Language Models at the Edge: A Survey on Techniques, Challenges, and Applications

Siem Hadish , Velibor Bojković , Moayad Aloqaily *Senior Member, IEEE* , Mohsen Guizani *Fellow, IEEE*
Mohamed Bin Zayed University of Artificial Intelligence (MBZUAI), UAE
Email: {siem.hadish, velibor.bojkovic, moayad.aloqaily}@mbzuai.ac.ae, mguizani@ieee.org

2024 2nd International Conference on Foundation and Large Language Models (FLLM) | 979-8-3503-5479-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/FLLM63129.2024.10852473

***Abstract*—The invention of the transformer architectures has spurred Large Language Models (LLMs) to the forefront of artificial intelligence, driving state-of-the-art advancements across various domains. However, the huge scale of these models, often comprising hundreds of billions of parameters, presents significant challenges in terms of computational resources for both training and deployment. Consequently, the development and implementation of LLMs have largely been confined to major technology corporations. Recent research has focused on addressing these limitations by developing compact LLMs capable of operating on edge devices, rather than relying on centralized server infrastructure. This approach offers numerous benefits, including reduced computational costs, lower latency, enhanced security, and the potential for task-specific, specialized models. This paper examines the latest research trends and developments in the field of compact LLMs, exploring their applications and discussing the challenges associated with their implementation. Our investigation aims to provide insights into the future landscape of accessible and efficient language models.**

***Index Terms*—AI, ML, LLM, Edge Computing, Edge AI, EdgeLLM, Model Compression**

I. Introduction

The remarkable capabilities of LLMs have revolutionized the field of AI and pushed the boundaries of language understanding and processing. The original transformer paper demonstrated the unique attention mechanism, highlighting its performance in language translation, setting new state of the art on such tasks [1]. This model architecture opened up a path to the development and deployment of multiple transformer-based Language Models and advanced the language-based tasks in the fields of question answering, language inference, text classification [2], [3].

The performance of these models is still improving setting up new state of the art performance. A 2020 study demonstrated a positive correlation between the increasing size of language models and their performance, indicating that larger models have enhanced capabilities [4], and following this thesis, large tech companies are developing and releasing larger and larger models with up to 100s of billions of parameters.These improvements seen in larger language models, however, come at the cost of their enormous size. The billions of parameters require huge computational power and memory, and the financial costs make these models hard to train or deploy. The latest version of llama, llama 3.1 for example was pre-trained on a cluster of 16k NVIDIA GPU clusters [5], making it out of reach for most individuals and organizations to replicate, fine-tune or even perform inference. The vast size and compute resources required to train and run these models have led to most of these models to be hosted on the server and clients sending web requests to these models through the internet.

The ongoing focus in AI research is on making models smaller, faster, more efficient, and less expensive to deploy, with an emphasis on reducing environmental impact. This includes compressing models to fit on edge devices, which enables faster and potentially more accurate inference by fine-tuning them for specific tasks, thus creating specialized language models [6]. This balance aims to enhance model performance while addressing practical concerns related to resources and sustainability.

A. Contribution (Comparison)

In this article, we explore the current state and trends in EdgeLLM, discussing a range of aspects like performances of specialized models, compression techniques, security aspects, and applications/use-cases of such models. The topics discussed include:

- **Resource-Efficient Training, Fine-Tuning, and Inference** We explore parameter-efficient techniques, introduce the ideas, how they apply to LLMs, and the prominent techniques shaping the current state of the art.
- **Distributed training/fine-tuning:** We explore the literature around distributed LLM training and fine-tuning, mostly in the context of federated learning.
- **Differential Privacy:** With federated learning distributed EdgeLLM training opens up an opportunity for private data to be used for training more robust LLMs. In this article we explore how differential privacy is used to maintain privacy and security.
- **Applications of EdgeLLMs:** The final section discusses EdgeLLM ecosystems collaborating as part of greater systems other than just stand alone models.

The comparison of our survey with other similar papers is given in table I.

II. Preliminaries

To comprehensively understand the deployment of Large Language Models (LLMs) on edge devices, it is essential to establish a foundational understanding of both LLMs and

TABLE I
A COMPARATIVE ANALYSIS OF SIMILAR SURVEYS

Article	Resource-efficient Techniques	Distributed Training	Differential Privacy	EdgeLLM Applications
[7]	✓	✗	✗	✗
[8]	✓	✓	✗	✗
[9]	✓	✗	✗	✗
[10]	✓	✓	✗	✓
Ours	✓	✓	✓	✓

edge computing. This section delineates the core concepts, architectures, and techniques that underpin the subsequent discussions on EdgeLLMs.

A. Large Language Models (LLMs)

LLMs are a class of artificial intelligence models characterized by their substantial number of parameters, often ranging from billions to hundreds of billions. These models leverage the transformer architecture [1], which utilizes self-attention mechanisms to capture intricate patterns in data, enabling sophisticated language understanding and generation capabilities. The evolution of LLMs, from models like GPT-1 [2] to more advanced iterations such as GPT-4, has significantly enhanced performance across various natural language processing (NLP) tasks, including machine translation, question answering, and text summarization.

1) Transformer Architecture: At the heart of LLMs lies the transformer architecture, introduced by Vaswani et al. [1]. This architecture comprises multiple layers of self-attention and feed-forward neural networks, allowing the model to weigh the relevance of different words in a sequence dynamically. Key components include:

- **Self-Attention Mechanism:** Facilitates the model's ability to focus on relevant parts of the input sequence when generating each word.
- **Multi-Head Attention:** Enables the model to attend to information from different representation subspaces at different positions.
- **Positional Encoding:** Injects information about the position of words in a sequence, compensating for the model's lack of inherent sequential awareness.

