focuses on the edge computing environment, emphasizing the collaboration between the edge and the server and resource utilization.
FedAgg	Based on the BSBODP, model-agnostic collaborative training among interactive nodes is achieved through online distillation on the generated bridge samples.	It is applied to the device-edge-cloud collaboration scenarios of federated learning, mainly used for distributed model training that requires data privacy protection.
Dynamic token-level edge-cloud collaboration framework	Utilizing SLMs on edge devices to interact with cloud-based LLMs at the token level, calling LLMs for verification and correction of “difficult” tokens according to the budget.	It is primarily applied to natural language generation applications on mobile devices, focusing on improving reasoning quality and controlling costs under limited resources.

as Top-1 probabilities or entropy of SLMs and the LLM’s probability distribution on SLM — generated tokens. A draft — verification method is then employed, where tokens with LLM probability above a threshold are from the SLM, and those below are from the LLM. The SLM generates tokens, and the LLM verifies and corrects “harder” tokens, achieving a balance between inference quality and cost.

### 5.3. Deployment suites

Deployment Suites constitute a category of software infrastructure that is closer to users and developers. Typically, these tools encompass a variety of LLM inference optimization functionalities and offer convenient development tools. A comprehensive comparison of the features of the four Deployment Suites is presented in Table 10.

MLC-LLM [205] is a general-purpose high-performance LLM local deployment framework that leverages deep learning compilation techniques to provide better optimization and cross-platform portability. MLC-LLM supports a variety of hardware platforms, ranging from smartphones to dedicated edge computing devices, offering great convenience for LLM edge deployment. llama.cpp [180] is a tool for local

deployment of LLMs based on pure C++ language. It provides a simple C++ API, supporting the deployment of LLM models on edge devices with various CPU architectures. It also offers optimization techniques, such as model quantization. The core of llama.cpp is GGML [209], an LLM tensor library for edge devices, endowing llama.cpp with strong portability. mllm [206] is a fast and lightweight multimodal LLM inference engine for mobile and edge devices, implemented in C++ and optimized for multiple CPU architectures. It supports various multimodal LLM models and provides a simple API for developers to deploy multimodal LLM models on edge devices. NanoLLM [207] is a local inference framework for LLMs on the NVIDIA Jetson platform. It offers some simple LLM optimization techniques and APIs, providing convenience for LLM inference on the Jetson platform.

## 6. Open challenges and future directions

As LLMs continue to advance and proliferate, their deployment on edge devices presents numerous opportunities and challenges. The potential to bring powerful AI capabilities closer to the user promises significant benefits, including reduced latency, enhanced privacy, and

**Table 10**  
Comparison of Different Deployment Suites.

	MLC-LLM	llama.cpp	mllm	NanoLLM
Applicable Model Architectures	Widely supported (Transformer, RWKV, custom architectures)	Primarily Llama series (Llama 1/2/3, Alpaca, etc.)	Lightweight models (TinyLLM, MobileBERT, etc.)	Edge-optimized architectures (TFLite converted models, NanoGPT, etc.)
Hardware Compatibility	All platforms (CPU/GPU/NPU, Android/iOS, WebGPU)	CPU first (ARM/x86), GPU (CUDA/Metal)	Mobile devices only (Android/iOS NPU acceleration)	Edge devices (Raspberry Pi, Jetson, MCU embedded)
Quantization Support	Flexible quantization (INT4/INT8/FP16)	GGUF format	Lightweight (FP16, 8-bit quantization)	Efficient quantization (INT8/INT4)
Distributed Inference	Has certain distributed inference capabilities	Multi-thread optimization	Only on single device	No distributed support
Edge Deployment Optimization	Medium	Low	High	Extremely high
Multi-modal Support	Supports multi-modal large models	Vision-text models	Vision-text models	Vision-text models

the ability to operate in environments with limited connectivity. However, the transition from cloud-based to edge-based LLM deployment is fraught with obstacles that must be carefully navigated. These challenges range from resource constraints and dynamic edge environments to privacy concerns and ethical considerations. Addressing these issues is critical to fully realizing the potential of LLMs in edge intelligence. Below, we delve into some of the key challenges and future directions in this field.

#### 6.1. Resource constraints on edge devices

Edge devices typically lack the performance capabilities of specialized accelerators. Due to the large number of parameters and high demands for computational and storage resources, LLMs face constraints in terms of storage space, computational power, bandwidth, and energy on edge devices. Traditional lightweight model structures and compression techniques can alleviate this issue to some extent, but they inevitably compromise the performance of LLMs. To address this problem, researchers need to explore more advanced compression and acceleration techniques and design inference engines that fully utilize the hardware of edge devices to improve the efficiency of large model inference in resource-constrained environments. Additionally, designing efficient model architectures is a crucial direction for future research. Recent studies, such as those on MoE [67,210], RWKV [211], Mamba [212], and RetNet [213], have demonstrated advantages over the classic Transformer [25] architecture. Researching resource-efficient model architectures, particularly those that can adapt to the software and hardware environments of edge devices, can further reduce the deployment and application costs of LLMs.

#### 6.2. Dynamicity of edge environments

Edge dynamicity encompasses the variability in edge device resources, network conditions, and user demands, posing challenges to the portability and efficiency of deploying and inferring LLMs at the edge. Significant differences in computational and storage resources across devices, coupled with fluctuations in network conditions over time, make it difficult to provide consistent and stable services with LLMs at the edge. Additionally, user requirements for model inference latency and accuracy can vary greatly, further complicating the deployment and inference of LLMs at the edge. Key research directions include modeling the dynamic nature of edge scenarios and developing scheduling mechanisms, collaboration strategies, and adaptive algorithms that can adjust to changes in resources and demands. Notably, in certain LLM deployment scenarios, such as on smartphones, models cannot fully monopolize device resources and may even have lower scheduling priorities. Research on LLMs as system services [214] has already begun and represents an important future direction.

#### 6.3. Heterogeneity of edge devices

The heterogeneity of edge devices, including differences in hardware configurations, functionalities, and operating systems, presents compatibility challenges for the acceleration and deployment of LLMs. While some progress [215–218] has been made in serving LLMs on heterogeneous computing platforms, the complexity of heterogeneity in edge environments is even greater. This is particularly true for different hardware such as SoCs, FPGAs, ASICs, and TPUs, where significant differences in performance and instruction sets pose open questions regarding the portability of LLMs across various platforms. Therefore, it is necessary to model a unified LLM computation paradigm that provides operator compatibility across different platforms. Additionally, exploring adaptive algorithms and flexible deployment frameworks is essential to accommodate diverse edge hardware. Hardware-aware model compression and acceleration techniques can also help optimize on-device inference performance.

#### 6.4. Edge-cloud collaboration

For tasks that exceed the capabilities of edge devices, better performance can be achieved through edge-cloud collaboration. Previous studies [197–200] have employed model partitioning and offloading techniques to distribute computation between the cloud and edge, thereby reducing the computational burden on edge devices. However, due to the autoregressive nature of LLMs, direct model partitioning results in significant communication overhead, making it difficult to apply in LLM applications. Another approach is task allocation based on task difficulty, known as Query Routing. Recent studies [201,202] have designed hybrid inference systems using this method. Designing hybrid inference systems that adaptively allocate queries of varying difficulty between the cloud and edge is a promising research direction.

#### 6.5. Privacy and security concerns

Deploying LLMs on edge devices raises significant privacy and security concerns. Sensitive data processed on edge devices can be vulnerable to breaches, making it essential to develop robust encryption methods, secure model execution environments, and privacy-preserving machine learning techniques. Ensuring data privacy while maintaining model performance is a critical challenge. Additionally, securing the models themselves against adversarial attacks and unauthorized access is paramount to protect both user data and the integrity of the models.

## 6.6. Hallucination issues

Gunasekar et al. [21] suggests that high-quality data can enable even small-sized LLMs to achieve high performance, inspiring many subsequent studies. However, small language models, due to their limited number of parameters, often exhibit significant performance variability, with hallucination issues being particularly prominent. Recent study [219] indicate that small language models suffer from severe overfitting problems, which hinder their application at the edge. Exploring techniques to mitigate overfitting and reduce hallucination in small language models is crucial for enhancing their reliability and usability in edge scenarios. Potential solutions may involve advanced regularization techniques, improved training methods, and hybrid approaches that combine the strengths of both small and large models.

## 6.7. Morality and law

As the expansion of LLMs to edge environments becomes increasingly prevalent, the significance of ethical and legal challenges becomes more pronounced. We anticipate that the behavior of LLMs aligns with human values, objectives, and expectations, ensuring their adherence to ethical standards and legal regulations, especially in sensitive domains such as healthcare and finance. However, deploying LLMs on edge devices typically requires techniques such as pruning and distillation to reduce the model's size and computational demands. These methods, however, may inadvertently impair the model's ability to adhere to ethical guidelines, potentially leading to biased or harmful outputs. Furthermore, the decentralized nature of edge computing complicates the attribution of responsibility. Determining responsibility becomes extremely challenging when LLMs generate inappropriate content or violate privacy regulations.

