上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

BACHELOR'S THESIS

论文题目：Early-Exit Offloading for Embedded Question Answering Applications

学生姓名：刘翊华，陈烁丞，鞠易茗

学生学号：518021910998，517021911139，518370910059

专　　业：电子与计算机工程

指导教师：邹桉

学院(系)：密西根学院

上海交通大学
毕业设计（论文）学术诚信声明

本人郑重声明：所呈交的毕业设计（论文），是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者签名：

日期：2021 年 12 月 29 日

上海交通大学

毕业设计（论文）版权使用授权书

本毕业设计（论文）作者同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本毕业设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本毕业设计（论文）。

保密□，在___年解密后适用本授权书。

本论文属于

不保密√。

（请在以上方框内打“√”）

作者签名：　　　　指导教师签名：

日期：2021 年 12 月 29 日　　　　日期：2021 年 12 月 29 日

基于提前退出机制的针对嵌入式问答应用的计算卸载

摘要

自然语言处理技术是人工智能领域一个很重要的方向，近年来发展迅速。其中问题回答是自然语言处理技术下的一个分支，其模型表现甚至超过了人类自身。然而，为了达到出色的准确率，模型的参数数量通常会很大，甚至有几十亿个参数，这使得模型在训练和预测过程中需要占用大量的内存空间。与此同时，嵌入式系统由于其个性化强，高实时性等特点，在军事、医疗和工业等领域获得了广泛的关注和应用。但又由于内核小，内存小等原因，很难将自然语言处理技术加入到嵌入式终端。因此厂家通常将训练和预测的过程放在云端，嵌入式仅仅作为一个传送、接收数据并且呈现出来的载体。但是云计算又会有网络延迟以及信息安全等风险，因此我们提出了数据卸载的解决方法。我们将训练的过程放在云端，而通过预先存在本地训练好的模型进行预测。为了尽可能达到高性能和低功耗，我们进行了提前退出、剪枝以及自适应注意力等方法。同时我们还提出了将无法回答的问题存为新的数据集，并且在云端对模型进行微调，以达到更好的效果。最终我们将这个模型放入 NVIDIA Jetson TX2 开发板上进行测试，在性能和功耗上都得到了不错的效果。

关键词：自然语言处理，计算卸载，嵌入式系统，提前退出

EARLY-EXIT OFFLOADING FOR EMBEDDED QUESTION ANSWERING APPLICATIONS

ABSTRACT

Natural language processing (NLP) is an important field in artificial intelligence and has developed rapidly in recent years. The model of question answering, a branch of natural language processing, outperforms even humans. However, in order to achieve excellent accuracy, the number of parameters of the model is usually very large, even billions of parameters, which makes the model need to occupy a large amount of memory space in the process of training and prediction. At the same time, because of its strong personalized and high real-time characteristics, embedded system in the military, medical and industrial fields have been widely concerned and applied. However, due to the small kernel and small memory, it is difficult to add NLP technology into embedded terminal. Therefore, manufacturers usually put the training and prediction process in the cloud and embedded device is only as a carrier to transmit, receive and present data. However, cloud computing has risks such as network latency and information security, so we proposed a solution of data offloading. We put the training process in the cloud, and we use pre-existing local models to make predictions. In order to achieve high performance and low power consumption as much as possible, we carried out early exit, pruning and adaptive attention methods. We also proposed to store unanswerable questions in new datasets and fine-tune the model in the cloud for better results. Finally, we put this model into the NVIDIA Jetson TX2 development board for testing, and got satisfying results in performance and power consumption.

Key words: Natural Language Processing, Computation Offloading, Embedded System, Early-Exit

Contents

Chapter 1. Introduction

1.1 Background

Natural Language Processing (NLP) is one of the aspects of artificial intelligence, which focuses on the theories and methods of effective communication between human and computer in the field of natural language. This idea came from Machine Translation, which has a pedigree going back to the 1950s. Chronologically NLP can be divided into three period, symbolic NLP, statistical NLP, and neural NLP. Nowadays the machine learning approach with deep neural network has become widespread and enables NLP to have a wide use in our daily life, including question answering, text classification, auto correct, speech recognition, etc.

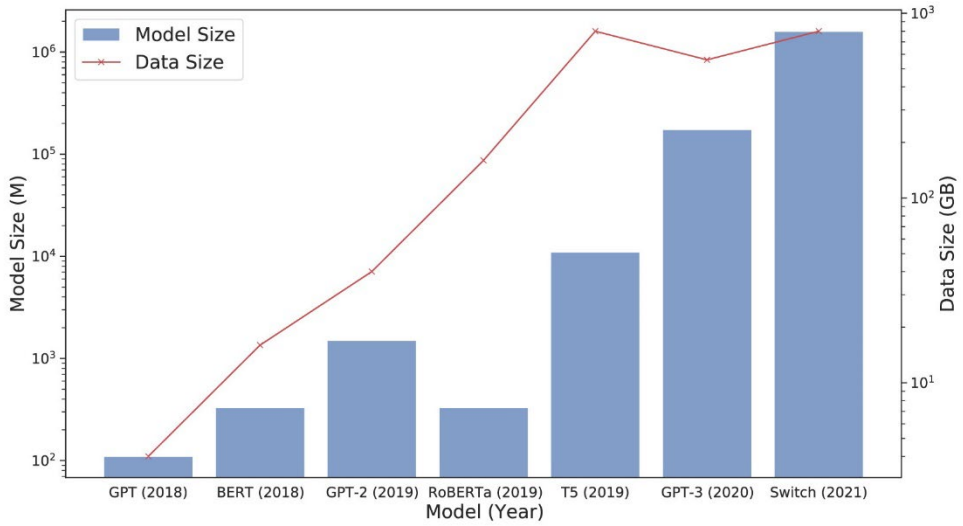

ILLUSTRATION 1.1 The model size and data size in recent NLP models ([1], Han et al., 2021: 4)

In recent years, more and more excellent models have emerged and their performance on NLP tasks has improved as well year by year. The advent of the model BERT is a

milestone development for NLP. It improved the evaluation on language understanding benchmark GLUE to 82.1%, which is a little bit lower than human performance with 87.1% accuracy ([1], Han et al., 2021: 2). In the following years, along with the appearance of some other excellent models, Roberta, T5, DEBERTA, etc., the performance of artificial intelligence even does a better job than human. Obviously, the model size grows along with the performance as well. When the size of pre-trained models becomes larger, large-scale pre-trained models with hundreds of millions of parameters can capture polysemous disambiguation, lexical and syntactic structures, as well as factual knowledge from the text ([1], Han et al., 2021: 4). For a 12 layers (transformer blocks) model, BERT Base, the number of parameters is up to 110 million while 340 million for BERT Large ([2], Devlin et al., 2019: 4179). As shown in ILLUSTRATION 1.1, to achieve a better performance, both the model size and the data size are increasing rapidly and even some models have more than billions of parameters.

1.2 Market Research

Nowadays embedded systems are more and more prevailing in a variety of fields in our life across sectors such as home and office, BFSI, security, automobile, defense, healthcare and other sectors ([3], Allied Market Research, 2019). From the cell phones around us, to the instruments used in military, embedded devices have already been ubiquitous and omnipotent. The number of its development and applications in real world is still growing at an astonishing rate.

Through pre-set program and plan, embedded computing systems are able to be used to control, monitor or perform a specific function of an electronic equipment. The growing market in technologically advanced consumer electronics, especially in the field of digitization of healthcare and industrial automation, drives the market of emb-

edded computing systems rapidly. Furthermore, the demand for embedded computing worldwide is increasing due to the potential growth in emerging economies and the future applications and evolutions in Internet of Things (IoT) ([3], Allied Market Research, 2019). With the dependency to the artificial intelligence in our daily life, the demand on the computing power of embedded systems becomes higher and higher. The advent of 5G technology is also expected to act as a growth opportunity for the embedded system market ([4], Markets and Markets, 2020). The global market of embedded systems is expected to grow from USD86898.31 million in 2019 to USD 118902.70 million by 2025 at a Compound Annual Growth Rate (CAGR) of 5.36% ([3], Allied Market Research, 2019).

1.3 Motivation

NLP still has a very promising development and application prospects. Among all the tasks of NLP, we choose the question answering (QA) considering the automated chat bots still have a lot of room from improvement and application in the huge market of embedded systems. However due to the limited storage memory in embedded systems, those quite large models are not appropriate to apply in users' terminals. There is a need for a methodology so that we can apply an excellent model to an embedded system with limited storage space. Nowadays the most common method is to keep the whole computing process in the cloud, which indicates that the embedded system only needs to transmit and receive data through the network so that it can share the strong computation and storage of the cloud.

However, with an increasing number of smart devices, the data size also has an explosive growth and a tremendous scale of data is uploaded to cloud to perform analytics and predictions. Under this situation, there are some problems with such a terminal-to-cloud connection. Since the bandwidth is fixed, the network will be bloc-

ked if too many users attempt to connect at the same time. Thus, the computation performance will be largely influenced. There may exist large latency and energy consumption during the data processing. Meanwhile, the data process in the cloud has the risk of information tampering and disclosure, largely threatening the security of user individual information. Modern terminal devices have a certain amount of computing power and storage space so that we can offload part the process to solve the problems.