B. Edge Computing

Edge computing refers to the paradigm of processing data closer to its source rather than relying on centralized cloud servers. This approach reduces latency, minimizes bandwidth usage, and enhances data privacy by keeping sensitive information on local devices. Edge devices encompass a wide range of hardware, including smartphones, IoT sensors, autonomous vehicles, and wearable technology, each with inherent constraints in computational power, memory, and energy consumption.

C. EdgeLLMs: Integrating LLMs with Edge Computing

Deploying LLMs on edge devices, termed EdgeLLMs, seeks to harness the advanced language capabilities of LLMs within the resource-constrained environments of edge computing. This integration promises several advantages:

- **Reduced Latency:** Local processing eliminates the need for data transmission to remote servers, enabling real-time applications.
- **Enhanced Privacy:** Sensitive data remains on-device, mitigating risks associated with data breaches and unauthorized access.
- **Operational Continuity:** EdgeLLMs can function without continuous internet connectivity, ensuring reliability in diverse environments.

D. Model Compression Techniques

Given the substantial size of LLMs, deploying them on edge devices necessitates effective model compression strategies to address computational and memory limitations. The primary techniques are explored in the next section.

E. Privacy and Security in EdgeLLMs

Maintaining data privacy and ensuring security are paramount when deploying LLMs on edge devices. Techniques such as Federated Learning [11] enable decentralized model training by aggregating updates from multiple devices without centralizing the data. Additionally, Differential Privacy [12] provides mathematical guarantees to protect individual data points during model training and inference.

F. Key Terminologies

- **Inference:** The process of generating predictions or outputs from a trained model.
- **Latency:** The delay between input submission and output generation in model processing.
- **On-Device Learning:** The capability to train or fine-tune models directly on edge devices.

III. RESOURCE-EFFICIENT TECHNIQUES

To adapt ”large” language models for edge devices, a variety of techniques are being developed to address multiple resource constraints. In this section, we will examine the most prevalent methods for enabling EdgeLLMs, categorizing them into training, fine-tuning, and inference phases.

A. Training

Although the general effort in EdgeLLMs is to compress the widely accepted pre-trained models, there can be multiple advantages of training your own models at the edge. Edge training is highly valued by IoT researchers, who try to combine sensing, storing, communicating, and intelligence at edge devices [13]. Edge training is concerned with communication efficiency on top of the computation and performance of the models. On edge training in the context of IoT is mostly associated with distributed learning and the paper [13] examines different types of distributed edge training methodologies including but not limited to federated learning (FL) [11], [14], decentralized learning [15]–[17], model split learning [18], [19], and sets optimization goal of the loss function:

$$\min_{\theta \in \mathbb{R}^d} \quad \mathcal{L}(\theta) := \sum_{k \in \mathcal{S}} w_k \mathcal{L}_k(\theta; \mathcal{D}_k) \tag{1}$$

Which is the weighted averaging of loss from a number of edge devices in $\mathcal{S}$ with dataset $\mathcal{D}_k$ local in k^{th} device. Regarding the training of LLMs from scratch at the edge, we found limited research. In [20], authors discuss the distribution of data and compute resources in the world, and refer to scaling laws [4], saying that the performance of LLMs will depend on the amount of data they are trained on. The data governance and policies limit the amount of data a single company can have and hence the best performing models in the future are going to come through federated pre-training at the edge [20]. Most existing training techniques rely on pre-trained models, which are then fine-tuned on edge devices according to specific needs and available datasets. The next section will explore compression techniques for fine-tuning.

B. Fine-tuning

Transfer learning is the transfer of knowledge from a model trained to solve task A to solve task B. Transfer learning aims to improve learning in a target domain (task B) by leveraging knowledge from a source domain (model trained for task A), which may have different features or distributions [21]. In the article [22], the authors explored the challenge of domain adaptation in sentiment analysis, recognizing that training separate models for different product domains would be impractical. They investigated the similarities between reviews across various product domains and found that, despite the differences, certain features were common across domains. By transferring features learned from domain A to domain B and subsequently fine-tuning the model, they achieved better performance compared to training a model solely on the dataset for domain B. This approach, leveraging shared intermediate representations, significantly outperformed the baseline results. To further test their findings the authors examined the transferability of features in deep neural networks, highlighting that while lower-layer features tend to be general across different tasks, higher-layer features become increasingly specific to the original task. They recognized the challenge of training separate models for different domains and instead focused on transferring features from a base model trained on domain A to a target model for domain B. Their experiments demonstrated that fine-tuning the transferred features on the target domain outperformed training a model from scratch on domain B, thereby validating the effectiveness of feature transfer across domains.

The large number of parameters and feature space in LLMs allows for absorbing vast knowledge that can be transferred into a number of domains. Entities with smaller compute resources therefore opt to fine-tune models with additional data for a specific task instead of training from scratch [23]. This strategy also holds for EdgeLLM where it might be optimal to compress and fine-tune.

Fine-tuning was found to be effective in vision models, and with some improvisation, for most smaller tasks in NLP, fine-tuning is the superior method than training from scratch [24]. The technique used by [24] for effectively fine-tuning NLP models is discriminative fine-tuning, which builds up from [25]'s claim stating that different layers capture different information, and therefore proposed that layers should learn at different rates, modifying SGD at step t as shown:

$$\theta_t^l = \theta_{t-1} - \eta \nabla_{\theta_t^l} \mathcal{L}(\theta_t) \tag{2}$$

Where instead of taking a constant learning rate η for all the layers, we take η^l, based on the layer improving the state of the art on multiple tasks.

However, just fine-tuning models is not enough for EdgeLLMs, as there is a further need to compress the models and fine-tuning process due to the constraints associated with edge devices.