## 6.8. Interpretability

As LLMs become increasingly complex, comprehending their decision-making processes becomes more challenging. The obscurity of their internal mechanisms and lack of transparency introduce unnecessary risks to downstream applications, particularly in critical domains such as healthcare where a deep understanding of how decisions are made is essential. To deploy large models on edge devices, optimization techniques such as pruning or distillation are commonly employed. While these methods effectively reduce the model's size and computational requirements, they may lead to a simplification of the model's structure, thereby diminishing its interpretability. The reduction of intermediate layers and complexity could result in the loss of information crucial for explaining the model's behavior. Consequently, ensuring the interpretability of models while maintaining performance on edge devices is one of the pressing challenges that future research must address.

## 6.9. User preferences and personalization

The deployment of LLMs on edge devices naturally increases the demand for personalized interactions due to their proximity to users and the environments in which they operate. Additionally, edge devices can collect a wealth of personalized data, such as location, device usage patterns, and user behavior, enabling models to tailor responses more effectively to individual needs. Research has already begun to focus on the personalization of large models [220], Doddapaneni et al. [221] introduced a User Embedding Model (UEM) that compresses user history into embedding vectors as soft prompts for language models, significantly enhancing the personalization accuracy of recommendation systems. Developing the capability for LLMs to learn and adapt in real-time from user interactions, ensuring that models can adjust to personal preferences while safeguarding data privacy, is a promising

avenue for future research. By incorporating controllable mechanisms for knowledge retention and forgetting, as well as continuous learning mechanisms, models can autonomously learn new skills and improve existing capabilities based on user interactions and local data. With this capability, edge LLMs can leverage on-device learning for personalized content delivery and enhanced user experiences, while also minimizing latency.

## 6.10. Green and sustainable development

Training and processing of LLMs require substantial data and significant computational resources. Consequently, LLMs are typically trained and deployed on resource-rich cloud servers. However, as large models increasingly migrate to wireless edge networks closer to end-user devices, edge AI faces considerable challenges. Operating resource-intensive models on battery-powered devices can rapidly deplete battery life. STI [222] employs an elastic pipeline planning with a preloaded buffer, effectively balancing latency and memory usage on mobile devices, thereby accelerating the NLP inference process on edge devices. This acceleration maximizes the utilization of computational resources. Recently, Yuan et al. [223] have proposed a novel mobile AI paradigm that integrates a foundational model on the NPU of mobile devices to achieve unified management and optimization for a variety of mobile AI tasks, and enhances efficiency and performance by providing short, offline fine-tuned adapters for different applications.

## 7. Conclusion

The transition of LLMs to the edge is an inevitable trend in the future and has made preliminary research progress. However, overcoming many technical challenges is still required to achieve this goal. Our survey comprehensively investigates the current state of research on LLM applications at the edge, covering all aspects from model design, optimization to deployment, and pointing out future research directions and challenges. The survey first emphasizes the difficulties faced by the edge side of LLMs, then summarizes and compares the comprehensive details of the current mainstream SLMs, discusses the current performance and application status and prospects of SLMs at the edge. We then thoroughly investigate related technologies for advanced edge-side acceleration, including model compression and inference optimization. The former optimizes the model architecture and weights at the model level before inference, including pruning, quantization, distillation, and low-rank decomposition. The latter optimizes the computation and storage of the model during inference runtime, including speculative decoding, KV cache compression, early exiting, kernel optimization, and memory offloading. We also discuss the limitations of current research methods, elucidate the development trends and future research directions of related technologies. Next, we investigate the development of model edge-side deployment frameworks, summarize the characteristics of the current mainstream frameworks, discuss the design principles of the frameworks and future development directions. Finally, we summarize the current state of research, emphasizing the importance and challenges of LLM edge-side applications, and looking forward to future research directions and development trends. We believe that edge intelligence enabled by LLMs will be more widely used in the future and hope that this survey will reveal the challenges and provide valuable insights into edge intelligence enabled by LLMs for researchers and practitioners alike.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

No data was used for the research described in the article.