1.4 Related Work

In the field of Natural Language Processing (NLP) and computation offloading for embedded systems, there are several high-performance optimized neural network models and offloading methods from different perspectives.

1.4.1 BERT

The Bidirectional Encoder Representations from Transformers (BERT) ([2], Devlin et al., 2019: 4171) divides the NLP model into two steps: pre-training and fine-tuning. The transformer encoder is with Gaussian Error Linear Units (GELU) nonlinearities. GELU is a useful activation function for neural networks achieving high performance ([5], Hendrycks, Gimpel, 2020: 1-2]).

$$\mathrm{GELU}(x) = x \cdot \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \approx 0.5x\left(1 + \tanh\left[\sqrt{\frac{2}{\pi}}\left(x + 0.44715x^3\right)\right]\right) \quad (1\text{-}1)$$

BERT uses a Masked Language Model (MLM) for language model pre-training and a Next Sentence Prediction (NSP) task for text-pair representation pre-training. For fine-tuning, taking question answering applications as an example, BERT takes ques-

tion-passage pairs as input and feeds token and representations to output layers.

1.4.2 ALBERT

ALBERT is a Lite BERT ([6], Lan et al., 2020: 1). It does three significant improvements on traditional BERT model: factorized embedded parameterization, cross-layer parameter sharing, and inter-sentence coherence loss (sentence-order prediction (SOP) loss). ALBERT achieves better performance than BERT given larger configurations and fewer parameters. There are different methods to do cross-layer parameter sharing. ALBERT shares all parameters across layers by default, but parameters to share can be customized, such as feed-forward network (FFN) only or attention parameters only.

1.4.3 EdgeBERT

EdgeBERT is a hardware/software (more specifically, hardware/algorithm) co-design for multi-task NLP whose purpose is to reduce energy consumption as much as possible while achieving improvements of accuracy and speed ([7], Tambe et al., 2021: 1). EdgeBERT proposes several main improvements, including entropy-based early exit and dynamic voltage-frequency scaling (DVFS). Our work is inspired by the strategies that EdgeBERT adopts to save energy, which is very important for embedded systems.

1.4.4 HuggingFace's Transformers

HuggingFace's Transformers is an open-source library, which consists of many well-

organized transformer architectures with a unified API ([8], Wolf et al., 2020: 38). It helps both researchers' and practitioners' access to this field more easily. The models in the library are composed of three blocks, including a tokenizer, a transformer, and a head, which can satisfy most needs from user. To facilitate easy use and distribution of pre-trained models, a community model hub is built to users, which has a number of models.

Chapter 2. Design Specification

2.1 Customer Requirements

Customer's requirements play a pivotal role in our project since it is application oriented. Considering the needs from customers, developers are expected to help solve current existing problems. Several aspects have been taken into account, including functionality, efficiency, security and universality.

1. **Functionality:** It is required to realize the basic function to implement question answering predictions on embedded systems. The result calculated locally should have a similar or better accuracy compared with the results on the cloud.
2. **Efficiency:** Since the data used to prediction needs to be transferred between terminal and the cloud, there exists a process of network communication. Thus, it's inevitable there will be network latency generated. For a better experience, a faster prediction is necessary.
3. **Security:** The safety of users' personal sensitive data should be placed at the first place. When these data are uploaded and analyzed on the cloud, there are risks and dangers of data leakage and data loss. Thus, information security should be taken into consideration.
4. **Universality:** Since the realization of prediction before needs a network connection, there are limitations to the user's use. A complementation without network requirements is needed.

General Requirements:

1. Can run question answering task (weight: 10)
2. Is suitable for embedded systems (weight: 10)

3. Have a good documentation for reference (weight: 6)

Performance Requirements:

1. Run fast for prediction (weight:9)
2. Run precisely for prediction (weight: 9)
3. Run for prediction with small power (weight: 8)
4. Lower network latency (weight: 4)
5. Protect personal sensitive data (weight: 8)
6. Can work without a network connection (weight: 6)
7. Occupy small storage space (weight: 8)
8. Support update in terms of user need (weight: 6)
9. Have a good transportability (weight: 7)

2.2 Engineering Specifications

The engineering requirements specifically provides guidelines for developers to design and construct the implementation of products. It outlines how technical problems are considered before we actually start our structural design.

1. **Functionality:** Considering the limitations of embedded systems, we can evaluate the functionality from two aspects, time and power. For function realization, a complementation with less time and less power will be preferred since it can improve the user experience and save the resources.
2. **Efficiency:** Specifically speaking, efficiency is a quite important perspective that determines the availability and competitiveness of our products. We are expected to improve the performance of prediction as much as possible.
3. **Security:** For prediction, to protect user information, the implementation of data encryption will consume quite a lot of resources. If we transfer the prediction to

locality, there is no need to waste extra resources to deal with encryption and decryption. And in this way, the personal information will still be local rather than accessed by the cloud so that the security can be guaranteed.

4. **Cost:** Local storage space and computation power on embedded systems are limited and expensive. In comparison, the cost is lower, and the power is greater on the cloud as a whole. Thus, for the training process, which is a procedure requires many resources, is better to work remotely.

Table 2.1 Engineering Specifications

Engineering Specification	Unit	Target Value
Train dataset on the cloud	-	Yes
To predict on the embedded system	-	Yes
Prediction time	s	< 1.0
Dataset size on the cloud	MiB	100
Training time	min/epoch	< 70
Support of look-up tables (LUTs)	-	Yes
Number of model parameters	MiB	< 15
Size of model	MiB	< 100
Power consumption	W	< 10
Protection of user information	-	Yes
Network latency	second	< 0.1
Frequency to update local model	hour	24
Support of subsequent model adjustments	-	Yes
Support of predicting offline	-	Yes
User guide and programmers' manual	-	Yes

In Table 2.1, the engineering specification is expanded to 15 detailed quality characteristics, which defines and constrains the concrete issues.

2.3 Quality Function Deployment

A House of Quality (HOQ) chart in the Quality Function Deployment (QFD) method

is shown in ILLUSTRATION 2.1. The detailed expansion has been showed in previous sections. In addition to the customer requirements and engineering specifications, this illustration also displays the relationship between them and the difficulty level we evaluate for each task. Then we compare our own project to others and get a better sense of orientation and objectives.

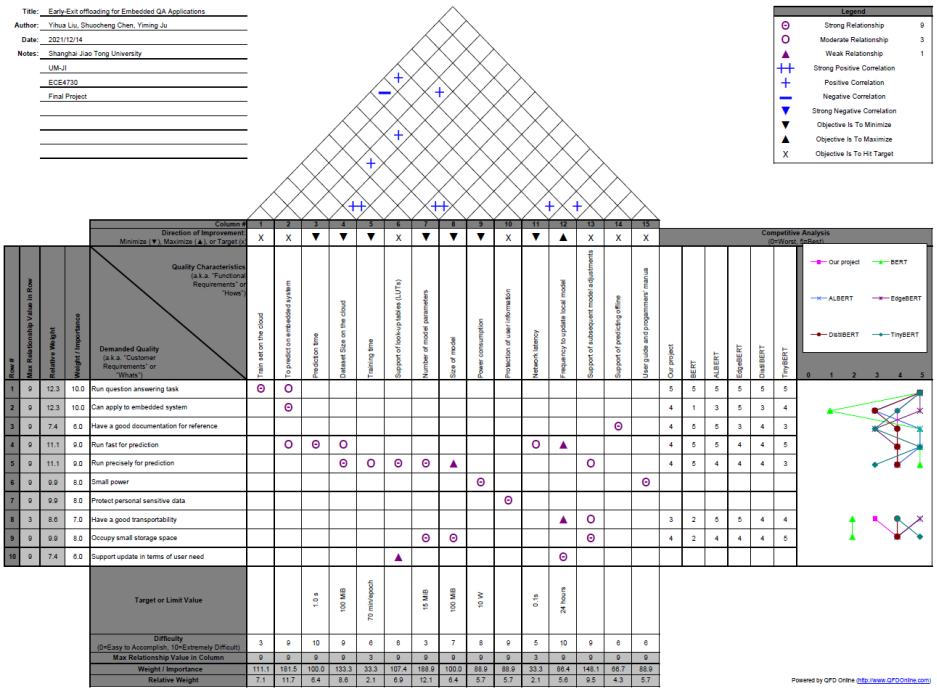

ILLUSTRATION 2.1 House of Quality (HOQ) chart

Chapter 3. Concept Generation and Selection

3.1 Concept Generation

3.1.1 Cloud Computing

Cloud computing is a service provided by data centers, which migrate the computing to the cloud due to the large data storage and powerful computing. Compared to a local terminal, external processing units on the cloud provides greater computing performance and it's more cost-effective than a single device since the cloud environment is maintained and developed by specialized staff. The users are allowed to benefit from the technologies offered by the cloud, without have a deep knowledge about how to realize. It effectively saves the learning cost and economic cost. Regardless of the location and time, users are able to access this service just by a web browser. Meanwhile, since user involvement are minimized through being excluded from the implementation process, the probability to cause errors is also minimized.
However, the whole computing process needs a wide and complete system, the data is distributed in a variety of links. The security of personal sensitive information may be threatened by external attack or internal break.