One of the techniques proposed to tackle the memory constraint of EdgeLLM fine-tuning is the zeroth-order fine-tuning proposed in [26]. The paper trades off the speed and accuracy of the common fine-tuning through back propagation for reducing the memory usage saving up to $12\times$ with OPT-13B model.

Another zeroth-order method which focuses on 0.1% of the weights was explored in the paper [27]. This method includes ideas of 4-bit quantization and salience of activations, which are mostly associated with inference and will be discussed in the inference section. The paper experimented with Llamma2-7B, Mistral-7B, an OPT-6.7B, where for the Llamma2-7B, the fine-tuning only needed less than 8GB of memory.

Low-Rank Adaptation (LoRA) method of fine-tuning [28] is a method of fine-tuning that is designed to reduce the storage of fine-tuned models for different tasks. During fine-tuning with LoRA, the original model is frozen and the subsequent updates are stored separately in a low-rank decomposed form, allowing to save multiple models for n different tasks, without storing n sets of large language model parameters.

$$h = W_0 x + \nabla W x => W_0 x + BAx \tag{3}$$

Where the output h is calculated by summing the outputs of the original model and the updates during fine-tuning. Instead of keeping the updates ∇Wx as is however, it is decomposed into a low-rank adaptation BA, where B and A are smaller matrices. The authors demonstrated that increasing the rank from 1 to 64 brings about minimal improvement and during inference, as the two matrices are added up, the increase in compute is zero compared to other adapting techniques.

Another approach used to tackle the constraints associated with fine-tuning is the use of parameter-efficient methods, similar to LoRA, this approach tries to keep most of the parameters unchanged while adding a relatively small amount of weights, around 3%, allowing for the storage of models to tackle multiple tasks by just switching the adapter weights [23]. In their proposal, the adapter modules are added after every attention and feed forward sub-layers found in every transformer layer. These adapters compress the output of the sub-layers, apply non-linearity and then project into the original input layer before normalizing and passing into the following sub-layers, allowing users just to swap a small portion of the weights and keep one large model. In their experiment, they compared their approach to fine-tuning, and they could find that the parameter-efficient method could produce results similar to full fine-tuning, within 0.4% accuracy on the GLUE benchmark with BERT model.

Extending the use of adapters as parameter-efficient technique, in [29], adapters are added to a single model per each task (2-3% of the weight)that the model is to be utilized for. This work, instead of swapping adapters or low-rank adaptation, combines all of them and using task embedding, information is passed on which task is required, can switch between multiple tasks without switching weights. After testing on 5 different NLP tasks, the versatile model could outperform, or reach to similar models as the state of the art fine-tuned models.

To further shrink the size and computational and storage needs of large language models with adapter based parameter-efficient fine-tuning, the authors in [30] propose a way to drop adapter based on their importance for further efficiency. The paper qunatifies adapter based fine-tunings advantage over the full version, stating that adapter based is about 60% faster. The authors tested the effectiveness of adapters by dynamically dropping the earlier adapters and studying its effect on the accuracy. They found that the first 5 layers do not cause much drop in accuracy, but increase the inference speed by above 20%.

C. Inference

Effective training and fine-tuning is crucial to save cost of compute and circumvent the memory limitations. Training and fine-tuning are however not as critical to enabling EdgeAI or EdgeLLM as much as inference; for this reason most focus in the area of EdgeLLM is model compression for inference, after training or fine-tuning.

Pruning [31]–[34], quantization [34]–[36], and knowledge distillation (KD) [37] are the three most prevalent model compression techniques currently leading the advancement of EdgeLLMs. Although these techniques predate the development of large language models (LLMs), they have proven to be highly effective, either in their original forms or with minor modifications, in facilitating the deployment of EdgeLLMs. In the following sections, we will explore the state-of-the-art advancements in each of these compression techniques, providing an in-depth look at how they contribute to optimizing and enhancing EdgeLLM performance.

1) Pruning: Pruning is the concept of removing weights or neurons in a neural network that have the least impact on the model's output. This technique aims to reduce the complexity and size of the model without significantly affecting its performance, thereby decreasing the computational resources required for inference. One of the earliest works in this domain [31] introduced the idea of pruning by addressing overfitting. The authors measured the importance of each weight using second-order approximations, specifically leveraging the Hessian matrix to estimate the sensitivity of the loss function to each weight. By identifying and deleting the least significant weights, they successfully reduced the size of a digit recognition model by a factor of four while slightly improving its accuracy. However, applying such techniques to Large Language Models (LLMs) is computationally expensive due to the necessity of calculating the full Hessian matrix, which scales quadratically with the number of parameters.

To quantify the impact of pruning a specific weight w_i on the loss function L, Optimal Brain Damage (OBD) employs a second-order Taylor series expansion around the current weight values. The loss function can be approximated as:

$$\Delta L \approx \frac{\partial L}{\partial w_i}\Delta w_i + \frac{1}{2}\frac{\partial^2 L}{\partial w_i^2}\Delta w_i^2 \tag{4}$$

At an optimal point, the first derivative $\frac{\partial L}{\partial w_i}$ is nearly zero, simplifying the approximation to:

$$\Delta L \approx \frac{1}{2}\frac{\partial^2 L}{\partial w_i^2}\Delta w_i^2 \tag{5}$$

Here, $\frac{\partial^2 L}{\partial w_i^2}$ represents the second derivative of the loss function with respect to the weight w_i, indicating the curvature of the loss landscape. This approximation allows OBD to prioritize the removal of weights that contribute least to the loss, thereby achieving model sparsity with minimal degradation in performance. Optimal Brain Surgeon (OBS) extends this approach by considering the interactions between different weights through the full Hessian matrix, providing a more accurate estimation of the loss impact when multiple weights are pruned simultaneously. By leveraging OBD and OBS, researchers can effectively compress LLMs, making them more suitable for deployment on resource-constrained edge devices without significantly compromising their linguistic and inferential capabilities.