## References

- [1] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, Improving Language Understanding by Generative Pre-Training, Technical Report, OpenAI, 2018.
- [2] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 4171–4186.
- [3] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F.L. Aleman, D. Almeida, J. AltenSchmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, et al., GPT-4 technical report, 2024, [arXiv:2303.08774](https://arxiv.org/abs/2303.08774).
- [4] B. Workshop, T.L. Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A.S. Luccioni, F. Yvon, M. Gallé, J. Tow, A.M. Rush, S. Biderman, A. Webson, P.S. Ammanamanchi, T. Wang, B. Sagot, N. Muennighoff, A.V. del Moral, et al., BLOOM: A 176B-parameter open-access multilingual language model, 2023, [arXiv:2211.05100](https://arxiv.org/abs/2211.05100).
- [5] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambrø, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, G. Lample, LLaMA: Open and efficient foundation language models, 2023, [arXiv:2302.13971](https://arxiv.org/abs/2302.13971).
- [6] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C.C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, et al., Llama 2: Open foundation and fine-tuned chat models, 2023, [arXiv:2307.09288](https://arxiv.org/abs/2307.09288).
- [7] N. Dhar, B. Deng, D. Lo, X. Wu, L. Zhao, K. Suo, An empirical analysis and resource footprint study of deploying large language models on edge devices, in: Proceedings of the 2024 ACM Southeast Conference, in: ACM SE '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 69–76.
- [8] X. Li, Z. Lu, D. Cai, X. Ma, M. Xu, Large language models on mobile devices: Measurements, analysis, and insights, in: Proceedings of the Workshop on Edge and Mobile Foundation Models, in: EdgeFM '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 1–6.
- [9] M. Xu, H. Du, D. Niyato, J. Kang, Z. Xiong, S. Mao, Z. Han, A. Jamalipour, D.I. Kim, X. Shen, V.C.M. Leung, H.V. Poor, Unleashing the power of edge-cloud generative AI in mobile networks: A Survey of AIGC services, IEEE Commun. Surv. & Tutorials 26 (2) (2024) 1127–1170.
- [10] L.S. Vailshery, IoT connections worldwide 2022–2033, 2024, URL: <https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/>. (Accessed: Jul. 26, 2024).
- [11] Z. Wan, X. Wang, C. Liu, S. Alam, Y. Zheng, J. Liu, Z. Qu, S. Yan, Y. Zhu, Q. Zhang, M. Chowdhury, M. Zhang, Efficient large language models: A survey, Trans. Mach. Learn. Res. (2024).
- [12] Z. Zhou, X. Ning, K. Hong, T. Fu, J. Xu, S. Li, Y. Lou, L. Wang, Z. Yuan, X. Li, S. Yan, G. Dai, X.-P. Zhang, Y. Dong, Y. Wang, A survey on efficient inference for large language models, 2024, [arXiv:2404.14294](https://arxiv.org/abs/2404.14294).
- [13] M. Xu, W. Yin, D. Cai, R. Yi, D. Xu, Q. Wang, B. Wu, Y. Zhao, C. Yang, S. Wang, Q. Zhang, Z. Lu, L. Zhang, S. Wang, Y. Li, Y. Liu, X. Jin, X. Liu, A survey of resource-efficient LLM and multimodal foundation models, 2024, [arXiv:2401.08092](https://arxiv.org/abs/2401.08092).
- [14] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, H. Jin, T. Chen, Z. Jia, Towards efficient generative large language model serving: A survey from algorithms to systems, 2023, [arXiv:2312.15234](https://arxiv.org/abs/2312.15234).
- [15] T. Ding, T. Chen, H. Zhu, J. Jiang, Y. Zhong, J. Zhou, G. Wang, Z. Zhu, I. Zharkov, L. Liang, The efficiency spectrum of large language models: An algorithmic survey, 2024, [arXiv:2312.00678](https://arxiv.org/abs/2312.00678).
- [16] G. Bai, Z. Chai, C. Ling, S. Wang, J. Lu, N. Zhang, T. Shi, Z. Yu, M. Zhu, Y. Zhang, C. Yang, Y. Cheng, L. Zhao, Beyond efficiency: A systematic survey of resource-efficient large language models, 2024, [arXiv:2401.00625](https://arxiv.org/abs/2401.00625).
- [17] J. Xu, Z. Li, W. Chen, Q. Wang, X. Gao, Q. Cai, Z. Ling, On-device language models: A comprehensive review, 2024, [arXiv:2409.00088](https://arxiv.org/abs/2409.00088).
- [18] Z. Lu, X. Li, D. Cai, R. Yi, F. Liu, X. Zhang, N.D. Lane, M. Xu, Small language models: Survey, measurements, and insights, 2024, [arXiv:2409.15790](https://arxiv.org/abs/2409.15790).
- [19] F. Wang, Z. Zhang, X. Zhang, Z. Wu, T. Mo, Q. Lu, W. Wang, R. Li, J. Xu, X. Tang, Q. He, Y. Ma, M. Huang, S. Wang, A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with LLMs, and trustworthiness, 2024, [arXiv:2411.03350](https://arxiv.org/abs/2411.03350).
- [20] C. Fourrier, N. Habib, A. Lozovskaya, K. Szafer, T. Wolf, Open LLM Leaderboard V2, Hugging Face, 2024, URL: [https://huggingface.co/spaces/open-llm-leaderboard/open\\_llm\\_leaderboard](https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard).
- [21] S. Gunasekar, Y. Zhang, J. Aneja, C.C.T. Mendes, A. Del Giorno, S. Gopi, M. Javaheripi, P. Kauffmann, G. de Rosa, O. Saarikivi, A. Salim, S. Shah, H.S. Behl, X. Wang, S. Bubeck, R. Eldan, A.T. Kalai, Y.T. Lee, Y. Li, Textbooks are all you need, 2023, [arXiv:2306.11644](https://arxiv.org/abs/2306.11644).
- [22] S. Hu, Y. Tu, X. Han, C. He, G. Cui, X. Long, Z. Zheng, Y. Fang, Y. Huang, W. Zhao, X. Zhang, Z.L. Thai, K. Zhang, C. Wang, Y. Yao, C. Zhao, J. Zhou, J. Cai, Z. Zhai, N. Ding, et al., MiniCPM: Unveiling the potential of small language models with scalable training strategies, 2024, [arXiv:2404.06395](https://arxiv.org/abs/2404.06395).
- [23] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebron, S. Shanghai, GQA: Training generalized multi-query transformer models from multi-head checkpoints, in: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Singapore, 2023, pp. 4895–4901.
- [24] N. Shazeer, Fast transformer decoding: One write-head is all you need, 2019, [arXiv:1911.02150](https://arxiv.org/abs/1911.02150).
- [25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, vol. 30, Curran Associates, Inc., 2017.
- [26] O. Press, L. Wolf, Using the output embedding to improve language models, in: M. Lapata, P. Blunsom, A. Koller (Eds.), Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers, Association for Computational Linguistics, Valencia, Spain, 2017, pp. 157–163, URL: <https://aclanthology.org/E17-2025>.
- [27] Z. Liu, C. Zhao, F. Iandola, C. Lai, Y. Tian, I. Fedorov, Y. Xiong, E. Chang, Y. Shi, R. Krishnamoorthi, L. Lai, V. Chandra, MobileLLM: Optimizing sub-billion parameter language models for on-device use cases, 2024, [arXiv:2402.14905](https://arxiv.org/abs/2402.14905).
- [28] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E.H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, W. Fedus, Emergent abilities of large language models, Trans. Mach. Learn. Res. (2022) URL: <https://openreview.net/forum?id=yzkSU5zdwd>.
- [29] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, ROBERTa: A robustly optimized BERT pretraining approach, 2019, [arXiv:1907.11692](https://arxiv.org/abs/1907.11692).
- [30] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, C. Leahy, The pile: An 800GB dataset of diverse text for language modeling, 2020, [arXiv:2101.00027](https://arxiv.org/abs/2101.00027).
- [31] E.M. Bender, T. Gebru, A. McMillan-Major, S. Shmitchell, On the dangers of stochastic parrots: Can language models be too big? in: Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, in: FAccT '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 610–623.
- [32] H. Laurençon, L. Saulnier, T. Wang, C. Akiki, A.V. del Moral, T. Le Scao, L. von Werra, C. Mou, E.G. Ponferrada, H. Nguyen, J. Frohberg, M. Šáško, Q. Lhoest, A. McMillan-Major, G. Dupont, S. Biderman, A. Rogers, L. Benallal, F. De Toni, G. Pistilli, et al., The BigScience ROOTS corpus: a 1.6TB composite multilingual dataset, in: Proceedings of the 36th International Conference on Neural Information Processing Systems, in: NIPS '22, Curran Associates Inc., Red Hook, NY, USA, 2024, pp. 31809–31826.
- [33] D. Soboleva, SlimPajama: A 627B token, cleaned and deduplicated version of RedPajama, 2023, URL: <https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama/>. (Accessed: Jul. 26, 2024).
- [34] R. Li, L.B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, Q. Liu, E. Zheltonozhskii, T.Y. Zhuo, T. Wang, O. Dehaene, J. Lamy-Poirier, J. Monteiro, N. Gontier, M.-H. Yee, L.K. Umapathi, et al., StarCoder: may the source be with you!, Trans. Mach. Learn. Res. (2023).
- [35] M. Bellagente, J. Tow, D. Mahan, D. Phung, M. Zhuravinskyi, R. Adithyan, J. Baicoianu, B. Brooks, N. Cooper, A. Datta, M. Lee, E. Mostaque, M. Pieler, N. Pinnaparaju, P. Rocha, H. Saini, H. Teufel, N. Zanicelli, C. Riquelme, Stable LM 2 1.6B technical report, 2024, [arXiv:2402.17834](https://arxiv.org/abs/2402.17834).
- [36] G. Penedo, Q. Malartic, D. Hesslow, R. Cojocaru, H. Alobeidli, A. Cappelli, B. Pannier, E. Almazrouei, J. Launay, The RefinedWeb dataset for falcon LLM: Outperforming curated corpora with web data only, Adv. Neural Inf. Process. Syst. 36 (2023) 79155–79172.
- [37] T. Computer, RedPajama: an open dataset for training large language models, 2024, URL: <https://github.com/togethercomputer/RedPajama-Data>. (Accessed: Mar. 30, 2024).
- [38] L. Soldaini, R. Kinney, A. Bhagia, D. Schwenk, D. Atkinson, R. Author, B. Bogin, K. Chandu, J. Dumas, Y. Elazar, V. Hofmann, A.H. Jha, S. Kumar, L. Lucy, X. Lyu, N. Lambert, I. Magnusson, J. Morrison, N. Muennighoff, A. Naik, et al., Dolma: an open corpus of three trillion tokens for language model pretraining research, 2024, [arXiv:2402.00159](https://arxiv.org/abs/2402.00159).
- [39] Common crawl - Open repository of web crawl data, 2024, URL: <https://commoncrawl.org/>. (Accessed: Jul. 26, 2024).
- [40] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P.J. Liu, Exploring the limits of transfer learning with a unified text-to-text transformer, J. Mach. Learn. Res. 21 (140) (2020) 1–67.
- [41] S. Yuan, H. Zhao, Z. Du, M. Ding, X. Liu, Y. Cen, X. Zou, Z. Yang, J. Tang, WuDaoCorpus: A super large-scale Chinese corpora for pre-training language models, AI Open 2 (2021) 65–68.

