Learning Diary of Edge Intelligence

1 Introduction

Edge computing has gained popularity due to the necessity of extracting valuable insights from overflowing information in real time. It can be defined as an architecture where data is processed as close as possible to the source. Edge computing is also referred to as a distributed computing framework for capturing, storing, processing, and analyzing data closer to its location. As a result of this approach, cloud services have shifted from the network core to the network edges. Following the rise of edge computing, we are now observing the rise of a combination of Edge Computing and Artificial Intelligence (AI), which is called Edge Intelligence.

2 Course Learning

When I went through the learning outcomes and content of the seminar course Edge Intelligence, I found that the course wasn't about edge computing alone. It focused on both local data processing frameworks to enhance real-time decision-making capabilities and the integration techniques of contemporary Large Language Models (LLMs) and other AI models to enhance the power of Edge Intelligence.

Thus, when I chose this course, I had multiple reasons in mind. I wanted to explore recent advancements in Edge Computing, the AI models and techniques incorporated into Edge Intelligence, the architecture and implementation of LLMs, and how AI and LLM models are contributing to the paradigm shift in Edge Intelligence. Additionally, I intended to develop some of my skills in presentation and communication. At the same time, I had just completed an internship at Ericsson in network security. As a result, I was also interested in exploring the security aspects of edge computing and the security and privacy issues of AI models used in Edge Intelligence.

I believe the course provided me with knowledge of two distinct fields. The first one enriched my understanding of the domain of Edge Intelligence, while the second one helped me strengthen my various personal skills. I studied multiple materials, including five specific research papers, and had a total of four sessions to present my own interpretation of those papers. The presentation slide template was open-ended, and we were required to study, analyze, and evaluate selected research papers for the presentation. We needed to represent our perception in a way that made the content understandable to everyone. We were also tasked with creating quizzes for the audience to assess their understanding of the material.

During the sessions, my classmates could ask questions during or after the presentation if anything was unclear. I believe this two-way communication significantly improved my presentation skills as well as my verbal communication abilities. In these sessions, I not only had to answer questions but also ask questions during others presentations. The four sessions boosted my confidence in presenting research work in front of an audience in a limited time frame. I learned how to compose thoughtful questions for others and how to respond to questions spontaneously in simple, clear language, which enhanced my collaboration skills. I learned how to evaluate and construct an effective presentation of information derived from research papers. I also gained a deeper understanding of the advanced and recent technologies in Edge Intelligence, along with the fundamental concepts of both LLMs and edge computing. Now, I am familiar with the implementation, challenges, and opportunities associated with

Edge Intelligence. This course helped me structure how I read and understand research papers and create high-quality presentations. Edge Intelligence is a vast field, but my fellow course participants and I discussed its challenges, implementations, applications, and the most recent studies. We covered diverse topics and engaged in effective discussions during the lecture sessions. In conclusion, I can say that our knowledge of both basic and advanced topics related to Edge Intelligence has significantly developed and increased.

As I am currently working on my thesis, which is related to network security and AI, the insights gained about AI implementation in edge security have helped me scale up my ideas in 5G security. For example, before taking this course, I was familiar with using LLMs and had a basic understanding of their underlying concepts. However, after presenting the structure of the transformer model in LLMs, I now have a detailed step-by-step understanding of how LLMs function, including their algorithms and workflows. I have also learned how edge computing has evolved into Edge Intelligence. In the following sections, I will describe the main concepts of the research papers I presented. After studying these papers, I can now identify the implementations, opportunities, and both past and future developments in the field of Edge Intelligence.

3 Topics Addressed in the Research Papers

In the following sections, I will describe the main concepts of the research papers I presented. After studying these papers, I can now identify the implementations, opportunities, and both past and future developments in the field of Edge Intelligence.

- **Presentation 1**

  First, I will review the papers from my first presentation, Communication-Efficient Learning of Deep Networks from Decentralized Data and Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge.

  - **Paper 1 [1]**

    The first paper introduced Federated Learning (FL) as a decentralized approach for utilizing AI models, where training data remains secure on mobile devices. FL was presented as a distributed machine learning algorithm where training is performed locally, and then a global model is updated. The authors' primary objective was to achieve optimization by minimizing the loss function. To protect privacy and reduce communication costs, the paper also highlighted several key challenges of FL, such as unbalanced datasets, massively distributed data, and limited communication among clients.

    The authors proposed a novel model averaging technique and compared its results with synchronized stochastic gradient descent. Their experimental setup considered practical constraints, such as minimizing communication costs, ensuring client participation only when devices were charged, plugged in, or connected to Wi-Fi, and accounting for the limited size of on-device datasets. The experiments were conducted in a controlled environment, ignoring challenges like client unavailability. Their work plan consisted of three steps: improving aggregation techniques to handle non-IID and unbalanced data, performing more complex computations on clients to reduce the number of communication rounds, and engaging more clients in parallel during each round to decrease communication frequency.