Furthermore, advanced pruning techniques have been developed specifically for LLMs like oBERT. Michel et al. (2019) [38] and Voita et al. (2020) [39] explored the pruning of multi-head self-attention mechanisms in transformer-based models.

Michel et al. investigated whether all attention heads contribute equally to the model's performance, discovering that many heads can be pruned without significant loss in accuracy. Their findings suggested that a substantial number of attention heads are redundant, allowing for more efficient models without degrading performance. Similarly, Voita et al. questioned the necessity of intermediate layers in BERT and demonstrated that pruning specific attention heads does not adversely affect the model's capabilities, thereby reducing model complexity and enhancing efficiency.

Sridhar and Sarah (2020) [40] introduced movement pruning, an adaptive sparsity technique that dynamically prunes less important weights during the fine-tuning process. Unlike static pruning methods, movement pruning continuously evaluates the importance of weights based on their updates (i.e., movement) during training. This dynamic approach allows for more nuanced weight removal, leading to highly sparse yet performant models suitable for deployment on edge devices. Sajjad et al. (2020) [41] extended the concept of movement pruning by incorporating it into transformer-based architectures, focusing on maintaining model performance while achieving high levels of sparsity. Their work demonstrated that movement pruning could effectively reduce the number of active parameters without compromising the model's ability to understand and generate language.

Lagunas et al. (2021) [42] proposed block pruning for faster transformers, which involves pruning contiguous blocks of weights rather than individual weights or entire heads. This method not only reduces the model size but also optimizes memory access patterns, leading to significant speedups during inference. By targeting blocks of weights, Lagunas et al. were able to maintain structural integrity within the model, ensuring that the pruned models remained robust and efficient. Sanh et al. (2020) [43] further refined movement pruning by integrating it with fine-tuning processes tailored for transformer-based models. Their approach emphasized the importance of adaptive sparsity, allowing the model to retain critical weights while discarding those that contribute least to performance. This balance between sparsity and accuracy makes movement pruning particularly effective for deploying LLMs on resource-constrained edge devices.

Lastly, [34] introduced a comprehensive three-stage compression pipeline—pruning, quantization, and Huffman coding—that enabled the deployment of deep neural networks on sub-100MB mobile applications. This pipeline not only reduced the model size significantly but also maintained accuracy, making it highly suitable for edge deployments where storage and computational resources are limited. By systematically applying these compression techniques, Han et al. demonstrated the feasibility of deploying sophisticated models on mobile devices without incurring substantial performance losses.

2) Quantization: Another technique used in model compression is quantization, the reduction of the precision of the weights and activations. Reducing the memory and compute requirements of the model without significantly reducing the performance. It is widely applied to deploy models in resource constrained environments, and it has proven to be effective with LLMs. This section will explore various implementations of quantization in compressing LLMs and highlight significant findings.

In the context of deep learning, one of the earliest papers to explore quantization of models is [34], which was discussed in the pruning section of this survey, and since then, there are more and more papers expanding and applying to the latest models. In the article [35], authors discuss making inference faster by storing the weights and inputs as integers instead of floating points with a tradeoff between accuracy and latency and compute resources. They map the quantized integers, q to respective real numbers, r with the formula:

$$r = S(q - Z) \tag{6}$$

Where S and Z are quantization parameters representing scale and zero-point respectively. The authors propose the weights and activatios quatized into 8 bits integer and then accumulated as int32, with the bias terms also at int32 to allow for larger values. They tested their approach with CNN models and they could reduce the latency by half while staying within 1-2% of the floating point accuracy.

Although written to explore quantization in the context of Convolutional Neural Networks, the paper [44] explores the different schemes of quantization which can be transferred into LLMs. The paper differentiated between Post Training Quantization and Quantization Aware training. The paper explores, symmetric or asymmetric layer wise, channel-wise or activation only quantization and based on their experiment, the most accurate was the Activation only quantization while speeding up the inference with $2\times$-$3\times$ speed. The authors in [45] go even further to propose 4 bit quantization of weights and activations with scale α optimized during training, with their proposed activation method outperforming 4-bit quantized ReLU .

In [46], they identify the issue of approximation-based quantization, with mismatch happening between the forward and backward passes. The authors then went ahead to formulate a differentiable non-linear quantization function to tackle this issue enabling them to achieve lossless results with only 3-bit quantization in ResNet-18. Not all the weights in deep neural networks (DNNs) are equally important as seen in the pruning section, and therefore for some weights we can go with the lowest precision possible while for some, we try to keep the maximum information. The paper [47] proposes mixed precision (MP) quantization where he eigenvalues of the hessian of the weights per each layers are calculated and precision is allotted proportionally with the eigenvalues. This methodology could compress ResNet-20 model weights by around $13\times$, and weights by x$\times$ while achieving similar performance.

In the context of LLMs quantization is playing a significant role in compressing huge models to save resources and enable language models at the edge. In [48], a procedure is developed allowing the multiplication of feed forward and attention matrices in Int8 instead of 32-bit floating point. Their procedure includes MP for outlier values which they empirically found they are less than 0.01. Matrices are decomposed into int8 and FP16 for the outliers giving out two matrix multiplications, which are then dequantized and accumulated after the Matrix multiplication process is complete. Although the traditional quantization of LLMs with Int8 fails for large models, the methodology suggested by the authors could perform with minimal additional perplexity and reducing the memory required by half.

Using the the concepts of OBD and OBS, [49] goes futher to quantize models into 3-bit and 4-bit values post training. In thier procedure, they apply layer wise quantization solving for:

$$argmin_{\hat{W}} \left\| WX - \hat{W}X \right\|_2^2 \tag{7}$$

Where $\hat{W}$ is the quantized sets. To minimize this equation, the paper refers to an approach shown in the authors' previous work, [50], which tries to minimize each row of the weight matrix after estimating the change in the objective function utilizing the second order derivatives.