3.1.2 Computation Offloading

To cope with the cons of cloud computation, the concept of computation offloading is proposed. Computation offloading is a transfer of resource from the remote server to local terminal. It offloads part of cloud computing to the terminal device. In this way,

it not only helps solve the problem of cloud congestion but also accelerates computing and lower the latency through reducing data exchange with the cloud. Meanwhile, since the critical data can only be processed and stored locally, the security of individual private information can be guaranteed.

3.1.3 Transfer Learning

Transfer learning is a research problem focusing on storing knowledge and applying it to a different but related task, which is to say, it aims to adapt an existing model to be endow with a different but related function that solves a different problem. There are two generally adopted approaches for transfer learning, feature extraction and fine-tuning, between which the former one extracts the predicted features before the final classification from a pre-trained model and applies them to a new model, and the latter one retains the pre-trained model by further training the selected layers.

Transfer learning separates the process of training from scratch (pre-training) and further adaptation and optimization, which enables computational redistribution to balance its resource consumption. Novel research seeks to utilize this feature to embedded devices, particularly after COVID-19 with the pattern recognition requirement for health monitoring devices, from which new inspirations of further application of transfer learning on embedded devices emerge.

3.1.4 Feature Extraction

Pre-trained deep learning models generate features corresponding to the input before the final output layer (usually a classifier or a decoder). The method of feature extraction seeks to utilize these features from pre-trained model as inputs to generate

task-specific output, usually by training a customized classifier on a small, specified dataset.

3.1.5 Fine-tuning

Pre-trained models already store abundant knowledge in certain fields for they are trained on large datasets containing various information. Fine-tuning is to train such pre-trained models further on a rather small dataset specified for certain tasks which can be achieved with relatively high performance without extensive consumption of computational resources. Fine-tuning is usually done by freezing selected layers of a pre-trained model while continuing to train the others to avoid overfitting on the new dataset. However, such operation is not necessary in our case since all parameters are shared among ALBERT layers. Fine-tuning is usually expected to perform better than feature extraction towards abundant tasks.

3.2 Concept Selection

3.2.1 Dataset Selection: SQuAD 2.0

SQuAD (Stanford Question Answering Dataset) 2.0 is based on SQuAD ([9], Rajpurkar et al., 2018: 784), which is a reading comprehension dataset of high quality of a large scale.

1. Diversity of answers

   There are various kinds of answer types, including not only dates, values, person, location but also noun phrase, adjective and verbs. It helps the integrity of the training model.

2. Scale

   The total examples are about two times of SQuAD 1.1. In the training dataset, the number of answerable questions is about two times compared to unanswerable questions. And in the dev dataset, the proportion is about 1:1, which is quite challenging.

3. Unsolvable questions

   Extractive reading comprehension systems can often locate the correct answer to a question in a context document, but they also tend to make unreliable guesses on questions for which the correct answer is not stated in the context. Existing datasets either focus exclusively on answerable questions or use automatically generated unanswerable questions that are easy to identify. To address these weaknesses, the authors present SQuAD 2.0, the latest version of SQuAD. SQuAD 2.0 combines existing SQuAD data with over 50,000 unanswerable questions written defiantly by crowd workers to look similar to answerable ones.

3.2.2 Development Kit Selection: NVIDIA Jetson TX2

There is a wide use of NVIDIA Jetson TX2 in actual industries, including manufacturing, life sciences, manufacturing, retail, etc. It is designed specifically for embedded systems. It provides 4 different power modes for working. It has a dual-core NVIDIA Denver 2 64-bit CPU and a quad-core ARM Cortex-A57 MPCore. Compared with Intel x86 instruction set architecture (ISA), ARM is a reduced instruction set computer (RISC) architecture, which is simpler and more efficient. It operates at only between 7.5 W and 15 W of power. Its computational resources are also sufficient – it has a 256-core NVIDIA Pascal GPU ([10], NVIDIA Corporation, 2021). Given its exceptional efficiency for power and speed, it is allowed to efficiently run deep neural networks on this develop kit with a high accuracy, like ind-

ustrial robots, medical equipment, machine vision cameras, and question answering applications for our case.

3.2.3 Model Selection

Table 3.1 Decision Matrix for Model Selection

Criterion	Weight	BERT				ALBERT			
		TensorFlow		PyTorch		TensorFlow		PyTorch	
		S	R	S	R	S	R	S	R
Performance	0.35	9	3.15	9	3.15	8	2.8	8	2.8
Space	0.35	4	1.4	4	1.4	7	2.45	7	2.45
Flexibility	0.2	3	0.6	8	1.6	3	0.6	8	1.6
Extensibility	0.1	7	0.7	5	0.5	7	0.7	5	0.5
Total			5.85		6.65		6.55		7.35

Here, “S” stands for score and “R” stands for rating.

Considering the limited storage space and computing power of embedded systems, the model size should be taken into the first place. We choose two most prevalent models: BERT and ALBERT. BERT has a better performance on complex problems while ALBERT also has a good performance but with smaller parameter numbers. Thus, we take four aspects into consideration to determine our pre-trained model, including performance, space occupation, flexibility and extension and then evaluate the degree for each item. From the decision metric, the combination of ALBERT and PyTorch receives a relatively outstanding evaluation. There are three improvements and advantages of ALBERT over BERT ([6], Lan et al., 2020: 3-5).

1. Cross-layer parameter sharing

   In BERT, as the number of layers increases, the number of parameters increases exponentially. To solve this problem, ALBERT proposes the concept of cross-layer parameter sharing. That is the first transformer learning and other 11

transformers reusing the parameters in the first one. There are a large number of parameters used in BERT, up to 110 million in BERT-base. In comparison, there are only 31 million parameters in ALBERT. And it turned out not to have much effect on accuracy.

2. Sentence order prediction (SOP)

   The main idea of SOP is to take two consecutive paragraphs from the same document as a positive sample and then swap the order of the two paragraphs and use it as a negative sample. This allows the model to learn finer grained differences about consistency at the paragraph level, which improves the performance.

3. Embedding factorization

   Embedding factorization is to split the embedding matrix into embeddings in input level, which have a relatively low dimension. In this way, an 80% reduction will be achieved in the parameters but with a little influence on performance.

Besides, PyTorch is more suitable for our design because it generally provides better layer abstraction than TensorFlow. The “torch.nn” module provides abstraction like “Module” and “ModuleList”. The model can be hierarchical constructed conveniently by PyTorch, providing ALBERT layer class and ALBERT layer group class.

Chapter 4. Final Design and Implementation Plan

4.1 Final Design

The overall architecture of our application is demonstrated in ILLUSTRATION 4.1, which is comprised of the remote server deployed with the full model and the local embedded system deployed with the lite model.

After an input is provided by the user, the lite model would first give a lite output, either no answer or an answer, as a response to the user. The input would also be uploaded to the remote server simultaneously, which would go through the full model to generate a full output.

In the cases where the two outputs don't match or an answer cannot be provided by the lite model, they would be stored in the error dataset, which would comprise a dataset with corresponding correct answer used to further fine-tune the full model. After one of these further fine-tuning is conducted, the model would be synchronized to the local embedded system.

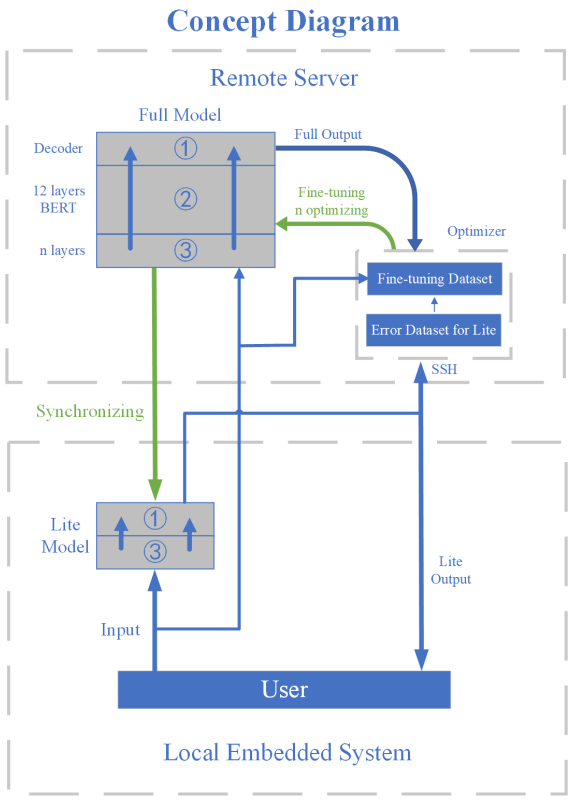

ILLUSTRATION 4.1 Design Diagram

4.1.1 Remote Server

The remote server is based on the VMware Bitfusion Virtual Desktop provided by the Joint Institute with the high computational capability of 8 Tesla-V100 32GB GPUs, on which the full model is deployed, fine-tuned, and further optimized. For training or evaluating teacher models, we use 2 GPU each time; for training or evaluating other models, we use 1 GPU each time.