- [42] FudanNLPLAB/CBook-150K, 2023, URL: <https://github.com/FudanNLPLAB/CBook-150K>, (Accessed: Jul. 26, 2024).
- [43] L.B. Allal, A. Lozhkov, E. Bakouch, L. von Werra, T. Wolf, SmollM - Blazingly fast and remarkably powerful, 2024, URL: <https://huggingface.co/blog/smollm>.
- [44] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X.V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P.S. Koura, A. Sridhar, T. Wang, L. Zettlemoyer, OPT: Open pre-trained transformer language models, 2022, [arXiv:2205.01068](https://arxiv.org/abs/2205.01068).
- [45] T. Brown, B. Mann, N. Ryder, M. Subbiah, J.D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, et al., Language models are few-shot learners, in: Advances in Neural Information Processing Systems, vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.
- [46] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, H. Wu, Mixed precision training, 2018, [arXiv:1710.03740](https://arxiv.org/abs/1710.03740).
- [47] Z. Du, Y. Qian, X. Liu, M. Ding, J. Qiu, Z. Yang, J. Tang, GLM: General language model pretraining with autoregressive blank infilling, in: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Dublin, Ireland, 2022, pp. 320–335.
- [48] T. GLM, A. Zeng, B. Xu, B. Wang, C. Zhang, D. Yin, D. Rojas, G. Feng, H. Zhao, H. Lai, H. Yu, H. Wang, J. Sun, J. Zhang, J. Cheng, J. Gui, J. Tang, J. Zhang, J. Li, L. Zhao, et al., ChatGLM: A family of large language models from GLM-130B to GLM-4 all tools, 2024, [arXiv:2406.12793](https://arxiv.org/abs/2406.12793).
- [49] Z. Cai, M. Cao, H. Chen, K. Chen, X. Chen, Z. Chen, Z. Chen, P. Chu, X. Dong, H. Duan, Q. Fan, Z. Fei, Y. Gao, J. Ge, C. Gu, Y. Gu, T. Gui, A. Guo, et al., InternLM2 technical report, 2024, [arXiv:2403.17297](https://arxiv.org/abs/2403.17297).
- [50] InternLM, Internlm/Internlm2\_5.1\_8b, 2024, URL: <https://huggingface.co/internlm/internlm2.5-1.8b>.
- [51] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, B. Hui, L. Ji, M. Li, J. Lin, R. Lin, D. Liu, G. Liu, C. Lu, K. Lu, J. Ma, et al., Qwen technical report, 2023, [arXiv:2309.16609](https://arxiv.org/abs/2309.16609).
- [52] Introducing Qwen1.5, 2024, URL: <http://qwenlm.github.io/blog/qwen1.5/>. (Accessed: Jun. 18, 2024).
- [53] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang, G. Dong, H. Wei, H. Lin, J. Tang, J. Wang, J. Yang, J. Tu, J. Zhang, J. Ma, J. Yang, J. Xu, J. Zhou, J. Bai, J. He, J. Lin, K. Dang, K. Lu, K. Chen, K. Yang, M. Li, M. Xue, N. Ni, P. Zhang, P. Wang, R. Peng, R. Men, R. Gao, R. Lin, S. Wang, S. Bai, S. Tan, T. Zhu, T. Li, T. Liu, W. Ge, X. Deng, X. Zhou, X. Ren, X. Zhang, X. Wei, X. Ren, X. Liu, Y. Fan, Y. Yao, Y. Zhang, Y. Wan, Y. Chu, Y. Liu, Z. Cui, Z. Zhang, Z. Guo, Z. Fan, Qwen2 technical report, 2024, [http://dx.doi.org/10.48550/arXiv.2407.10671](https://dx.doi.org/10.48550/arXiv.2407.10671), [arXiv:2407.10671](https://arxiv.org/abs/2407.10671).
- [54] Qwen Team, Qwen2.5: A party of foundation models!, 2024, Qwen, URL: <http://qwenlm.github.io/blog/qwen2.5/>.
- [55] Y. Li, S. Bubeck, R. Eldan, A. Del Giorno, S. Gunasekar, Y.T. Lee, Textbooks are all you need II: phi-1.5 technical report, 2023, [arXiv:2309.05463](https://arxiv.org/abs/2309.05463).
- [56] A. Hughes, Phi-2: The surprising power of small language models, 2023, URL: <https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/>, (Accessed: Jun. 18 2024).
- [57] M. Abdin, S.A. Jacobs, A.A. Awan, J. Aneja, A. Awadallah, H. Awadalla, N. Bach, A. Bahree, A. Bakhtiari, J. Bao, H. Behl, A. Benhaim, M. Bilenko, J. Bjorck, S. Bubeck, Q. Cai, M. Cai, C.C.T. Mendes, W. Chen, V. Chaudhary, et al., Phi-3 technical report: A highly capable language model locally on your phone, 2024, [arXiv:2404.14219](https://arxiv.org/abs/2404.14219).
- [58] G. Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M.S. Kale, J. Love, P. Tafti, L. Huszenot, P.G. Sessa, A. Chowdhery, A. Roberts, A. Barua, A. Botev, A. Castro-Ros, A. Slone, A. Hélio, et al., Gemma: Open models based on gemini research and technology, 2024, [arXiv:2403.08295](https://arxiv.org/abs/2403.08295).
- [59] P. Zhang, G. Zeng, T. Wang, W. Lu, TinyLlama: An open-source small language model, 2024, [arXiv:2401.02385](https://arxiv.org/abs/2401.02385).
- [60] S. Biderman, H. Schoelkopf, Q.G. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M.A. Khan, S. Purohit, U.S. Prashanth, E. Raff, A. Skowron, L. Sutawika, O.V.D. Wal, Pythia: A suite for analyzing large language models across training and scaling, in: Proceedings of the 40th International Conference on Machine Learning, PMLR, 2023, pp. 2397–2430.
- [61] O. Thawakar, A. Vayani, S. Khan, H. Cholakal, R.M. Anwer, M. Felsberg, T. Baldwin, E.P. Xing, F.S. Khan, MobiLLama: Towards accurate and lightweight fully transparent GPT, 2024, [arXiv:2402.16840](https://arxiv.org/abs/2402.16840).
- [62] D. Groeneveld, I. Beltagy, P. Walsh, A. Bhagia, R. Kinney, O. Tafjord, A.H. Jha, H. Ivison, I. Magnusson, Y. Wang, S. Arora, D. Atkinson, R. Author, K.R. Chandu, A. Cohan, J. Dumas, Y. Elazar, Y. Gu, J. Hessel, T. Khot, et al., OLMo: Accelerating the science of language models, 2024, [arXiv:2402.00838](https://arxiv.org/abs/2402.00838).
- [63] OpenBMB, Openbmb/MiniCPM3-4B, 2024, URL: <https://huggingface.co/openbmb/MiniCPM3-4B>.
- [64] S. Mehta, M.H. Sekhavat, Q. Cao, M. Horton, Y. Jin, C. Sun, I. Mirzadeh, M. Najibi, D. Belenko, P. Zatoulkal, M. Rastegari, OpenELM: An efficient language model family with open training and inference framework, 2024, [arXiv:2404.14619](https://arxiv.org/abs/2404.14619).
- [65] B. Zhang, R. Sennrich, Root mean square layer normalization, in: Advances in Neural Information Processing Systems, vol. 32, Curran Associates, Inc., 2019.