SmoothQuant, proposed in [51] explains the challenges with quantizing activations, caused by outliers [48]. The *smoothing* in SmoothQuant is the scaling down the outlier activations at the cost of slightly increasing the size of the weights making them a bit harder and the activations possible to quantize, and through their experiments proved that they could speed up inference by 1.56× and reduce the memory use by half while maintaining accuracy across benchmarks. [52] is another paper making use of the idea that only few of the activations are salient. Their approach is to quantize the weights with MP, depending on the size of the activation, where the weights corresponding to the most important 0.1-3% of the weights are kept at FP16 while the other weights are quantized, and their result demonstrated that this approach was better than weight only quantization.

[53] takes the whole quantization process to the extreme by setting all the weights in the model as -1, 0, 1 they named **BitNet b1.58**. The matrix multiplication in this process is replaced by assigning signs and addition, removing the nead to scale the activations, resulting in up to 3.55× reduction in memory use, up to 2.71× reduction in latency, and surprisingly slight improvement in perplexity compared to LLAMA 3.9B. A more in-depth exploration of quantized LLMs is discussed in the article [54] titled "Evaluating Quantized Large Language Models"

3) Knowledge Distillation, KD: is a way of compressing a model to a smaller size allowing for smaller storage, memory, compute and energy requirement to produce similar results as the larger and more complicated models, distilling the knowledge from the larger (teacher model) to the smaller (student model). The transfer of knowledge in the case of KD is done to harness the learning advantages of a Larger Model and the generalization capabilities and speed of a smaller model. A larger more sophisticated model is trained and a smaller model is trained on the probability distribution (soft target) predicted by the larger model [37], which is calculated as:

$$p(_i, T) = \frac{exp(_i/T)}{\sum_j exp(_i/T)} \tag{8}$$

The distillation loss in the case of vanilla KD is:

$$L(_t, _s) = \mathcal{L}(p(_t, T), p(_s, T)) \tag{9}$$

Where T is the temperature hyperparameter, and the loss $\mathcal{L}(p(_t, T), p(_s, T))$ is generaly Kullback Lieber divergence loss [55].

Trying to match the probability distribution of the teacher model's and the student models output might work well with DNNs for classification tasks, however with LLM it gets more complicated and the authors in [56] tried to group KD in LLM based on the type of knowledge extracted from the LLMs. Eliciting knowledge from labling tasks is one of their examples and is more or less similar to the process expressed in the previous equation with the addition of Information(I) and Demonstrations (c) in order to feed richer information to the LLM described as:

$$\mathcal{D} = \{x, y | x \sim \mathcal{X}, y \sim p_T(y | I \oplus c \oplus x)\} \tag{10}$$

Where $\mathcal{D}$ stands for the data set used for distillation and $\oplus$ is the the combining of two chunks of text. However, the vast capability of LLMs makes it harder to have a set of labeled data and effectively distill the performance of the teacher to the student. Approaches such as enlarging datasets and enhancing data curation have become prominent methodologies to address this issue. Another way of passing knowledge discussed in the survey was the learning of features instead of outputs [56]–[58].

$$\mathcal{D}^{(feat)} = \{(x, y, \phi_{feat}(x, y; \theta_T)) | x \sim \mathcal{X}, y \sim \mathcal{Y})\} \tag{11}$$

Similar to the output probability distribution, the loss is measures as Kullback-Lieber divergence Loss.

One prominent example of a student language model that has distilled knowledge from a larger model is DistilBERT [59], which is 40% the size of BERT model, 60% faster and retains 97% accuracy. The paper used soft target probabilities described in the introduction of KD with temperature $T \neq 1$ applied during the distillation process.

Another LLM distillation example is [60], which reports 96.8% performance of $BERT_{BASE}$. This model has four layers and is 7.5× smaller and 9.4× faster. The authors use layer wise knowledge distillation, e.g., an embedding layer's "knowledge" is distilled into an embedding layer in the student model. The loss is then calculated as:

$$\mathcal{L}_{model} = \sum_{x \in \mathcal{X}} \sum_{m=0}^{M+1} \lambda_m \mathcal{L}_{layer}(^S_m(x), ^T_{g(m)}(x)) \tag{12}$$

Where $\mathcal{L}_{layer}$ is the loss for a given model layer, and $m, g(m)$ are teh layer in the teacher model and it mapping in the student model respectively. The authors also provided specialized formulas for each layer types.

In [58], the authors classified LLM KD into black-box KD and white-box KD based on the availability of the hidden states. The authors differentiated that the normal labeling or classification models with a finite set of potential outputs, with the probabilities easily summed to 1, and LLM outputs with so many possible output. After they discussed the limitations of the conventional Kullback Leiber Divergence (KLD), they developed a distilled model utilizing reverse KLD, explained in [61], which yielded a better result than the common forward KLD in LLMs.

Knowledge Distillation (KD) compresses large models into smaller, efficient ones without losing significant performance. This approach, highlighted in notable LLM distillation papers, is key to enabling efficient and high-performing EdgeLLMs in resource-constrained environments.

IV. Private data and EdgeLLMs

LLMs are data hungry and the amount of publicly available data is limited [4]. Further training therefore is going to be reliant on private data. EdgeLLM, applying the principles of Federated Learning therefore is going to be a potential enabler to improved LLM training and fine-tuning with the increased available private data. In this section, we are going to see recent works in security and privacy preserving EdgeLLM techniques.