For the error dataset and its co-derived fine-tuning dataset, we split the SQuAD 2.0 into two parts. The full model is fined-tuned on the larger part before deployment, and the smaller part is regarded as the extra fine-tuning dataset derived from the error dataset.

4.1.2 Local Embedded System

The local embedded system consists of an NVIDIA Jetson TX2 board, an extra 512 GB SATA 3.0 SSD, and other peripherals like USB-hub and HDMI screen. The extra storage is added because the embedded 32 GB eMMC 5.1 storage of NVIDIA Jetson TX2 is not sufficient for libraries. Early-exiting (EE) is employed on a full model, ensuring most inputs would only go through the first few layers before a final classification with an entropy assessment to ensure their accuracy.

4.1.3 Model Design

The basic layer-wise structure of the ALBERT is shown in ILLUSTRATION 4.2, the input goes through the embedded layer for word embedding at first. Then the embedded input goes through every single layer of transformers, between which an early-exiting chance is guaranteed and would be utilized to be the distinctive feature for lite model.

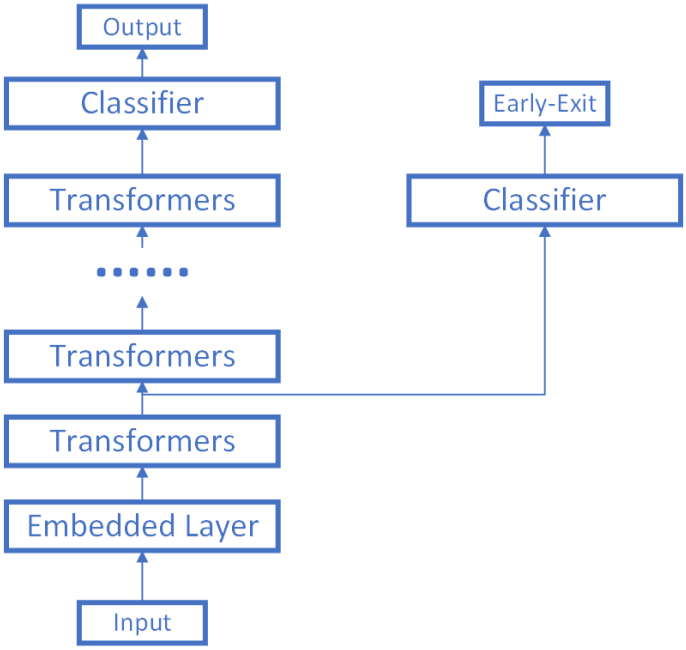

ILLUSTRATION 4.2 Layer-Wise Structure

4.2 Implementation Plan

Our implementation is based on HuggingFace's Transformers, an open-source Python package published by Hugging Face organization, which contains a self-contained implementation of the ALBERT model. Based on this architecture, we upgraded each module with our specified optimization methods, while retaining the logic of module skeletons to ensure its stability and applicability, for the dependencies of HuggingFace's Transformers are simple, easy to build, and have high accessibility in all platforms including the ARMv8/AArch64 architecture of the NVIDIA Jetson TX2 serving as the local embedded system.

4.3 Optimization

Various optimizations are experimented on our model to improve its performance on our embedded system with restricted computational resources. The designed optimizations are as follows.

4.3.1 Network Pruning

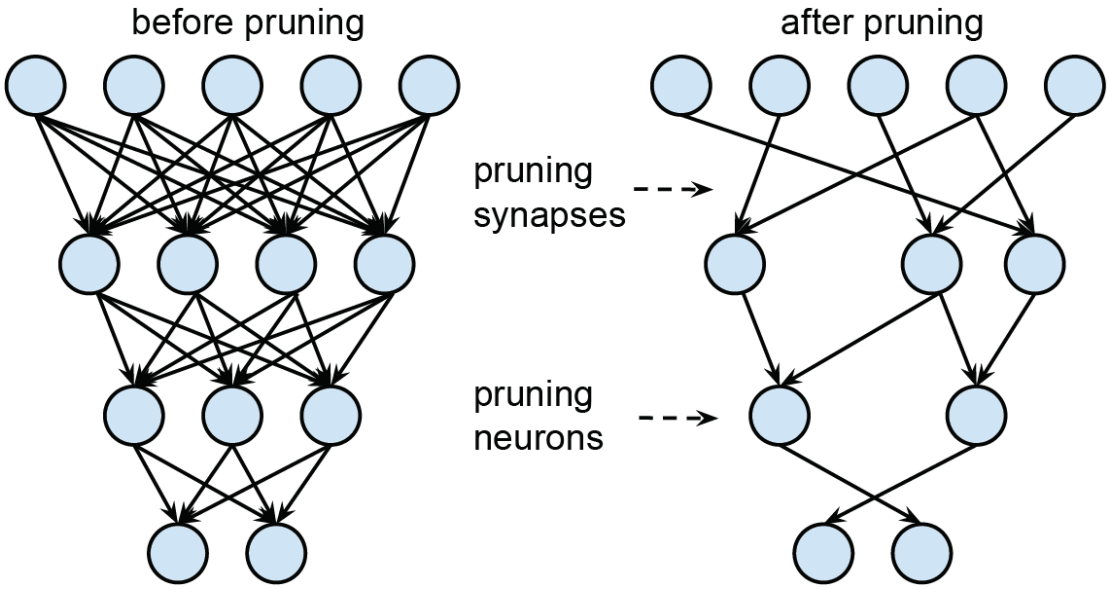

ILLUSTRATION 4.3 Static Network Pruning ([11], Han et al., 2015: 1138)

Network Pruning is a classic optimization method for neural networks, which is to delete certain neurons completely from the neural network according to certain standards. It reduces the total number of parameters in order to reduce the total computations required, which accelerates the processes of both training and evaluation with only minimal influences on accuracy. In our design, we focused on movement pruning, a method that prunes the neurons that have the most tendency to converge to zero during training. If the weight $W_{i,j}$ is greater than 0, it increases; otherwise, it decreases ([12], Sanh, Wolf, Rush, 2020: 4).

$$\frac{\partial \mathcal{L}}{\partial W_{i,j}} < 0 \text{ if } W_{i,j} > 0 \tag{4-1}$$

$$\frac{\partial \mathcal{L}}{\partial W_{i,j}} < 0 \text{ if } W_{i,j} < 0 \tag{4-2}$$

$\mathcal{L}$ stands for the loss. The score S can be calculated as

$$S = -\sum_{t} \left(\frac{\partial \mathcal{L}}{\partial W_{i,j}} \right)^{(t)} W_{i,j}^{(t)} \tag{4-3}$$

where t is the time step.

Experiments on magnitude pruning ([13], Han, Mao, Dally, 2016: 1), a static prune method that deletes the neurons with the smallest weights after training, are also conducted to inspect its effectiveness. Different from movement pruning, its score is simply

$$S = | W_{i,j} | \tag{4-4}$$

4.3.2 Adaptive Attention Span

The attention mechanism is widely adopted in transformers, which assigns weights, i.e., importance of words, to sentences. Adaptive attention span is a method introduced for transformers, which is to dynamically adjust the span of the masks of attention mechanism by training so that some masks can be turned off with minimal only influences on the accuracy. Given a soft-mask function

$$m_z(x) = \min\left(\max\left(\frac{1}{R}(R + z - x), 0 \right), 1 \right) \tag{4-5}$$

where x is the distance while z and R are parameters, the loss function can be modified as ([14], Suckbaatar et al., 2019: 331-333):

$$L = -\log\left(\prod_{t=1}^{T} P\left(w_t \mid w_{t-1},\ldots,w_1\right)\right) + \frac{\lambda}{M}\sum_i z_i \tag{4-6}$$

where M is head numbers of each layer, λ is parameter of regularization, and P is the probability of a token sequence.

This optimization leads to a decent deficit in the consumption of computational resources, for in each transformer of ALBERT there are 12 attention heads requiring abundant computation. It is applied to our model in every transformer layer.