- [66] L.B. Allal, A. Lozhkov, E. Bakouch, L. von Werra, T. Wolf, SmollM2 - with great data, comes great performance, 2024, URL: <https://huggingface.co/HuggingFaceTB/SmollM2-1.7B>.
- [67] A.Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D.S. Chaplot, D. de las Casas, E.B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L.R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, et al., Mixtral of experts, 2024, [arXiv:2401.04088](https://arxiv.org/abs/2401.04088).
- [68] P. Singer, P. Pfeiffer, Y. Babakhin, M. Jeblick, N. Dhankhar, G. Fodor, S.S. Ambati, H2O-Danube-1.8B technical report, 2024, [http://dx.doi.org/10.48550/arXiv.2401.16818](https://dx.doi.org/10.48550/arXiv.2401.16818).
- [69] P. Pfeiffer, P. Singer, Y. Babakhin, G. Fodor, N. Dhankhar, S.S. Ambati, H2O-Danube3 technical report, 2024, [http://dx.doi.org/10.48550/arXiv.2407.09276](https://dx.doi.org/10.48550/arXiv.2407.09276), [arXiv:2407.09276](https://arxiv.org/abs/2407.09276).
- [70] G. Team, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A.M. Dai, A. Hauth, K. Millican, D. Silver, M. Johnson, I. Antonoglou, J. Schrittwieser, A. Gläse, J. Chen, E. Pitler, T. Lillicrap, A. Lazaridou, O. Firat, J. Molloy, et al., Gemini: A family of highly capable multimodal models, 2024, [arXiv:2312.11805](https://arxiv.org/abs/2312.11805).
- [71] M. Wu, A. Waheed, C. Zhang, M. Abdul-Mageed, A.F. Aji, LaMini-LM: A diverse herd of distilled models from large-scale instructions, in: Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, St. Julian's, Malta, 2024, pp. 944–964.
- [72] I. Timiryasov, J.-L. Tastet, Baby llama: knowledge distillation from an ensemble of teachers trained on a small dataset with no performance penalty, 2023, [arXiv:2308.02019](https://arxiv.org/abs/2308.02019).
- [73] A. Warstadt, A. Mueller, L. Choshen, E. Wilcox, C. Zhuang, J. Ciro, R. Mosquera, B. Paranjape, A. Williams, T. Linzen, R. Cotterell, Findings of the BabyLM challenge: Sample-efficient pretraining on developmentally plausible corpora, in: Proceedings of the BabyLM Challenge At the 27th Conference on Computational Natural Language Learning, Association for Computational Linguistics, Singapore, 2023, pp. 1–34.
- [74] Y. Gu, L. Dong, F. Wei, M. Huang, MiniLLM: Knowledge distillation of large language models, in: The Twelfth International Conference on Learning Representations, 2023.
- [75] C. Zhang, D. Song, Z. Ye, Y. Gao, Towards the law of capacity gap in distilling language models, 2024, [arXiv:2311.07052](https://arxiv.org/abs/2311.07052).
- [76] M. Xia, T. Gao, Z. Zeng, D. Chen, Sheared LLaMA: Accelerating language model pre-training via structured pruning, in: The Twelfth International Conference on Learning Representations, 2023, URL: <https://openreview.net/forum?id=09iOdaeOzp>.
- [77] Meta, Llama 3.2: Revolutionizing edge AI and vision with open, customizable models, 2024, Meta AI, URL: <https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/>.
- [78] M.R. Islam, N. Dhar, B. Deng, T.N. Nguyen, S. He, K. Suo, Characterizing and understanding the performance of small language models on edge devices, in: 2024 IEEE International Performance, Computing, and Communications Conference, IPCCC, 2024, pp. 1–10, [http://dx.doi.org/10.1109/IPCCC59868.2024.10850044](https://dx.doi.org/10.1109/IPCCC59868.2024.10850044).
- [79] S. Merity, C. Xiong, J. Bradbury, R. Socher, Pointer sentinel mixture models, 2016, [arXiv:1609.07843](https://arxiv.org/abs/1609.07843).
- [80] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y.K. Li, F. Luo, Y. Xiong, W. Liang, DeepSeek-Coder: When the Large Language Model Meets Programming – The Rise of Code Intelligence, 2024, [arXiv:2401.14196](https://arxiv.org/abs/2401.14196).
- [81] C. Team, H. Zhao, J. Hui, J. Howland, N. Nguyen, S. Zuo, A. Hu, C.A. Choquette-Choo, J. Shen, J. Kelley, K. Bansal, L. Vilnis, M. Wirth, P. Michel, P. Choy, P. Joshi, R. Kumar, S. Hashmi, S. Agrawal, Z. Gong, J. Fine, T. Warkentin, A.J. Hartman, B. Ni, K. Korevec, K. Schaefer, S. Huffman, CodeGemma: Open code models based on gemma, 2024, [arXiv:2406.11409](https://arxiv.org/abs/2406.11409).
- [82] A. Mitra, H. Khanpour, C. Rosset, A. Awadallah, Orca-math: Unlocking the potential of SLMs in grade school math, 2024, [arXiv:2402.14830](https://arxiv.org/abs/2402.14830).
- [83] Y. Zhu, M. Zhu, N. Liu, Z. Ou, X. Mou, J. Tang, LLaVA-phi: Efficient multi-modal assistant with small language model, 2024, [arXiv:2401.02330](https://arxiv.org/abs/2401.02330).
- [84] Y. Yang, H. Sun, J. Li, R. Liu, Y. Li, Y. Liu, H. Huang, Y. Gao, MindLLM: Pre-training lightweight large language model from scratch, evaluations and domain applications, 2023, [arXiv:2310.15777](https://arxiv.org/abs/2310.15777).
- [85] X. Chu, L. Qiao, X. Lin, S. Xu, Y. Yang, Y. Hu, F. Wei, X. Zhang, B. Zhang, X. Wei, C. Shen, MobileVLM : A fast, strong and open vision language assistant for mobile devices, 2023, [arXiv:2312.16886](https://arxiv.org/abs/2312.16886).
- [86] X. Chu, L. Qiao, X. Zhang, S. Xu, F. Wei, Y. Yang, X. Sun, Y. Hu, X. Lin, B. Zhang, C. Shen, MobileVLM V2: Faster and stronger baseline for vision language model, 2024, [arXiv:2402.03766](https://arxiv.org/abs/2402.03766).
- [87] B. Zhou, Y. Hu, X. Weng, J. Jia, J. Luo, X. Liu, J. Wu, L. Huang, TinyLLaVA: A framework of small-scale large multimodal models, 2024, [arXiv:2402.14289](https://arxiv.org/abs/2402.14289).
- [88] M. He, Y. Liu, B. Wu, J. Yuan, Y. Wang, T. Huang, B. Zhao, Efficient multimodal learning from data-centric perspective, 2024, [arXiv:2402.11530](https://arxiv.org/abs/2402.11530).