LLMs are susceptible to a range of security and privacy attacks. The survey [62] mentions a range of attacks like extracting personal information from sensitive queries, backdoor (data poisoning), membership inference, and model inversion attacks. Although they are different types of attacks and have different sets of solutions, EdgeLLM can be a solution, through the local storage of weights and processing of prompts. Removing the potential of query leakages and keeping your data at the edge and only training through federated learning excludes the potential of a backdoor during fine-tuning.

The paper [62] argues Federated training might not be enough to tackle the privacy issues related to extracting information from the model. The authors suggest EW-Tuning, a differential privacy (DP) technique that adds the minimum amount of noise to be added to SGD to hide private information, providing a guarantee of privacy in the case of Federated EdgeLLM training.

In the case of smaller edge devices, where the resources would not allow for back-propagation, the paper [63] suggests federated prompt tuning, a scheme instead of updating the pre-trained model, learns the optimal prompts in a federated manner, allowing for privacy preserving way of training federated EdgeLLMs. When training a federated model with DP, the private data might not be sufficient and using the public data might be necessary. Another work in [64] takes the same approach of federated prompt generation where instead of fine-tuning the model in a federated manner, they focus on federated prompt generation utilizing the users on device information to extract more relevant knowledge from LLMs. Opposed to storing the LLM locally, this approach only focuses on device prompt optimization, efficient personalized prompt generation can be used in developing robust EdgeLMs.

The authors in [65] explored the potential use of existing LLMs and the publicly available data to assist in the training of an edgeLM. The authors proposed the use of KD, and a portion of public data to improve accuracy, and explained using tokenizers from existing public LLMs would help in curbing information leakage. With the use of a novel data distribution matching algorithm, and selecting around 0.08% of the publicly available data most similar to the private data, the models developed could outperform the models trained without the public data. Another finding was with knowledge distillation and utilizing 1% of the public data, the private federated training could achieve similar results as utilizing a 100% of the public data with without KD.

With the concepts of federated learning and differential privacy, LLM training can be exposed to even larger amount of data that currently used in the current pre-trained models. EdgeLLM can make this privacy preserving scheme a reality and a pathway for the everyday user data towards stronger language models.

V. EdgeLLM applications

Compressing and fine-tuning language models can allow on edge highly specialized task specific language models to be accessed with low latency and allowing to keep your prompts with you [6]. However, EdgeLLMs can do more and even be part of a larger ecosystem. EdgeLLMs are increasingly being integrated into various sectors, including the Metaverse, networking, and Internet of Things (IoT) solutions. In [66], the author explores the use of LLMs in networking for planning, analyzing, calculating, and executing networking-related tasks with the goal of reducing human intervention into the processes. The author also compares LLM techniques like zero-shot, few-shot, RAG and chain of thought techniques. Networking devices equipped with such tools would enable more detailed and pervasive analysis and surveillance tools. The article [13] discusses the role of LLMs in 6G networking to enable shared multimodal perception, interactive grounding, and alignment. In their framework, LLMs were deployed at three stages, at the cloud, at the end devices and the middle network devices. Both [67], [68] explore EdgeLLMs in the context of the Metaverse. The Metaverse is an immersive environment with the largest issues facing it being content generation, scalability and latency. GenAI can be a potential solution addressing the generation of content. Both papers demonstrate AI generated content AIGC can be a potential solution to content generation, scalability and curbed latency with the help of EdgeLLMs. Another area of EdgeLLMs could be the IoT where there are multiple devices with different compute capacity, tasks and environment they are exposed to. The papers [69], [70] suggest the use of egdge deployed LLMs to create a synchronized system where AI generated knowledge is used to manage and run ecosystems. Mixture of experts for example is used to effectively plan actions and smaller tasks are assigned to experts to solve problems in a divide and conquer manner [69].

VI. Discussion

This survey has provided an extensive overview of the latest advancements in the development and deployment of Language Models at the edge (EdgeLLMs). We have explored various techniques for compressing and optimizing large language models to make them feasible for edge computing environments. From resource-efficient training and fine-tuning methods to inference optimizations like pruning, quantization, and knowledge distillation. Each approach plays a crucial role in enabling the deployment of LLMs on resource-constrained devices.

The significance of EdgeLLMs extends beyond mere technical achievements. By enabling real-time, on-device language processing, these models offer substantial benefits, including reduced latency, improved privacy, and enhanced security, particularly in applications involving sensitive or private data. The potential for EdgeLLMs to be integrated into a wide array of applications, from IoT ecosystems to the Metaverse, highlights their transformative impact on the future of AI deployment.

Despite the remarkable progress made in this field, several challenges remain. Issues related to model accuracy, resource constraints, and security must be addressed to fully realize the potential of EdgeLLMs. Furthermore, the field is still evolving, with ongoing research needed to refine existing techniques and explore new avenues, such as federated learning and differential privacy, to enhance the robustness and applicability of these models.

Looking ahead, future research should focus on overcoming these challenges by developing more efficient and secure methods for EdgeLLM deployment. By addressing these issues, the research community can pave the way for more accessible and effective AI solutions that benefit a broader range of users and applications.

In conclusion, the landscape of EdgeLLMs is both dynamic and promising. As research continues to advance, we anticipate that these models will play an increasingly central role in the deployment of AI across a variety of domains, offering new opportunities and driving further innovation in the field.

Acknowledgements

This research was supported by the Technology Innovation Institute (TII), Abu Dhabi, UAE, under the CyberAI project (grant number: TII/DSRC/2022/3036).

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[2] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49313245

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[4] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," 2020. [Online]. Available: https://arxiv.org/abs/2001.08361

[5] M. AI, "Meta ai: Introducing llama 3," https://ai.meta.com/blog/meta-llama-3/, 2024.