4.3.3 AdamW Optimizer

AdamW optimizer is the Adaptive Moment Estimation Algorithm (Adam) optimizer ([15], Kingma, Ba, 2015: 1) with decoupled weight decay ([16], Loshchilov, Hutter, 2019: 1). Weight decay is a method of network regularization to restrict the complexity of the model ([17], Hanson, Pratt, 1988: 177). Adam is a combination of momentum method and RMSprop algorithm ([18], Tieleman, Hinton, 2012: 26):

$$M_t = \beta_1 M_{t-1} + \left(1-\beta_1\right)\boldsymbol{g}_t \tag{4-7}$$

$$G_t = \beta_2 G_{t-1} + \left(1-\beta_2\right)\boldsymbol{g}_t \odot \boldsymbol{g}_t \tag{4-8}$$

$$\hat{M}_t = \frac{M_t}{1-\beta_1^t} \tag{4-9}$$

$$\hat{G}_t = \frac{G_t}{1-\beta_2^t} \tag{4-10}$$

$$\theta_t = \theta_{t-1} - \eta_t\left(\frac{\alpha \hat{M}_t}{\sqrt{\hat{G}_t + \varepsilon}}\right) \tag{4-11}$$

Here, α is the learning rate, β_1 and β_2 are the weight decay rates of the two moving average, $\boldsymbol{g}_t$ is the t-th gradient of batch, M_t is the mean of the gradient

(the first moment), G_t is the variant of the gradient (the second moment), and θ_t is the parameter vector. $\hat{M}_t$ and $\hat{G}_t$ are modified values of M_t and G_t, and η_t is a coefficient for decaying the learning rate α. For AdamW, θ_t is instead updated by ([16], Loshchilov, Hutter, 2019: 2-3)

$$\theta_t = \theta_{t-1} - \eta_t \left(\frac{\alpha \hat{M}_t}{\sqrt{\hat{G}_t + \varepsilon}} + \lambda \theta_{t-1} \right) \tag{4-12}$$

where λ is the weight decay rate per step.

4.3.4 Learning Rate Warmup

To improve the stability of training, the learning rate could be dynamically adjusted. A relatively smaller learning rate could be adopted for first several iterations, and the initial learning rate is used when the gradient has decreased sufficiently. The model we use requires a few epochs, so complex warmup scheduler is not necessary. A lambda learning rate (LR) warmup scheduler (torch.optim.lr_scheduler.LambdaLR) is adopted:

$$\alpha_{\text{new}} = \lambda \alpha_{\text{init}} \tag{4-13}$$

4.3.5 FP16 Optimization

Floating point numbers have a complicated representation as indicated by IEEE standards. Although more floating-point digits mean higher accuracy, it would also lead to exponential increases in the number of computations that need to be done in simple arithmetic. Thus, a tradeoff between accuracy and speed needs to be evaluated. It's widely acknowledged that reduction of digits from 32 to 16 would lead to almost

no sacrifice in accuracy while significantly increasing the speed of training and evaluation. It could be applied by using a PyTorch extension called NVIDIA Apex.

4.3.6 Knowledge Distillation

Knowledge distillation divides the work into two parts: a teacher network model and a student network model ([19], Hinton, Vinyals, Dean, 2015: 1). The teacher network model storing more knowledge, i.e., trained on a larger dataset, can be used as a knowledge source to train the student network model on a smaller dataset or with fewer training epochs, improving its overall performance while saving the training time. Experiments of this method on our model are conducted. During training, the total loss is calculated by

$$\text{total\_loss}=(1-\alpha)\cdot\frac{\text{start\_loss}+\text{end\_loss}}{2}+\alpha\cdot\text{teacher\_loss} \tag{4-14}$$

Here, "teacher_loss" is the loss of the teacher model, α is the distillation rate, "start_loss" and "end_loss" are mean Kullback-Leibler divergence (also called relative entropy) losses of logits of the start positions and the end positions of an example:

$$\text{loss}=\frac{D_{\text{KL}}(P\parallel Q)}{\text{batch\_size}}=\frac{\sum_{i=1}^{N}l_i}{\text{batch\_size}} \tag{4-15}$$

where $D_{\text{KL}}(P\parallel Q)$ is the relative entropy between discrete probability distributions P and Q, N is the number of losses. Each batch's loss l_i is ([20], MacKay, 2003: 34)

$$l_i=P(i)\cdot\log\left(\frac{P(i)}{Q(i)}\right) \tag{4-16}$$

However, PyTorch ATen assumes that the input Q is already log probabilities in

"kl_div_non_log_target" function, so "torch.nn.functional.kl_div" or "torch.nn.KLDivLoss" function works as ([21], Torch Contributors, 2021):

$$l_i = P(i) \cdot \left(\log\left(P(i)\right) - Q(i) \right) \tag{4-17}$$

where $P(i)$ is the target true distribution and $Q(i)$ is the input predicted distribution. Thus, $P(i)$ and $Q(i)$ are given by

$$P(i) = \text{log\_softmax}\left(\text{student\_start\_logit}\right) \tag{4-18}$$

$$Q(i) = \text{softmax}\left(\text{teacher\_start\_logit}\right) \tag{4-19}$$

Logits are raw values. Softmax and Log_softmax functions are defined as

$$\text{Softmax}\left(x_i\right) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{4-20}$$

$$\text{Log\_softmax}\left(x_i\right) = \log\left(\text{Softmax}\left(x_i\right)\right) = \log\left(\frac{e^{x_i}}{\sum_j e^{x_j}}\right) \tag{4-21}$$

4.4 Implementation Architecture

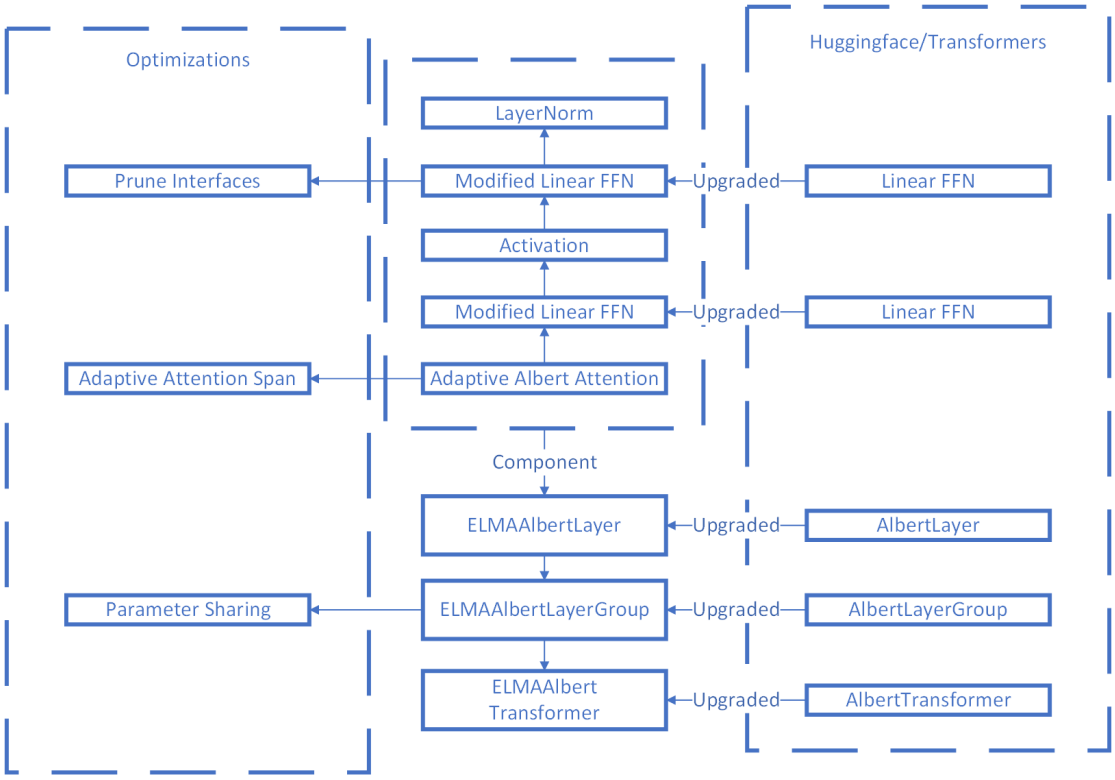

ILLUSTRATION 4.4 Implementation Architecture

ILLUSTRATION 4.4 shows the bottom implementation of our optimized ALBERT model. The basic ALBERT layer from HuggingFace's Transformers contains an attention layer, two linear feed-forward networks (FFN), an activation function layer, and a normalization layer. In our ELMAAlbertLayer, FFNs are modified to add prune interfaces, enabling the optimization method of movement prune. Attention layers are modified to include the optimization method of adaptive attention span. Like it is in HuggingFace's Transformers, an ELMAAlbertLayerGroup is then comprised of twelve ELMAAlbertLayers, which is the key to parameter sharing. Then it comprises a larger module ELMAAlbertTransformer that contains more function about Early-Exiting.