- [89] H. Lu, W. Liu, B. Zhang, B. Wang, K. Dong, B. Liu, J. Sun, T. Ren, Z. Li, H. Yang, Y. Sun, C. Deng, H. Xu, Z. Xie, C. Ruan, DeepSeek-VL: Towards real-world vision-language understanding, 2024, [arXiv:2403.05525](#).
- [90] H. Wei, L. Kong, J. Chen, L. Zhao, Z. Ge, E. Yu, J. Sun, C. Han, X. Zhang, Small language model meets with reinforced vision vocabulary, 2024, [arXiv:2401.12503](#).
- [91] Y. Yao, T. Yu, A. Zhang, C. Wang, J. Cui, H. Zhu, T. Cai, H. Li, W. Zhao, Z. He, Q. Chen, H. Zhou, Z. Zou, H. Zhang, S. Hu, Z. Zheng, J. Zhou, J. Cai, X. Han, G. Zeng, D. Li, Z. Liu, M. Sun, MiniCPM-V: A GPT-4V level MLLM on your phone, 2024, <http://dx.doi.org/10.48550/arXiv.2408.01800>, [arXiv:2408.01800](#).
- [92] Y. Jin, J. Li, Y. Liu, T. Gu, K. Wu, Z. Jiang, M. He, B. Zhao, X. Tan, Z. Gan, Y. Wang, C. Wang, L. Ma, Efficient multimodal large language models: A survey, 2024, [arXiv:2405.10739](#).
- [93] W. Chen, Z. Li, M. Ma, Octopus: On-device language model for function calling of software APIs, 2024, <http://dx.doi.org/10.48550/arXiv.2404.01549>, [arXiv:2404.01549](#).
- [94] W. Chen, Z. Li, Octopus v2: On-device language model for super agent, 2024, <http://dx.doi.org/10.48550/arXiv.2404.01744>, [arXiv:2404.01744](#).
- [95] W. Chen, Z. Li, Octopus v3: Technical report for on-device sub-billion multimodal AI agent, 2024, <http://dx.doi.org/10.48550/arXiv.2404.11459>, [arXiv:2404.11459](#).
- [96] W. Chen, Z. Li, Octopus v4: Graph of language models, 2024, <http://dx.doi.org/10.48550/arXiv.2404.19296>, [arXiv:2404.19296](#).
- [97] T. Gunter, Z. Wang, C. Wang, R. Pang, A. Narayanan, A. Zhang, B. Zhang, C. Chen, C.-C. Chiu, D. Qiu, D. Gopinath, D.A. Yap, D. Yin, F. Nan, F. Weers, G. Yin, H. Huang, J. Wang, J. Lu, J. Peebles, K. Ye, M. Lee, N. Du, Q. Chen, Q. Keunebroek, S. Wiseman, S. Evans, T. Lei, V. Rathod, X. Kong, X. Du, Y. Li, Y. Wang, Y. Gao, Z. Ahmed, Z. Xu, Z. Lu, A. Rashid, A.M. Jose, A. Doane, A. Bencomo, A. Vanderby, A. Hansen, A. Jain, A.M. Anupama, A. Kamal, B. Wu, C. Brum, C. Maalouf, C. Erdenebileg, C. Dulhanly, D. Moritz, D. Kang, E. Jimenez, E. Ladd, F. Shi, F. Bai, F. Chu, F. Hohman, H. Kotek, H.G. Coleman, J. Li, J. Bigham, J. Cao, J. Lai, J. Cheung, J. Shan, J. Zhou, J. Li, J. Qin, K. Singh, K. Vega, K. Zou, L. Heckman, L. Gardiner, M. Bowler, M. Cordell, M. Cao, N. Hay, N. Shahdadpuri, O. Godwin, P. Digne, P. Rachapudi, R. Tantawi, R. Frigg, S. Davarnia, S. Shah, S. Guha, S. Sirovica, S. Ma, S. Ma, S. Wang, S. Kim, S. Jayaram, V. Shankar, V. Paidi, V. Kumar, X. Wang, X. Zheng, W. Cheng, Y. Shrager, Y. Ye, Y. Tanaka, Y. Guo, Y. Meng, Z.T. Luo, Z. Ouyang, A. Aygar, A. Wan, A. Walkingshaw, A. Narayanan, A. Lin, A. Farooq, B. Ramerth, C. Reed, C. Bartels, C. Chaney, D. Riazati, E.L. Yang, E. Feldman, G. Hochstrasser, G. Seguin, I. Belousova, J. Pelemans, K. Yang, K.A. Vahid, L. Cao, M. Najibi, M. Zuliani, M. Morton, M. Cho, N. Bhendawade, P. Dong, P. Maj, P. Agrawal, Q. Shan, Q. Fu, R. Poston, S. Xu, S. Liu, S. Rao, T. Heeramun, T. Merth, U. Rayala, V. Cui, V.R. Sridhar, W. Zhang, W. Zhang, W. Wu, X. Zhou, X. Liu, Y. Zhao, Y. Xia, Z. Ren, Z. Ren, Apple intelligence foundation language models, 2024, <http://dx.doi.org/10.48550/arXiv.2407.21075>, [arXiv:2407.21075](#).
- [98] C. Zhang, Z. Yang, J. Liu, Y. Han, X. Chen, Z. Huang, B. Fu, G. Yu, Appagent: Multimodal agents as smartphone users, 2023, arXiv preprint [arXiv:2312.13771](#).
- [99] A. Yan, Z. Yang, W. Zhu, K. Lin, L. Li, J. Wang, J. Yang, Y. Zhong, J. McAuley, J. Gao, et al., Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation, 2023, arXiv preprint [arXiv:2311.07562](#).
- [100] H. Wen, Y. Li, G. Liu, S. Zhao, T. Yu, T.J.-J. Li, S. Jiang, Y. Liu, Y. Zhang, Y. Liu, Autodroid: Llm-powered task automation in android, in: Proceedings of the 30th Annual International Conference on Mobile Computing and Networking, 2024, pp. 543–557.
- [101] M.D. Vu, H. Wang, Z. Li, J. Chen, S. Zhao, Z. Xing, C. Chen, GPTVoiceTasker: LLM-powered virtual assistant for smartphone, 2024, arXiv preprint [arXiv:2401.14268](#).
- [102] M. Santacroce, Z. Wen, Y. Shen, Y. Li, What matters in the structured pruning of generative language models?, 2023, [arXiv:2302.03773](#).
- [103] X. Ma, G. Fang, X. Wang, LLM-pruner: On the structural pruning of large language models, 2023, [arXiv:2305.11627](#).
- [104] T. Chen, T. Ding, B. Yadav, I. Zharkov, L. Liang, LoRAShear: Efficient large language model structured pruning and knowledge recovery, 2023, [arXiv:2310.18356](#).
- [105] M. Zhang, H. Chen, C. Shen, Z. Yang, L. Ou, X. Yu, B. Zhuang, LoRAPrune: Pruning meets low-rank parameter-efficient fine-tuning, 2023, [arXiv:2305.18403](#).
- [106] H. Wang, Z. Zhang, S. Han, SpAtten: Efficient sparse attention architecture with cascade token and head pruning, in: 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), IEEE Computer Society, 2021, pp. 97–110.
- [107] E. Kurtic, E. Frantar, D. Alistarh, ZipLM: Inference-aware structured pruning of language models, 2023, [arXiv:2302.04089](#).
- [108] E. Frantar, D. Alistarh, SparseGPT: Massive language models can be accurately pruned in one-shot, in: Proceedings of the 40th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, 202, PMLR, New York, 2023, pp. 10323–10337.
- [109] M. Sun, Z. Liu, A. Bair, J.Z. Kolter, A simple and effective pruning approach for large language models, 2023, [arXiv:2306.11695](#).
- [110] H. Shao, B. Liu, B. Xiao, K. Zeng, G. Wan, Y. Qian, One-shot sensitivity-aware mixed sparsity pruning for large language models, 2024, [arXiv:2310.09499](#).
- [111] H. Xia, Z. Zheng, Y. Li, D. Zhuang, Z. Zhou, X. Qiu, Y. Li, W. Lin, S.L. Song, Flash-LLM: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity, 2023, [arXiv:2309.10285](#).
- [112] R. Agarwal, N. Vieillard, Y. Zhou, P. Stanczyk, S. Ramos, M. Geist, O. Bachem, On-policy distillation of language models: Learning from self-generated mistakes, 2024, [arXiv:2306.13649](#).
- [113] C. Liang, S. Zuo, Q. Zhang, P. He, W. Chen, T. Zhao, Less is more: Task-aware layer-wise distillation for language model compression, in: Proceedings of the 40th International Conference on Machine Learning, PMLR, New York, 2023, pp. 20852–20867.
- [114] X. Zhu, B. Qi, K. Zhang, X. Long, B. Zhou, PaD: Program-aided distillation specializes large models in reasoning, 2023, [arXiv:2305.13888](#).
- [115] L.H. Li, J. Hessel, Y. Yu, X. Ren, K.-W. Chang, Y. Choi, Symbolic chain-of-thought distillation: Small models can also “think” step-by-step, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, vol. 1, Association for Computational Linguistics, Toronto, Canada, 2023, pp. 2665–2679.
- [116] N. Ho, L. Schmid, S.-Y. Yun, Large language models are reasoning teachers, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, 1, Association for Computational Linguistics, Toronto, Canada, 2023, pp. 14852–14882.
- [117] P. Wang, Z. Wang, Z. Li, Y. Gao, B. Yin, X. Ren, SCOTT: Self-consistent chain-of-thought distillation, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, vol. 1, Association for Computational Linguistics, Toronto, Canada, 2023, pp. 5546–5558.
- [118] C.-Y. Hsieh, C.-L. Li, C.-k. Yeh, H. Nakhost, Y. Fujii, A. Ratner, R. Krishna, C.-Y. Lee, T. Pfister, Distilling step-by-step! Outperforming larger language models with less training data and smaller model sizes, in: Findings of the Association for Computational Linguistics: ACL 2023, Association for Computational Linguistics, Toronto, Canada, 2023, pp. 8003–8017.
- [119] Z. Chen, Q. Gao, A. Bosselut, A. Sabharwal, K. Richardson, DISCO: Distilling counterfactuals with large language models, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, 1, Association for Computational Linguistics, Toronto, Canada, 2023, pp. 5514–5528.
- [120] Y. Jiang, C. Chan, M. Chen, W. Wang, Lion: Adversarial distillation of proprietary large language models, in: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Singapore, 2023, pp. 3134–3154.
- [121] X. Xu, M. Li, C. Tao, T. Shen, R. Cheng, J. Li, C. Xu, D. Tao, T. Zhou, A survey on knowledge distillation of large language models, 2024, [arXiv:2402.13116](#).
- [122] Y. Huang, Y. Chen, Z. Yu, K. McKeown, In-context learning distillation: Transferring few-shot learning ability of pre-trained language models, 2022, [arXiv:2212.10670](#).
- [123] X. Zhu, J. Li, Y. Liu, C. Ma, W. Wang, A survey on model compression for large language models, 2023, [arXiv:2308.07633](#).
- [124] Z. Liu, B. Oguiz, C. Zhao, E. Chang, P. Stock, Y. Mehdad, Y. Shi, R. Krishnamoorthi, V. Chandra, LLM-QAT: Data-free quantization aware training for large language models, 2023, [arXiv:2305.17888](#).
- [125] M. Chen, W. Shao, P. Xu, J. Wang, P. Gao, K. Zhang, Y. Qiao, P. Luo, EfficientQAT: Efficient quantization-aware training for large language models, 2024, [arXiv:2407.11062](#).
- [126] X. Shen, Z. Kong, C. Yang, Z. Han, L. Lu, P. Dong, C. Lyu, C.-h. Li, X. Guo, Z. Shu, W. Niu, M. Leeser, P. Zhao, Y. Wang, EdgeQAT: Entropy and distribution guided quantization-aware training for the acceleration of lightweight LLMs on the edge, 2024, [arXiv:2402.10787](#).
- [127] J. Lin, J. Tang, H. Tang, S. Yang, X. Dang, C. Gan, S. Han, AWQ: Activation-aware weight quantization for LLM compression and acceleration, 2023, [arXiv:2306.00978](#).
- [128] E. Frantar, S. Ashkboos, T. Hoefer, D. Alistarh, OPTQ: Accurate quantization for generative pre-trained transformers, in: Proceedings of the 11th International Conference on Learning Representations, OpenReview.net, New York, 2023.
- [129] T. Dettmers, R. Svirshevski, V. Egiazarian, D. Kuznedelev, E. Frantar, S. Ashkboos, A. Borzunov, T. Hoefer, D. Alistarh, SpQR: A sparse-quantized representation for near-lossless LLM weight compression, 2023, [arXiv:2306.03078](#).
- [130] C. Lee, J. Jin, T. Kim, H. Kim, E. Park, OWQ: Lessons learned from activation outliers for weight quantization in large language models, 2023, [arXiv:2306.02272](#).
- [131] V. Egiazarian, A. Panferov, D. Kuznedelev, E. Frantar, A. Babenko, D. Alistarh, Extreme compression of large language models via additive quantization, in: Forty-First International Conference on Machine Learning, 2024.
- [132] T. Dettmers, M. Lewis, Y. Belkada, L. Zettlemoyer, GPT3.int8Q: 8-bit matrix multiplication for transformers at scale, Adv. Neural Inf. Process. Syst. 35 35 (2022) 30318–30332.
- [133] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, S. Han, SmoothQuant: Accurate and efficient post-training quantization for large language models, in: Proceedings of the 40th International Conference on Machine Learning, PMLR, 2023, pp. 38087–38099.