[6] Z. Zhou, L. Li, X. Chen, and A. Li, "Mini-giants: "small" language models and open source win-win," 2024. [Online]. Available: https://arxiv.org/abs/2307.08189

[7] Y. Jin, J. Li, Y. Liu, T. Gu, K. Wu, Z. Jiang, M. He, B. Zhao, X. Tan, Z. Gan, Y. Wang, C. Wang, and L. Ma, "Efficient multimodal large language models: A survey," 2024.

[8] Z. Wan, X. Wang, C. Liu, S. Alam, Y. Zheng, J. Liu, Z. Qu, S. Yan, Y. Zhu, Q. Zhang, M. Chowdhury, and M. Zhang, "Efficient large language models: A survey," *Transactions on Machine Learning Research*, 2024, survey Certification. [Online]. Available: https://openreview.net/forum?id=bsCCJHbO8A

[9] C. White, M. Safari, R. Sukthanker, B. Ru, T. Elsken, A. Zela, D. Dey, and F. Hutter, "Neural architecture search: Insights from 1000 papers," 2023. [Online]. Available: https://arxiv.org/abs/2301.08727

[10] G. Qu, Q. Chen, W. Wei, Z. Lin, X. Chen, and K. Huang, "Mobile edge intelligence for large language models: A contemporary survey," 2024. [Online]. Available: https://arxiv.org/abs/2407.18921

[11] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," 2019. [Online]. Available: https://arxiv.org/abs/1902.04885

[12] C. Dwork, "Differential privacy," in *Proceedings of the 3rd Theory of Cryptography Conference (TCC 2006)*, Springer. Berlin, Heidelberg: Springer, 2006, pp. 1–12.

[13] K. B. Letaief, Y. Shi, J. Lu, and J. Lu, "Edge artificial intelligence for 6g: Vision, enabling technologies, and applications," 2021. [Online]. Available: https://arxiv.org/abs/2111.12444

[14] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2023. [Online]. Available: https://arxiv.org/abs/1602.05629

[15] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[16] S. Warnat-Herresthal, H. Schultze, K. L. Shastry, S. Manamohan, S. Mukherjee, V. Garg, R. Sarveswara, K. Händler, P. Pickkers, E. Azoulay *et al.*, "Swarm learning for decentralized and confidential clinical machine learning," *Nature*, vol. 594, no. 7862, pp. 265–270, 2021.

[17] A. H. Sayed, S.-Y. Tu, J. Chen, X. Zhao, and Z. J. Towfic, "Diffusion strategies for adaptation and learning over networks: an examination of distributed strategies and network behavior," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 155–171, 2013.

[18] M. Li, D. G. Andersen, A. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, p. 19–27.

[19] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," 2018. [Online]. Available: https://arxiv.org/abs/1810.06060

[20] L. Sani, A. Iacob, Z. Cao, B. Marino, Y. Gao, T. Paulik, W. Zhao, W. F. Shen, P. Aleksandrov, X. Qiu, and N. D. Lane, "The future of large language model pre-training is federated," 2024. [Online]. Available: https://arxiv.org/abs/2405.10853

[21] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[22] X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: a deep learning approach," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11. Madison, WI, USA: Omnipress, 2011, p. 513–520.

[23] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. de Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," 2019. [Online]. Available: https://arxiv.org/abs/1902.00751

[24] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," 2018. [Online]. Available: https://arxiv.org/abs/1801.06146

[25] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" 2014. [Online]. Available: https://arxiv.org/abs/1411.1792

[26] S. Malladi, T. Gao, E. Nichani, A. Damian, J. D. Lee, D. Chen, and S. Arora, "Fine-tuning language models with just forward passes," 2024. [Online]. Available: https://arxiv.org/abs/2305.17333

[27] W. Guo, J. Long, Y. Zeng, Z. Liu, X. Yang, Y. Ran, J. R. Gardner, O. Bastani, C. D. Sa, X. Yu, B. Chen, and Z. Xu, "Zeroth-order fine-tuning of llms with extreme sparsity," 2024. [Online]. Available: https://arxiv.org/abs/2406.02913

[28] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021. [Online]. Available: https://arxiv.org/abs/2106.09685

[29] Z. Lin, A. Madotto, and P. Fung, "Exploring versatile generative language model via parameter-efficient transfer learning," 2020. [Online]. Available: https://arxiv.org/abs/2004.03829

[30] A. Rücklé, G. Geigle, M. Glockner, T. Beck, J. Pfeiffer, N. Reimers, and I. Gurevych, "Adapterdrop: On the efficiency of adapters in transformers," 2021. [Online]. Available: https://arxiv.org/abs/2010.11918

[31] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proceedings of the 2nd International Conference on Neural Information Processing Systems*, ser. NIPS'89. Cambridge, MA, USA: MIT Press, 1989, p. 598–605.

[32] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015. [Online]. Available: https://arxiv.org/abs/1506.02626

[33] B. Hassibi, D. Stork, and G. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE International Conference on Neural Networks*, 1993, pp. 293–299 vol.1.

[34] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2016. [Online]. Available: https://arxiv.org/abs/1510.00149

[35] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," 2017. [Online]. Available: https://arxiv.org/abs/1712.05877

[36] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," 2016. [Online]. Available: https://arxiv.org/abs/1609.07061

[37] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. [Online]. Available: https://arxiv.org/abs/1503.02531

[38] P. Michel, O. Levy, and G. Neubig, "Are sixteen heads really better than one?" in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[39] E. Voita, D. Talbot, F. Moiseev, R. S. N. Sridhar, and A. Sarah, "Undivided attention: Are intermediate layers necessary for bert?" *arXiv preprint arXiv:2012.11881*, 2020.

[40] R. Sridhar and T. Sarah, "Movement pruning: Adaptive sparsity by fine-tuning," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 20 378–20 389.