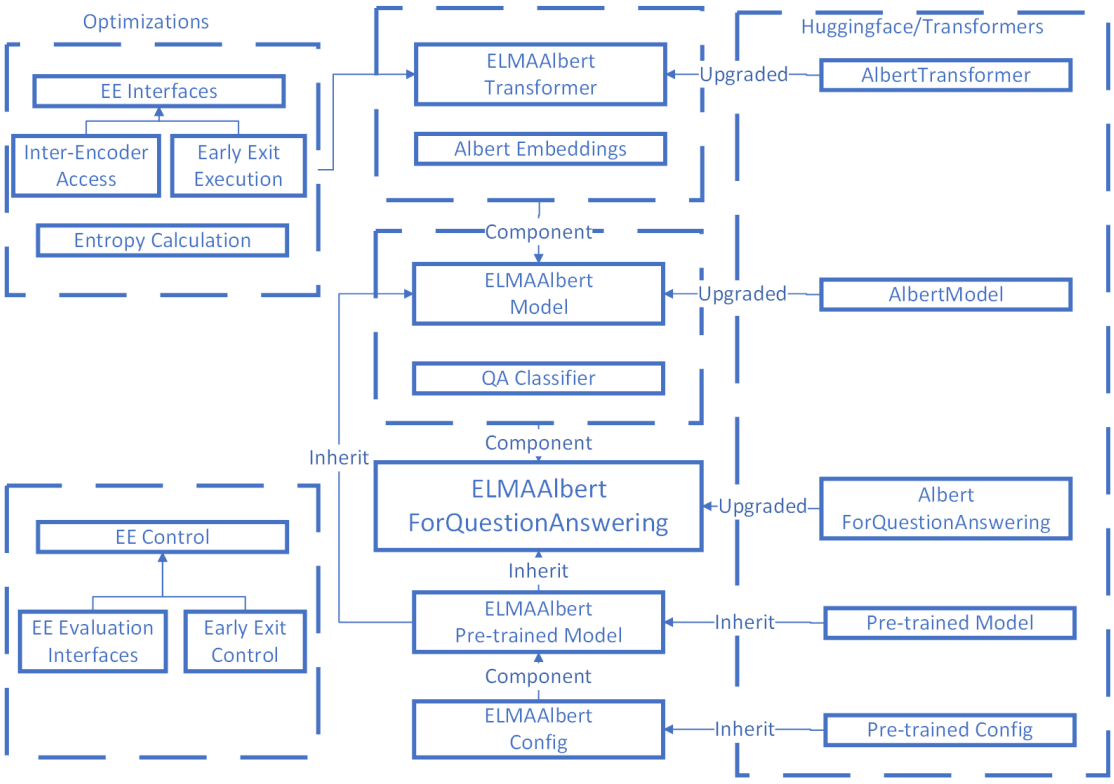

ILLUSTRATION 4.5 Top Implementation of the Model

ILLUSTRATION 4.5 shows the top implementation of the model. ELMAAlbertTransformer that serves as an interface hub between the bottom half and the top half, which is the reason why interfaces of EE are implemented here. As shown in ILLUSTRATION 4.2, an embedded layer is compiled together to form ELMAAlbertModel that inherits the base model ELMAAlbertPre-trainedModel. The latter model imports various configs, including the basic ALBERT config and the ELMA-specified config. A question-answering classifier is supplemented to form the final module ELMAAlbertForQuestionAnswering that we import directly in the main training-evaluating program. The final module offers direct control interfaces of EE which can be manipulated simply by-passing specified variables. For each level of the hierarchy, the early-exit exceptions are well-controlled.

In ELMAAlbertForQuestionAnswering class, the loss function is defined as

$$\text{loss}=\frac{\text{start\_loss}+\text{end\_loss}}{2} \tag{4-22}$$

where "start_loss" and "end_loss" are cross entropy losses commonly used for classification problems. Cross entropy loss is defined as

$$\mathcal{L}(p,q)=\frac{\sum_{i=1}^{N}\mathcal{L}_i(p,q)}{N} \tag{4-23}$$

$$\mathcal{L}_i(p,q)=-\sum_{c=1}^{C}w_c p(c)\log q(c) \tag{4-24}$$

where N is the minibatch dimension, w_i is the weight, c is the class index, $q(c)$ is the Softmax of the input (the predicted distribution), and $p(c)$ is the target (the true distribution).

In "Entropy Calculation", the entropy in terms of informatic theory is defined as

$$H=-\sum_{i=1}^{N}P(i)\log\left(P(i)\right) \tag{4-25}$$

where N is the number of logits, $P(i)$ is the Softmax of the logits x_i:

$$P(i)=\text{Softmax}(x_i)=\frac{e^{x_i}}{\sum_{j=1}^{N}e^{x_j}} \tag{4-26}$$

Substitute (4-26) into (4-25),

$$\begin{aligned}H&=-\sum_{i=1}^{N}\left(\frac{e^{x_i}}{\sum_{j=1}^{N}e^{x_j}}\log\left(\frac{e^{x_i}}{\sum_{j=1}^{N}e^{x_j}}\right)\right)\\&=\log\left(\sum_{i=1}^{N}e^{x_i}\right)-\frac{\sum_{i=1}^{N}x_i e^{x_i}}{\sum_{i=1}^{N}e^{x_i}}\end{aligned} \tag{4-27}$$

4.5 Project Timeline

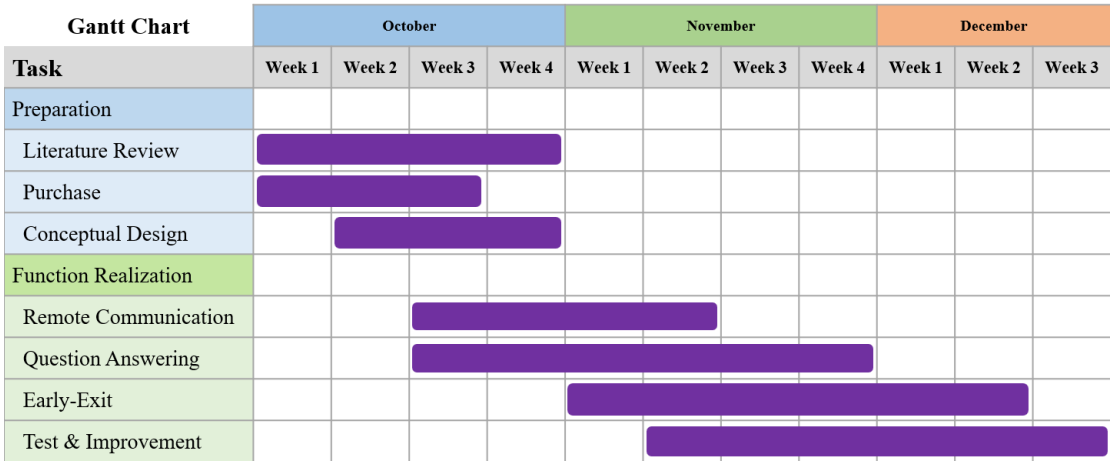

ILLUSTRATION 4.6 Gantt Chart of the Project

4.6 Development Environment

4.6.1 Local Embedded System

For the embedded system on NVIDIA Jetson TX2 board, the development environment is based on NVIDIA JetPack SDK 4.6. The specifications are:

1. Ubuntu 18.04.6 LTS (bionic) with Linux kernel 4.9.253-tegra
2. CUDA Toolkit 10.2 for L4T
3. cuDNN 8.2.1
4. TensorRT 8.0.1-1+cuda10.2 (8.0.1.6)
5. GCC (Ubuntu/Linaro 7.5.0-3ubuntu1~18.04) 7.5.0
6. Python 3.6.9
7. PyTorch 1.10.0 provided by NVIDIA official release specifically for Jetson

4.6.2 Remote Server

For the remote VMware Bitfusion server, the specifications are:

1. Ubuntu 20.04.3 LTS (focal) with Linux kernel 5.11.0-38-generic
2. CUDA Toolkit 11.2
3. GCC (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
4. Python 3.7.12
5. PyTorch 1.10.0

Chapter 5. Validation Results

5.1 Model Performance

During training and evaluation, the maximal sequence length is set to 128, and ALBERT Base version 2 pre-trained model is adopted. Besides, the learning rate is set to 3×10^{-5}. On the server, the training time for first half epochs are approximately 36 minutes per epoch, and the training time for the second half epochs are approximately 35 minutes per epoch.

The model performance is evaluated by two main metrics provided by SQuAD: exact score and f1 score. Exact score is defined by

$$\text{exact}=\max\left(\text{exact}_i\right) \tag{5-1}$$

where exact_i is defined by

$$\text{exact}_i=\left(\text{norm}\left(\text{orig\_answer}_i\right)==\text{norm}\left(\text{pred\_answer}\right)\right) \tag{5-2}$$

Here, $\text{norm}(\ )$ function is used to normalize text. Normalization is to remove some punctuations, accents of Latin characters, and transform uppercase characters to lowercase characters if the corresponding argument is passed. ALBERT model is an "uncased model", so the argument is set to true. "pred_answer" is the predicted answer to a question, while "orig_answer" is the original text answer to the same question with the corresponding question ID (QID). In the SQuAD 2.0 dataset, each question may have multiple answer texts, so we need to compare each of the answer text with the predicted answer and take the maximum value. The exact score of each question has only two possible values: 0 and 1. If the normalized predicted answer is exactly the same as the normalized original answer text, the i-th exact score of the question is 1, otherwise, it is 0. Thus, as long as one of the original answer matches

the predicted answer, the exact score of the question is 1.

F1 score is defined by

$$f1=\max\left(f1_i\right) \tag{5-3}$$

where $f1_i$ is defined by

$$f1_i=\frac{2\times precision\times recall}{precision+recall} \tag{5-4}$$

where precision is defined by

$$precision=\frac{TP}{TP+FP} \tag{5-5}$$

and recall is defined by

$$recall=\frac{TP}{TP+FN} \tag{5-6}$$

TP means “true positive”, FP means “false positive”, and FN means “false negative”.