- [134] Z. Yao, R. Yazdani Aminabadi, M. Zhang, X. Wu, C. Li, Y. He, ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers, *Adv. Neural Inf. Process. Syst.* 35 (2022) 27168–27183.
- [135] X. Wei, Y. Zhang, Y. Li, X. Zhang, R. Gong, J. Guo, X. Liu, Outlier suppression+: Accurate quantization of large language models by equivalent and effective shifting and scaling, in: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Singapore, 2023, pp. 1648–1665.
- [136] X. Wei, Y. Zhang, X. Zhang, R. Gong, S. Zhang, Q. Zhang, F. Yu, X. Liu, Outlier suppression: Pushing the limit of low-bit transformer language models, *Adv. Neural Inf. Process. Syst.* 35 (2022) 17402–17414.
- [137] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, Y. Zhu, OliVe: Accelerating large language models via hardware-friendly outlier-victim pair quantization, in: Proceedings of the 50th Annual International Symposium on Computer Architecture, in: ISCA '23, Association for Computing Machinery, New York, 2023, pp. 1–15.
- [138] J. Liu, R. Gong, X. Wei, Z. Dong, J. Cai, B. Zhuang, QLLM: Accurate and efficient low-bitwidth quantization for large language models, 2023, [arXiv:2310.08041](https://arxiv.org/abs/2310.08041).
- [139] Q. Li, Y. Zhang, L. Li, P. Yao, B. Zhang, X. Chu, Y. Sun, L. Du, Y. Xie, FPTQ: Fine-grained post-training quantization for large language models, 2023, [arXiv:2308.15987](https://arxiv.org/abs/2308.15987).
- [140] X. Shen, P. Dong, L. Lu, Z. Kong, Z. Li, M. Lin, C. Wu, Y. Wang, Agile-quant: Activation-guided quantization for faster inference of LLMs on the edge, *Proc. the AAAI Conf. Artif. Intell.* 38 (17) (2024) 18944–18951.
- [141] Y. Shang, Z. Yuan, Q. Wu, Z. Dong, PB-LLM: Partially binarized large language models, 2023, [arXiv:2310.00034](https://arxiv.org/abs/2310.00034).
- [142] W. Huang, Y. Liu, H. Qin, Y. Li, S. Zhang, X. Liu, M. Magno, X. Qi, BiLLM: Pushing the limit of post-training quantization for LLMs, 2024, [arXiv:2402.04291](https://arxiv.org/abs/2402.04291).
- [143] H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, F. Wei, BitNet: Scaling 1-bit transformers for large language models, 2023, [arXiv:2310.11453](https://arxiv.org/abs/2310.11453).
- [144] S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, F. Wei, The era of 1-bit LLMs: All large language models are in 1.58 bits, 2024, [arXiv:2402.17764](https://arxiv.org/abs/2402.17764).
- [145] E.J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, LoRA: Low-rank adaptation of large language models, in: Proceedings of the 10th International Conference on Learning Representations, OpenReview.net, New York, 2022.
- [146] A. Kaushal, T. Vaidhya, I. Rish, LORD: Low rank decomposition of monolingual code LLMs for one-shot compression, 2023, [arXiv:2309.14021](https://arxiv.org/abs/2309.14021).
- [147] Y. Li, Y. Yu, Q. Zhang, C. Liang, P. He, W. Chen, T. Zhao, LoSparse: Structured compression of large language models based on low-rank and sparse approximation, in: Proceedings of the 40th International Conference on Machine Learning, PMLR, New York, 2023, pp. 20336–20350.
- [148] M. Xu, Y.L. Xu, D.P. Mandic, TensorGPT: Efficient compression of the embedding layer in LLMs based on the tensor-train decomposition, 2023, [arXiv:2307.00526](https://arxiv.org/abs/2307.00526).
- [149] T. Dettmers, A. Pagnoni, A. Holtzman, L. Zettlemoyer, QLoRA: Efficient finetuning of quantized LLMs, 2023, [arXiv:2305.14314](https://arxiv.org/abs/2305.14314).
- [150] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, J. Jumper, Accelerating large language model decoding with speculative sampling, 2023, [arXiv:2302.01318](https://arxiv.org/abs/2302.01318).
- [151] Y. Leviathan, M. Kalman, Y. Matias, Fast inference from transformers via speculative decoding, in: Proceedings of the 40th International Conference on Machine Learning, in: ICML'23, vol. 202, JMLR.org, Honolulu, Hawaii, USA, 2023, pp. 19274–19286.
- [152] S. Kim, K. Mangalam, S. Moon, J. Malik, M.W. Mahoney, A. Gholami, K. Keutzer, Speculative decoding with big little decoder, *Adv. Neural Inf. Process. Syst.* 36 (2023) 39236–39256.
- [153] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, Z. Wang, Z. Zhang, R.Y.Y. Wong, A. Zhu, L. Yang, X. Shi, C. Shi, Z. Chen, D. Arfeen, R. Abhyankar, Z. Jia, SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification, in: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, in: ASPLOS '24, vol. 3, Association for Computing Machinery, New York, NY, USA, 2024, pp. 932–949.
- [154] Y. Li, F. Wei, C. Zhang, H. Zhang, EAGLE: Speculative sampling requires rethinking feature uncertainty, 2024, [arXiv:2401.15077](https://arxiv.org/abs/2401.15077).
- [155] T. Cai, Y. Li, Z. Geng, H. Peng, J.D. Lee, D. Chen, T. Dao, Medusa: Simple LLM inference acceleration framework with multiple decoding heads, 2024, [arXiv:2401.10774](https://arxiv.org/abs/2401.10774).
- [156] Y. Fu, P. Bailis, I. Stoica, H. Zhang, Break the sequential dependency of LLM inference using lookahead decoding, 2024, [arXiv:2402.02057](https://arxiv.org/abs/2402.02057).
- [157] D. Xu, W. Yin, X. Jin, Y. Zhang, S. Wei, M. Xu, X. Liu, LLMCad: Fast and scalable on-device large language model inference, 2023, [arXiv:2309.04255](https://arxiv.org/abs/2309.04255).
- [158] R. Svirshevski, A. May, Z. Chen, B. Chen, Z. Jia, M. Ryabinin, SpecExec: Massively parallel speculative decoding for interactive LLM inference on consumer devices, 2024, [arXiv:2406.02532](https://arxiv.org/abs/2406.02532).
- [159] C. Hooper, S. Kim, H. Mohammadzadeh, M.W. Mahoney, Y.S. Shao, K. Keutzer, A. Gholami, KVQuant: Towards 10 million context length LLM inference with KV Cache Quantization, 2024, [arXiv:2401.18079](https://arxiv.org/abs/2401.18079).
- [160] S. Ge, Y. Zhang, L. Liu, M. Zhang, J. Han, J. Gao, Model tells you what to discard: Adaptive KV cache compression for LLMs, in: The Twelfth International Conference on Learning Representations, 2023.
- [161] G. Xiao, Y. Tian, B. Chen, S. Han, M. Lewis, Efficient streaming language models with attention sinks, in: The Twelfth International Conference on Learning Representations, 2023.
- [162] Z. Liu, A. Desai, F. Liao, W. Wang, V. Xie, Z. Xu, A. Kyriolidis, A. Shrivastava, Scissorhands: exploiting the persistence of importance hypothesis for LLM KV cache compression at test time, in: Proceedings of the 37th International Conference on Neural Information Processing Systems, in: NIPS '23, Curran Associates Inc., Red Hook, NY, USA, 2024, pp. 52342–52364.
- [163] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, Z.A. Wang, B. Chen, H2O: Heavy-hitter oracle for efficient generative inference of large language models, *Adv. Neural Inf. Process. Syst.* 36 (2023) 34661–34710.
- [164] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, B. Chen, X. Hu, KIVI: A tuning-free asymmetric 2bit quantization for KV cache, 2023, [arXiv:2402.02750](https://arxiv.org/abs/2402.02750).
- [165] S. Dong, W. Cheng, J. Qin, W. Wang, QAQ: Quality adaptive quantization for LLM KV cache, 2024, [arXiv:2403.04643](https://arxiv.org/abs/2403.04643).
- [166] R. Liu, H. Bai, H. Lin, Y. Li, H. Gao, Z. Xu, L. Hou, J. Yao, C. Yuan, IntactKV: Improving large language model quantization by keeping pivot tokens intact, 2024, [arXiv:2403.01241](https://arxiv.org/abs/2403.01241).
- [167] T. Schuster, A. Fisch, J. Gupta, M. Dehghani, D. Bahri, V. Tran, Y. Tay, D. Metzler, Confident adaptive language modeling, *Adv. Neural Inf. Process. Syst.* 35 (2022) 17456–17472.
- [168] Z. Zeng, Y. Hong, H. Dai, H. Zhuang, C. Chen, ConsistentEE: A consistent and hardness-guided early exiting method for accelerating language models inference, in: Proceedings of the AAAI Conference on Artificial Intelligence, 38, 2024, pp. 19506–19514.
- [169] L. Del Corro, A. Del Giorno, S. Agarwal, B. Yu, A. Awadallah, S. Mukherjee, SkipDecode: Autoregressive skip decoding with batching and caching for efficient LLM inference, 2023, [arXiv:2307.02628](https://arxiv.org/abs/2307.02628).
- [170] Y. Chen, X. Pan, Y. Li, B. Ding, J. Zhou, EE-LLM: Large-scale training and inference of early-exit large language models with 3D parallelism, 2024, [arXiv:2312.04916](https://arxiv.org/abs/2312.04916).
- [171] S. Bae, J. Ko, H. Song, S.-Y. Yun, Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding, in: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Singapore, 2023, pp. 5910–5924.
- [172] M. Elhoushi, A. Shrivastava, D. Liskovich, B. Hosmer, B. Wasti, L. Lai, A. Mahmoud, B. Acun, S. Agarwal, A. Roman, A.A. Aly, B. Chen, C.-J. Wu, LayerSkip: Enabling early exit inference and self-speculative decoding, 2024, [arXiv:2404.16710](https://arxiv.org/abs/2404.16710).
- [173] T. Dao, D. Fu, S. Ermon, A. Rudra, C. Ré, FlashAttention: Fast and memory-efficient exact attention with IO-awareness, *Adv. Neural Inf. Process. Syst.* 35 (2022) 16344–16359.
- [174] T. Dao, FlashAttention-2: Faster attention with better parallelism and work partitioning, 2023, [arXiv:2307.08691](https://arxiv.org/abs/2307.08691).
- [175] T. Dao, D. Haziza, F. Massa, G. Sizov, Flash-decoding for long-context inference, 2023, URL: <https://pytorch.org/blog/flash-decoding/>, (Accessed: Feb. 3, 2024).
- [176] K. Hong, G. Dai, J. Xu, Q. Mao, X. Li, J. Liu, K. Chen, Y. Dong, Y. Wang, FlashDecoding++: Faster large language model inference on GPUs, 2024, [arXiv:2311.01282](https://arxiv.org/abs/2311.01282).
- [177] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C.H. Yu, J. Gonzalez, H. Zhang, I. Stoica, Efficient memory management for large language model serving with PagedAttention, in: Proceedings of the 29th Symposium on Operating Systems Principles, in: SOSP '23, Association for Computing Machinery, New York, 2023, pp. 611–626.
- [178] J. Wei, S. Cao, T. Cao, L. Ma, L. Wang, Y. Zhang, M. Yang, T-MAC: CPU renaissance via table lookup for low-bit LLM deployment on edge, 2024, [arXiv:2407.00088](https://arxiv.org/abs/2407.00088).
- [179] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, et al., Transformers: State-of-the-art natural language processing, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Association for Computational Linguistics, Online, 2020, pp. 38–45.
- [180] G. Gerganov, Ggerganov/llama.cpp, 2023, URL: <https://github.com/ggerganov/llama.cpp>. (Accessed: Jun. 26, 2024).
- [181] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Re, I. Stoica, C. Zhang, FlexGen: High-throughput generative inference of large language models with a single GPU, in: Proceedings of the 40th International Conference on Machine Learning, PMLR, New York, 2023, pp. 31094–31116.
- [182] S. Chen, Y. Lin, M. Zhang, Y. Wu, Efficient and economic large language model inference with attention offloading, 2024, [arXiv:2405.01814](https://arxiv.org/abs/2405.01814).