[41] H. Sajjad *et al.*, "Movement pruning for transformer-based models," in *Conference Name*, Organization. Location: Publisher, 2020, p. Page Numbers.

[42] F. Lagunas, E. Charlaix, V. Sanh, and A. Rush, "Block pruning for faster transformers," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2021, pp. 10 619–10 629.

[43] V. Sanh, T. Wolf, and A. Rush, "Movement pruning: Adaptive sparsity by fine-tuning," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 20 378–20 389.

[44] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018. [Online]. Available: https://arxiv.org/abs/1806.08342

[45] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "Pact: Parameterized clipping activation for quantized neural networks," 2018. [Online]. Available: https://arxiv.org/abs/1805.06085

[46] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X. Hua, "Quantization networks," 2019. [Online]. Available: https://arxiv.org/abs/1911.09464

[47] Z. Dong, Z. Yao, A. Gholami, M. Mahoney, and K. Keutzer, "Hawq: Hessian aware quantization of neural networks with mixed-precision," 2019. [Online]. Available: https://arxiv.org/abs/1905.03696

[48] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Llm.int8(): 8-bit matrix multiplication for transformers at scale," 2022. [Online]. Available: https://arxiv.org/abs/2208.07339

[49] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," 2023. [Online]. Available: https://arxiv.org/abs/2210.17323

[50] E. Frantar, S. P. Singh, and D. Alistarh, "Optimal brain compression: A framework for accurate post-training quantization and pruning," 2023. [Online]. Available: https://arxiv.org/abs/2208.11580

[51] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," 2024. [Online]. Available: https://arxiv.org/abs/2211.10438

[52] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for llm compression and acceleration," 2024. [Online]. Available: https://arxiv.org/abs/2306.00978

[53] S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, and F. Wei, "The era of 1-bit llms: All large language models are in 1.58 bits," 2024. [Online]. Available: https://arxiv.org/abs/2402.17764

[54] S. Li, X. Ning, L. Wang, T. Liu, X. Shi, S. Yan, G. Dai, H. Yang, and Y. Wang, "Evaluating quantized large language models," 2024. [Online]. Available: https://arxiv.org/abs/2402.18158

[55] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, p. 1789–1819, Mar. 2021. [Online]. Available: http://dx.doi.org/10.1007/s11263-021-01453-z

[56] X. Xu, M. Li, C. Tao, T. Shen, R. Cheng, J. Li, C. Xu, D. Tao, and T. Zhou, "A survey on knowledge distillation of large language models," 2024. [Online]. Available: https://arxiv.org/abs/2402.13116

[57] I. Timiryasov and J.-L. Tastet, "Baby llama: knowledge distillation from an ensemble of teachers trained on a small dataset with no performance penalty," 2023. [Online]. Available: https://arxiv.org/abs/2308.02019

[58] Y. Gu, L. Dong, F. Wei, and M. Huang, "Minillm: Knowledge distillation of large language models," 2024. [Online]. Available: https://arxiv.org/abs/2306.08543

[59] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2020. [Online]. Available: https://arxiv.org/abs/1910.01108

[60] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "Tinybert: Distilling bert for natural language understanding," 2020. [Online]. Available: https://arxiv.org/abs/1909.10351

[61] T. Minka *et al.*, "Divergence measures and message passing," Technical report, Microsoft Research, Tech. Rep., 2005.

[62] B. Yan, K. Li, M. Xu, Y. Dong, Y. Zhang, Z. Ren, and X. Cheng, "On protecting the data privacy of large language models (llms): A survey," 2024. [Online]. Available: https://arxiv.org/abs/2403.05156

[63] J. Sun, Z. Xu, H. Yin, D. Yang, D. Xu, Y. Chen, and H. R. Roth, "Fedbpt: Efficient federated black-box prompt tuning for large language models," 2023. [Online]. Available: https://arxiv.org/abs/2310.01467

[64] F.-E. Yang, C.-Y. Wang, and Y.-C. F. Wang, "Efficient model personalization in federated learning via client-specific prompt generation," 2023. [Online]. Available: https://arxiv.org/abs/2308.15367

[65] B. Wang, Y. J. Zhang, Y. Cao, B. Li, H. B. McMahan, S. Oh, Z. Xu, and M. Zaheer, "Can public large language models help private cross-device federated learning?" 2024. [Online]. Available: https://arxiv.org/abs/2305.12132

[66] Y. Huang, H. Du, X. Zhang, D. Niyato, J. Kang, Z. Xiong, S. Wang, and T. Huang, "Large language models for networking: Applications, enabling techniques, and challenges," 2023. [Online]. Available: https://arxiv.org/abs/2311.17474

[67] M. Xu, D. Niyato, H. Zhang, J. Kang, Z. Xiong, S. Mao, and Z. Han, "Sparks of gpts in edge intelligence for metaverse: Caching and inference for mobile aigc services," 2023. [Online]. Available: https://arxiv.org/abs/2304.08782

[68] G. Liu, H. Du, D. Niyato, J. Kang, Z. Xiong, A. Jamalipour, S. Mao, and D. I. Kim, "Fusion of mixture of experts and generative artificial intelligence in mobile edge metaverse," 2024. [Online]. Available: https://arxiv.org/abs/2404.03321

[69] J. Wang, H. Du, D. Niyato, J. Kang, Z. Xiong, D. I. Kim, and K. B. Letaief, “Toward scalable generative ai via mixture of experts in mobile edge networks,” 2024. [Online]. Available: https://arxiv.org/abs/2402.06942

[70] Y. Liu, H. Du, D. Niyato, J. Kang, S. Cui, X. Shen, and P. Zhang, “Optimizing mobile-edge ai-generated everything (aigx) services by prompt engineering: Fundamental, framework, and case study,” 2023. [Online]. Available: https://arxiv.org/abs/2309.01065