The evaluation results of the teacher model undergoing 3 * 2 training epochs with batch size of 32 are:

Table 5.1 Evaluation Results of the Teacher Model

Metrics	Values
HasAns_exact	65.72199730094466
HasAns_f1	71.02531166019085
HasAns_total	5928
NoAns_exact	84.44070647603027
NoAns_f1	84.44070647603027
NoAns_total	5945
best_exact	75.10317527162469
best_f1	77.75103575521024
exact	75.09475280047165
f1	77.74261328405724
total	11873

Since SQuAD 2.0 dataset also includes items that do not have answers, the metrics of

SQuAD 2.0 can separately evaluate two parts: items that have answers (HasAns) and items that do not have answers (NoAns). The metric "total" is the number of evaluated items. The two scores "best_exact" and "best_f1" are the maximum values of exact scores and f1 scores among the list of questions.

For each two epochs of training the student models, we train the model two times. Each time, the model has to warm up. The evaluation results of the model undergoing 5 * 2 epochs with batch size of 64 without pruning are

Table 5.2 Evaluation Results of the Model without Pruning (10 epochs)

Metrics	Values
HasAns_exact	58.5863697705803
HasAns_f1	64.27142340520555
HasAns_total	5928
NoAns_exact	80.52144659377629
NoAns_f1	80.52144659377629
NoAns_total	5945
best_exact	69.56961172407985
best_f1	72.40806855437212
exact	69.56961172407985
f1	72.4080685543721
total	11873

For the model undergoing 15 * 2 epochs with batch size of 64 without pruning, the evaluation results are

Table 5.3 Evaluation Results of the Model without Pruning (30 epochs)

Metrics	Values
HasAns_exact	57.52361673414305
HasAns_f1	64.53721945367413
HasAns_total	5928
NoAns_exact	79.2094196804037
NoAns_f1	79.2094196804037
NoAns_total	5945

Table 5.3

Metrics	Values
best_exact	68.38204329150173
best_f1	71.88382354260784
exact	68.38204329150173
f1	71.88382354260779
total	11873

Compared with the model undergoing 5 * 2 epochs with batch size of 64 without pruning, the performance is slightly dropped, because overfitting happens. The evaluation time is 3653 s (5.87 iterations per second).

The model undergoing 5 * 2 epochs with batch size of 64, prune percentile of 30% still has a good performance:

Table 5.4 Evaluation Results of the Model with Pruning (10 epochs)

Metrics	Values
HasAns_exact	56.62955465587044
HasAns_f1	62.889758756630144
HasAns_total	5928
NoAns_exact	81.42977291841883
NoAns_f1	81.42977291841883
NoAns_total	5945
best_exact	69.04741851259159
best_f1	72.17303881995325
exact	69.04741851259159
f1	72.1730388199533
total	11873

Comparing Table 5.3 with Table 5.4, HasAns scores are reduced while NoAns scores are increased. Overall, the properly pruned model may have the same performance as the unpruned model.

The distilled student model undergoing 5 * 2 epochs with batch size of 64 and distillation rate of 10% without pruning:

Table 5.5 Evaluation Results of the Distilled Model without Pruning (10 epochs)

Metrics	Values
HasAns_exact	20.091093117408906
HasAns_f1	21.46024993152106
HasAns_total	5928
NoAns_exact	97.79646761984861
NoAns_f1	97.79646761984861
NoAns_total	5945
best_exact	58.999410427019285
best_f1	59.68300864095496
exact	58.999410427019285
f1	59.68300864095489
total	11873

The distilled model has lower HasAns scores but higher NoAns scores. The overall performance is not reduced significantly.

Then, we perform entropy-based early-exit. By grid search, we choose the early exit entropy as [0.1, 0.15]. The HasAns scores are not reduced significantly.

Table 5.6 Evaluation Results of the Early-Exit Model without Pruning (10 epochs)

Metrics	Values
HasAns_exact	47.31781376518219
HasAns_f1	57.29484640037728
HasAns_total	5928
NoAns_exact	0.5382674516400336
NoAns_f1	0.5382674516400336
NoAns_total	5945
best_exact	50.07159100480081
best_f1	50.07418253438635
exact	23.894550661163986
f1	28.875924320848696
total	11873

The exit layer counter is

Table 5.7 Exit Layer Counter of Model without Pruning (10 epochs)

Layer #	Count
1	4298
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	17156

The counter indicates that 17156 items are not early exited, while 4298 items are early exited at layer 1. The average exit layer is 9.796308380721543. Expected computation saving is 0.8163590317256954. The evaluation time is 2616.126591 seconds (8.20 iterations per second, 0.121941 seconds per example), which is much less.

If network pruning is applied, the performance will be better. By grid search, we choose the early exit entropy as [1.0, 1.0].

Table 5.8 Evaluation Results of the Early-Exit Model with Pruning (epoch 10)

Metrics	Values
HasAns_exact	46.002024291497975
HasAns_f1	53.867946342340446
HasAns_total	5928
NoAns_exact	70.39529015979815
NoAns_f1	70.39529015979815
NoAns_total	5945
best_exact	58.224543080939945
best_f1	62.143851412630475
exact	58.21612060978691

Table 5.8

Metrics	Values
f1	62.14345034257536
total	11873

and the exit layer counter is

Table 5.9 Exit Layer Counter of Model without Pruning (10 epochs)

Layer #	Count
1	4944
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	16388

The evaluation time is 21 min 19 s (16.77 iterations per second), which is much less.

5.2 Power and Energy Consumption

We measured the power the energy consumption for evaluating on NVIDIA Jetson TX2 by KeySight E36311A Triple Output Programmable DC Power Supply 6 V, 5 A / ±25 V, 1 A. We evaluate the early-exit version and non-early-exit version of the model without pruning trained undergoing 5 epochs. The energy consumption is

Table 5.10 Energy Consumption

Early Exited	Energy Consumption (J)
Yes	23914.33
No	28471.62

We can observe that the early-exited model has less energy consumption.

The power consumption figure is

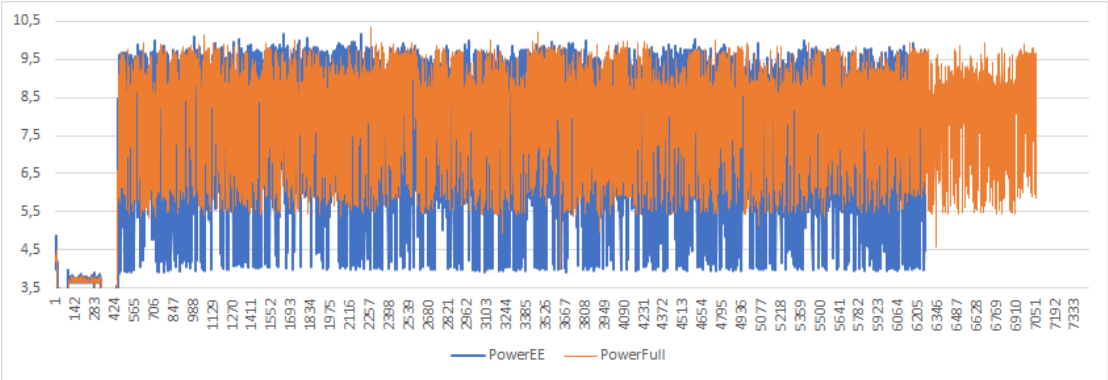

ILLUSTRATION 5.1 Power Consumption Figure Regarding Early-Exiting

The “PowerEE” line is the power consumption of the model with early-exit while the “PowerFull” line is the power consumption of the model without early-exit. We can observe that the early-exited model has less evaluation time and less power consumption.

Chapter 6. Discussion and Conclusions

6.1 Discussions

6.1.1 Prune Percentile

When training models, network pruning is adopted. However, the prune percentile should not be too high, otherwise there might be a severe performance drop. If the prune percentile of 60% is used instead of 30% and the distillation rate is still 10%, the evaluation results is

Table 6.1 Evaluation Results of the Distilled Model with Movement Pruning (percentile = 60%, distillation rate = 10%)

Metrics	Values
HasAns_exact	0.0
HasAns_f1	6.673887085102844
HasAns_total	96
NoAns_exact	0.0
NoAns_f1	0.0
NoAns_total	112
best_exact	53.84615384615385
best_f1	54.11786130536131
exact	0.0
f1	3.080255577739774
total	208

By examining the program, the 0 scores are not caused by program logic. The program gives nonzero values of start positions and end positions, but they are not proper.

If we change pruning method from movement pruning to magnitude pruning, the performance is still unacceptable.

Table 6.2 Evaluation Results of the Distilled Model with Magnitude Pruning (percentile = 60%, distillation rate = 10%)

Metrics	Values
HasAns_exact	1.0416666666666667
HasAns_f1	6.988357494114073
HasAns_total	96
NoAns_exact	0.0
NoAns_f1	0.0
NoAns_total	112
best_exact	54.32692307692308
best_f1	54.43376068376069
exact	0.4807692307692308
f1	3.2253957665141875
total	208

If changing distillation rate from 10% to 50%, the performance is still unacceptable.