    Four formulas were implemented where they had two basic federated averaging algorithms and two updated versions, each with one formula for client-side gradient computation and another for server-side aggregation. The experiments addressed four objectives: examining whether reducing communication frequency impacts performance, analyzing the effects of parallelism by

increasing the number of clients, studying the effect of increasing local epochs and varying mini-batch sizes, and comparing the performance of basic and updated averaging algorithms. The authors conducted experiments on the MNIST and CIFAR-10 datasets for image classification using models such as a Two-Layer Neural Network (2NN), a Convolutional Neural Network (CNN), and a smaller CNN (based on a TensorFlow tutorial). For language modeling, they used the Shakespeare dataset and the Social Network Text Dataset with Stacked Character-Level LSTMs.

Key observations included that increasing the client fraction improves convergence rates, especially with smaller mini-batch sizes. Smaller batch sizes and more participating clients enhanced generalization and convergence rates in non-IID scenarios. Adding more local SGD updates per client per round helped achieve target accuracy while reducing communication rounds. FedAvg outperformed FedSGD by achieving superior test accuracy, reducing communication rounds, and handling non-IID data effectively.

The authors concluded that FedAvg is robust in handling pathological, non-IID data with unbalanced roles and minimal divergence. It significantly reduces communication rounds compared to FedSGD. However, increasing local computation (larger E and smaller B) must be balanced to avoid plateauing or divergence in specific tasks.

– **Paper 2 [2]**

The authors of paper 2 introduced FedCS to improve FLs efficiency by managing heterogeneous client resources in a cellular network. This paper directly addressed unbalanced users and limited communication, while indirectly addressing the issue of massively distributed data. FedCS tackled challenges such as setting deadlines for model activities (model download, update, and upload) and reducing machine learning model training time by optimizing client participation. Similar to the first paper, this study made assumptions regarding the presence of a server and base station to manage the FL process, limited resource blocks for bandwidth sharing, optimized modulation and coding schemes for radio communication, and stable channel states and throughput.

The FedCS protocol worked in multiple steps. After initializing the global model, the server sent resource requests to selected clients to inquire about their available resources. A greedy algorithm was then used to select clients with the least time required for model updates and uploads. The updating deadline was dynamic, and the authors suggested tuning it based on requirements. After the clients were selected, the global parameters were sent to them via multicast transmission. After the deadline, clients updated the model using local data and uploaded the updates. At the end, the server aggregated the results from all the selected clients and updated the global model.

The authors experimented with object classification in a simulated Mobile Edge Computing (MEC) setup. They did this with an edge server, a base station (BS), 1,000 clients, and the CIFAR-10 and Fashion-MNIST datasets. In the experiment, a Convolutional Neural Network (CNN) model was implemented with six convolutional layers and three fully connected layers. The model was trained using stochastic gradient descent with a mini-batch size of 50 and a learning rate of 0.025. Each training round had a specific time limit, which could vary from 1 to 10 minutes. Additionally, the total training time was 400 minutes.

The results showed that FedCS outperformed the baseline FedLim in both the CIFAR-10 and Fashion-MNIST experiments. Specifically, FedCS achieved 75% accuracy on CIFAR-10, 76.5 minutes faster than FedLim, and 85% accuracy on Fashion-MNIST, 33.3 minutes faster. FedCS also involved more clients per training round on average (7.7 clients), compared to FedLim (3.3 clients), which improved training efficiency.

FedCS continued to outperform FedLim, achieving higher accuracy levels across both datasets even in the challenging non-IID scenario. However, both methods showed reduced accuracy

and higher variance due to the non-IID data distribution.

- **Presentation 2**

  In this section, I will describe the contributions of two papers from my second and third presentations. The first paper is about the Transformer Network Architecture, which is widely known as the foundational architecture for LLMs. It proposed an entirely new approach to language sequence modeling, introducing the concept of attention. The authors demonstrated that this attention-based sequence analysis model outperforms traditional recurrent and convolutional neural networks.

  - **Paper 1 [3]**

    The transformer model architecture has two main parts: the input encoder and the output decoder, similar to the recurrent neural networks encoder and decoder. However, the Long Short-Term Memory (LSTM) architecture has some limitations, such as slower computation for long sequences, vanishing or exploding gradients, dependency on long-range data, and the inability to put attention on any specific part of the sequence. Therefore, in the transformer model, the authors emphasized incorporating attention to any part of the sequence. Both the encoder and decoder have similar functionalities but differ in some parts. In the input section, we have input embedding, and in the output section, we have output embedding.

    Although the names are the same, their working procedures are not identical. In the embedding step, each token or word in the sequence is converted into embedded vectors of a fixed size. Then, according to the position of the token in the sequence, positional encoding is performed using a specific positional calculation equation. There is a method to calculate the self-attention of a sequence, which requires three vectors: the first vector, Q (Query), represents the information we need to find from the input; the second vector, K (Key), represents the input content that can be matched against the queries; and the third vector, V (Value), holds the actual information needed to be extracted from the sequence. A weight vector, W, is calculated during the training phase of the transformer model. To calculate attention, the dot product of Q and the transpose of K is taken, followed by the softmax of the dot product, and then multiplied by V, divided by the square root of the determinant of K.