- [183] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Re, B. Chen, Deja Vu: Contextual sparsity for efficient LLMs at inference time, in: Proceedings of the 40th International Conference on Machine Learning, PMLR, New York, 2023, pp. 22137–22176.
- [184] Y. Song, Z. Mi, H. Xie, H. Chen, PowerInfer: Fast large language model serving with a consumer-grade GPU, 2023, [arXiv:2312.12456](https://arxiv.org/abs/2312.12456).
- [185] K. Alizadeh, I. Mirzadeh, D. Belenko, K. Khatamifarid, M. Cho, C.C. Del Mundo, M. Rastegari, M. Farajtabar, LLM in a flash: Efficient large language model inference with limited memory, 2024, [arXiv:2312.11514](https://arxiv.org/abs/2312.11514).
- [186] R. Yi, L. Guo, S. Wei, A. Zhou, S. Wang, M. Xu, EdgeMoE: Fast on-device inference of MoE-based large language models, 2023, [arXiv:2308.14352](https://arxiv.org/abs/2308.14352).
- [187] NVIDIA/TensorRT-LLM, 2023, URL: <https://github.com/NVIDIA/TensorRT-LLM>, (Accessed: Jun. 29, 2024).
- [188] R.Y. Aminabadi, S. Rajbhandari, A.A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley, Y. He, DeepSpeed-inference: Enabling efficient inference of transformer models at unprecedented scale, in: SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, 2022, pp. 1–15.
- [189] huggingface/text-generation-inference, 2023, URL: <https://github.com/huggingface/text-generation-inference>. (Accessed: Jun. 29, 2024).
- [190] InternLM/lmdeploy, 2023, URL: <https://github.com/InternLM/lmdeploy>. (Accessed: Jun. 29, 2024).
- [191] PyTorch ExecuTorch, 2023, URL: <https://pytorch.org/executorch-overview>. (Accessed: Jun. 27, 2024).
- [192] TensorFlow lite | ML for mobile and edge devices, 2017, URL: <https://www.tensorflow.org/lite>. (Accessed: May 21, 2024).
- [193] alibaba/MNN, 2020, URL: <https://github.com/alibaba/MNN>. (Accessed: Jun. 29, 2024).
- [194] H. Ni, The ncnn contributors, ncnn, 2017, URL: <https://github.com/Tencent/ncnn>, (Accessed: Jun. 29, 2024).
- [195] Z. Xue, Y. Song, Z. Mi, L. Chen, Y. Xia, H. Chen, PowerInfer-2: Fast large language model inference on a smartphone, 2024, [arXiv:2406.06282](https://arxiv.org/abs/2406.06282).
- [196] L. Li, S. Qian, J. Lu, L. Yuan, R. Wang, Q. Xie, Transformer-lite: High-efficiency deployment of large language models on mobile phone GPUs, 2024, [arXiv:2403.20041](https://arxiv.org/abs/2403.20041).
- [197] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, T. Abdelzaher, Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency, in: Proceedings of the 18th Conference on Embedded Networked Sensor Systems, in: SenSys '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 476–488.
- [198] Z. Zhao, K. Wang, N. Ling, G. Xing, EdgeML: An AutoML framework for real-time deep learning on the edge, in: Proceedings of the International Conference on Internet-of-Things Design and Implementation, in: IoTDI '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 133–144.
- [199] S. Laskaridis, S.I. Venieris, M. Almeida, I. Leontiadis, N.D. Lane, SPINN: Synergistic progressive inference of neural networks over device and cloud, in: Proceedings of the 26th Annual International Conference on Mobile Computing and Networking, in: MobiCom '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1–15.
- [200] K. Huang, W. Gao, Real-time neural network inference on extremely weak devices: Agile offloading with explainable AI, in: Proceedings of the 28th Annual International Conference on Mobile Computing and Networking, in: MobiCom '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 200–213.
- [201] D. Ding, A. Mallick, C. Wang, R. Sim, S. Mukherjee, V. Rühle, L.V.S. Lakshmanan, A.H. Awadallah, Hybrid LLM: Cost-efficient and quality-aware query routing, in: The Twelfth International Conference on Learning Representations, 2023.
- [202] Y. Wang, K. Chen, H. Tan, K. Guo, Tabi: An efficient multi-level inference system for large language models, in: Proceedings of the Eighteenth European Conference on Computer Systems, in: EuroSys '23, Association for Computing Machinery, New York, NY, USA, 2023, pp. 233–248.
- [203] F. Cai, D. Yuan, Z. Yang, L. Cui, Edge-LLM: A collaborative framework for large language model serving in edge computing, in: 2024 IEEE International Conference on Web Services, ICWS, IEEE, 2024, pp. 799–809.
- [204] Z. Wu, S. Sun, Y. Wang, M. Liu, B. Gao, Q. Pan, T. He, X. Jiang, Agglomerative federated learning: Empowering larger model training via end-edge-cloud collaboration, in: IEEE INFOCOM 2024-IEEE Conference on Computer Communications, IEEE, 2024, pp. 131–140.
- [205] mlc-ai/mlc-llm, 2023, URL: <https://github.com/mlc-ai/mlc-llm>. (Accessed: Feb. 4, 2024).
- [206] UbiquitousLearning/mllm, 2023, URL: <https://github.com/UbiquitousLearning/mllm>. (Accessed: Feb. 4, 2024).
- [207] D. Franklin, dusty-nv/NanoLLM, 2024, URL: <https://github.com/dusty-nv/NanoLLM>. (Accessed: Jun. 30, 2024).
- [208] Z. Hao, H. Jiang, S. Jiang, J. Ren, T. Cao, Hybrid SLM and LLM for edge-cloud collaborative inference, in: Proceedings of the Workshop on Edge and Mobile Foundation Models, in: EdgeFM '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 36–41.
- [209] G. Gerganov, ggerganov/ggml, 2023, URL: <https://github.com/ggerganov/ggml>. (Accessed: Jun. 30, 2024).
- [210] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, Z. Chen, GShard: Scaling giant models with conditional computation and automatic sharding, in: International Conference on Learning Representations, 2020.
- [211] B. Peng, E. Alcaide, Q. Anthony, A. Albalak, S. Arcadinho, S. Biderman, H. Cao, X. Cheng, M. Chung, L. Derczynski, X. Du, M. Grella, K. Gv, X. He, H. Hou, P. Kazienko, J. Kocon, J. Kong, B. Koptyra, H. Lau, J. Lin, K.S.I. Mantri, F. Mom, A. Saito, G. Song, X. Tang, J. Wind, S. Woźniak, Z. Zhang, Q. Zhou, J. Zhu, R.-J. Zhu, RWKV: Reinventing RNNs for the transformer era, in: Findings of the Association for Computational Linguistics: EMNLP 2023, Association for Computational Linguistics, Singapore, 2023, pp. 14048–14077.
- [212] A. Gu, T. Dao, Mamba: Linear-time sequence modeling with selective state spaces, 2024, [arXiv:2312.00752](https://arxiv.org/abs/2312.00752).
- [213] Y. Sun, L. Dong, S. Huang, S. Ma, Y. Xia, J. Xue, J. Wang, F. Wei, Retentive network : A successor to transformer for large language models, 2023, [arXiv:2307.08621](https://arxiv.org/abs/2307.08621).
- [214] W. Yin, M. Xu, Y. Li, X. Liu, LLM as a system service on mobile devices, 2024, [arXiv:2403.11805](https://arxiv.org/abs/2403.11805).
- [215] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, G. Kim, D. Mahajan, J. Park, NeuPIMs: NPU-PIM heterogeneous acceleration for batched LLM inferencing, in: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, in: ASPLOS '24, vol. 3, Association for Computing Machinery, New York, NY, USA, 2024, pp. 722–737.
- [216] Y. Hu, C. Imes, X. Zhao, S. Kundu, P.A. Beerel, S.P. Crago, J.P. Walters, PipeEdge: Pipeline parallelism for large-scale model inference on heterogeneous edge devices, in: 2022 25th Euromicro Conference on Digital System Design (DSD), 2022, pp. 298–307.
- [217] Y. Jiang, R. Yan, X. Yao, Y. Zhou, B. Chen, B. Yuan, HexGen: Generative inference of large language model over heterogeneous environment, in: Proceedings of the 41st International Conference on Machine Learning, PMLR, 2024, pp. 21946–21961.
- [218] Z. Xuanlei, B. Jia, H. Zhou, Z. Liu, S. Cheng, Y. You, HeteGen: Efficient heterogeneous parallel inference for large language models on resource-constrained devices, Proc. Mach. Learn. Syst. 6 (2024) 162–172.
- [219] H. Zhang, J. Da, D. Lee, V. Robinson, C. Wu, W. Song, T. Zhao, P. Raja, D. Slack, Q. Lyu, S. Hendryx, R. Kaplan, M. Lunati, S. Yue, A careful examination of large language model performance on grade school arithmetic, 2024, [arXiv:2405.00332](https://arxiv.org/abs/2405.00332).
- [220] A. Salemi, S. Mysore, M. Bendersky, H. Zamani, Lamp: When large language models meet personalization, 2023, arXiv preprint [arXiv:2304.11406](https://arxiv.org/abs/2304.11406).
- [221] S. Doddapaneni, K. Sayana, A. Jash, S. Sodhi, D. Kuzmin, User embedding model for personalized language prompting, 2024, arXiv preprint [arXiv:2401.04858](https://arxiv.org/abs/2401.04858).
- [222] L. Guo, W. Choe, F.X. Lin, Sti: Turbocharge nlp inference at the edge via elastic pipelining, in: Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, vol. 2, 2023, pp. 791–803.
- [223] J. Yuan, C. Yang, D. Cai, S. Wang, X. Yuan, Z. Zhang, X. Li, D. Zhang, H. Mei, X. Jia, et al., Mobile foundation model as firmware, in: Proceedings of the 30th Annual International Conference on Mobile Computing and Networking, 2024, pp. 279–295.