    In the input encoder, we have multi-head attention where each attention head focuses on different aspects. One head might focus on syntactic relationships, while another might capture semantic relationships. In the output decoder, we have masked multi-head attention, which means that the decoder only has access to the previous tokens and does not consider future tokens. In other words, although the calculation for the next word in the sequence is possible, the masked multi-head attention prevents access to the next word, focusing only on previous or existing tokens. Then, the Add & Norm function and the Feed Forward Network (FFN) of both the encoder and decoder are the same. In the Add & Norm part, the layer input is added to the layer output, and normalization is performed using the mean and standard deviation. In the feed-forward part, linear transformations are performed first, followed by the application of a non-linear activation function, and then another linear transformation.

    After input embedding, positional encoding, multi-head attention, the Add & Norm part, feed-forward, and another Add & Norm step, the output of the input encoder is passed to the input of the decoder. The output string or sequence shifts to the right, and the decoder receives the input as key-value pairs from the encoder. The output decoder processes the shifted sequence through output embedding, positional encoding, masked multi-head attention, Add & Norm, and then multi-head attention along with the key-value pairs from the encoder. The decoder uses input representations to compare and decide which input parts are most relevant for generating the next word. These hold the actual content of the input representation that the decoder can use to generate the next word.

These three inputs go to the multi-head attention part, followed by the Add & Norm section. Similar to the encoder, feed-forward computation is performed, followed by another Add & Norm step. In the last two stages, the Linear layer translates the high-dimensional representation into a probability distribution over all possible tokens in the vocabulary, and the Softmax function produces the desired output.

The authors discussed the performance of their attention model on two major machine translation tasks. The tasks are translating English to German and English to French. The model outperformed LSTMs with attention and CNNs in terms of both speed and accuracy. For the English-to-French task, the model achieved good performance after 3.5 days of training on 8 GPUs, compared to over 10 days for previous models. This improvement is attributed to the absence of recurrence and the reliance on self-attention, which allows for better parallelization and faster training. Additionally, the authors found that using six layers for both the encoder and decoder yielded optimal results.

– **Paper 2 [4]**

In the second paper, the author discussed prompt engineering in detail. He started with the basic definition of a prompt and then categorized prompts into different types to offer better understanding. The prompts can be instructions, questions, and input data. The explanation of prompt engineering included aspects like textual interfaces, context awareness, template creation, and iterative processes. How the multimodal applications can be incorporated into prompt engineering was also discussed. Various prompt designing techniques were highlighted, such as Chain of Thought, encouraging factual responses, explicitly ending prompt instructions, and usage of AI to correct itself. Advanced techniques like Chain of Thought with zero-shot and manual-shot prompting, and Tree of Thought, were also explained.

ART (Automatic Multi-step Reasoning and Tool) was defined as a combination of automated Chain of Thought prompting with the use of external tools. Self-consistency was described as prompting the LLM to produce multiple answers to the same question, with coherence among responses serving as a gauge for credibility. Expert prompting was introduced as leveraging the LLMs ability to generate nuanced answers by adopting the persona of relevant experts. Chains and Rails in advanced prompt engineering were differentiated, with Chains breaking tasks into distinct components and Rails providing structured outputs by setting boundaries for relevance, safety, and accuracy.

The study also covered Automatic Prompt Engineering, including prompt generation, scoring, refinement, and iteration. Retrieval-augmented generation was suggested as a method to enhance LLMs by addressing limitations in accessing real-time or specialized information. LLM agents such as Reasoning without Observation (ReWOO), which creates reasoning plans without external data, Reason and Act (ReAct), which enhances problem-solving capabilities, and Dialog-Enabled Resolving Agents (DERA), which involves multiple agents collaborating to resolve queries, were explained. Finally, the author discussed tools and frameworks related to prompt engineering, such as LangChain and NeMo Guardians by Nvidia.

4 Conclusion

Here I have tried to explain my learning and understanding after reading the research papers in short. I didn't use any equations or figures to simplify the explanation within the limited page restriction. In the end, this course helped me in various ways. While doing the thesis, I am experiencing the benefits.

References

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," *International Conference on Artificial Intelligence and Statistics*, pp. 1273 to 1282, Apr. 2017.

[2] T. Nishio and R. Yonetani, "Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge," *arXiv preprint arXiv:1804.08333*, Oct. 2018.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Å. Kaiser, and I. Polosukhin, "Attention Is All You Need," *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6000 to 6010, Dec. 2017.

[4] X. Amatriain, "Prompt Design and Engineering: Introduction and Advanced Methods," *arXiv preprint arXiv:2401.14423 [cs.SE]*, Jan. 2024, last revised May 2024. [Online]. Available: https://arxiv.org/abs/2401.14423

NB. GPT-4 has been used to correct the grammatical and syntactical errors. As English is my second language, I have used the model to restructure my sentences in some places