Table 6.3 Evaluation Results of the Distilled Model with Movement Pruning (percentile = 60%, distillation rate = 50%)

Metrics	Values
HasAns_exact	0.0
HasAns_f1	2.8692996817996814
HasAns_total	96
NoAns_exact	0.0
NoAns_f1	0.0
NoAns_total	112
best_exact	53.84615384615385
best_f1	53.89423076923077
exact	0.0
f1	1.3242921608306222
total	208

Therefore, the problem is caused by a too high prune percentile.

6.1.2 Early Exit Entropy

The early exit entropy should not be too high or too low. If the early exit entropy is too low, there are few layers exited or even no layer exited. However, if the early exit entropy is too high, most layers exited early, and the performance will be very poor. For example, for model undergoing 15 * 2 epochs without pruning, if the early exit entropy is set as [1.0, 1.0], the evaluation results are

Table 6.4 Evaluation Results of the Early-Exit Model without Pruning (30 epochs)

Metrics	Values
HasAns_exact	0.0
HasAns_f1	0.0
HasAns_total	5928
NoAns_exact	99.98317914213625
NoAns_f1	99.98317914213625
NoAns_total	5945
best_exact	50.07159100480081
best_f1	50.07159100480081
exact	50.06316853364777
f1	50.06316853364777
total	11873

and the exit layer counter is

Table 6.5 Exit Layer Counter of Model without Pruning (30 epochs)

Layer #	Count
1	21453
2	0
3	0

Table 6.5

Layer #	Count
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	1

6.2 Conclusions

In this project, we introduce the idea of offloading into the design of question answering model. Our model applies early exit, prune, and adaptive attention methods to achieve a better performance and lower power consumption. The specifications are satisfied:

1. Functional Realization. The functions of the application are well realized by our implementation and are examined by our validations.
2. Efficiency. The efficiency is improved by the properly working early-exit method.
3. Security. The security is ensured by SSH connections during computation offloading. The data for evaluations and predictions are stored locally.
4. Cost. The cost is reduced during computation offloading by moving the huge computation of training models to the cloud.

In conclusion, as the project is mainly for application purposes, it’s quite successful from the technical perspective.

6.3 Future Work

6.3.1 Synchronization

The implementation of synchronization now is to manually migrate the updated model from the server to the locality. However, if this idea is applied to physical industries, manual transfer is obviously not reasonable or appropriate. Thus, an automatically synchronization method should be put forward. One possible way is to build the connection between the cloud and the terminal with SSH, secure shell, which can provide users a secure way to access to the server.

6.3.2 Parameter Sharing Inside a Layer Group

As discussed in previous chapters, the ALBERT model has the advantage of parameter sharing but may has a little loss of accuracy in complex tasks. To reach a subtle balance between accuracy and parameters number, we can change the number of layers to share the parameters.

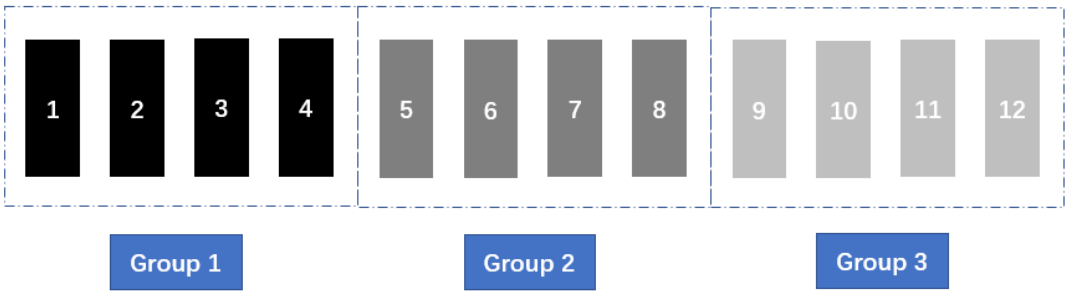

ILLUSTRATION 6.1 Parameter Sharing in Different Layer Groups

Assume there are 12 layers totally in a model. If we divide them into 3-layer groups,

with each group has four layers, the parameters will not be shared in the whole model. Instead, there are three sets of parameters shared in three groups. 1, 2, 3, 4 share the same parameters, for example, and is different from group 2 and group 3.

6.3.3 Grid Search

Grid search is a mean of adjusting parameters used to optimize the hyperparameters of a model. There is a parameter list and their possible values. Then according to the combination of each possible parameter, the model is trained with different parameters and evaluated. In terms of the evaluation results, the maximum performance can be found so that the optimal parameter combination is determined.

In our implementation, there are totally six parameters that apply to grid search, including epoch, learning rate, batch size, parameters for the distillation rate, and threshold.

6.3.4 Optimizer

In the processing of deep learning back propagation, the optimizer is to guide each parameter of the loss function to update the appropriate size in the right direction, so that the updated parameters make the loss function value approach the global minimum continuously. Now the default optimizer used in our implementation is AdamW, which is a variant of the optimizer Adam with an addition of weight decay lowering the chance of overfitting. To achieve a better performance, we may try some other optimizers. One direct approach is to apply Nesterov Accelerated Gradient (NAG) to Adam as Nadam ([22], Dozat, 2016: 1-3).

Nadam changes the modified first momentum $\hat{M}_t$ as

$$\hat{M}_t = \frac{\beta_{1,t+1} M_t}{1-\prod_{i=1}^{t+1}\beta_{1,i}} + \frac{\left(1-\beta_{1,t}\right)\boldsymbol{g}_t}{1-\prod_{i=1}^{t}\beta_{1,i}} \tag{6-1}$$

Besides, researchers are still arguing that the traditional Stochastic Gradient Descent (SGD) methods like Stochastic Gradient Descent with Warm Restarts (SGDR) ([23], Loshchilov, Hutter, 2017:1]) are more applicable to some cases than Adam optimizers. SGDR performs learning rate decay as

$$\alpha_t = \alpha_{\min}^i + \frac{1}{2}\left(\alpha_{\max}^i - \alpha_{\min}^i\right)\left(1+\cos\left(\frac{\pi T}{T_i}\right)\right) \tag{6-2}$$

where α is the learning rate, T is the epoch number, and t is the iteration number of batches.

6.3.5 Entropy Look-Up Table (LUT)

For current work, entropy is selected by grid search ideally. However, inspired by EdgeBERT, an entropy look-up table (LUT) could be generated for predicting the exit layer counter. The prediction is also a machine learning model, but it could be much simpler. The entropy raw dataset of start logits and end logits could be saved when training and evaluating. Based on the raw dataset, a sequential model with an activation function of Rectified Linear Unit (ReLU)

$$\mathrm{ReLU}\left(x\right) = \max\left(0, x\right) \tag{6-3}$$

and loss function of mean squared error

$$\mathcal{L} = \left(P-Q\right)^2 \tag{6-4}$$

could be constructed for generating an LUT (P is the true distribution and Q is the predicted distribution). The LUT could consist of original entropy or precited entropy for different layers, so that the predicted entropy could be fed to the early-exit calcula-

tion. This method is expected to save time and improve efficiency. The sample start and end entropy dataset entries derived from the model undergoing 5 * 2 epochs without pruning or distillation are

Table 6.6 Sample Start Entropy Dataset Entry of Model without Pruning (10 epochs)

Layer #	Entropy
1	1.4614043235778809
2	1.4510130882263184
3	1.4096579551696777
4	1.2934203147888184
5	1.3720407485961914
6	1.2034330368041992
7	0.2853541374206543
8	0.15657329559326172
9	0.3611793518066406
10	0.47565269470214844
11	0.33516573905944824
12	4.838024139404297

Table 6.7 Sample End Entropy Dataset Entry of Model without Pruning (10 epochs)

Layer #	Entropy
1	1.4569621086120605
2	1.4251103401184082
3	1.2901549339294434
4	1.3065252304077148
5	1.6546766757965088
6	1.200993537902832
7	0.1871790885925293
8	0.09745502471923828
9	0.12936687469482422
10	0.2665257453918457
11	0.16759300231933594
12	4.823031902313232

Let the number of predicted layers to be 4, the sample start and end look-up table entries of entropy predicting model based on the model undergoing 5 * 2 epochs without pruning or distillation are

Table 6.8 Sample Start and End LUT Entry of Entropy Predicting Model

Layer #	Start Entropy	End Entropy
8	1.282165572047233582e-02	1.335857063531875610e-02
9	3.322174027562141418e-02	1.545288786292076111e-02
10	4.768469184637069702e-02	9.329434484243392944e-03
11	4.646288603544235229e-02	8.547436445951461792e-03

The corresponding training losses and evaluation losses are

Table 6.9 Start and End Losses of Entropy Predicting Model

Epoch	Start Losses	End Losses
Train epoch 1	0.0851	0.0716
Train epoch 2	0.0706	0.0644
Train epoch 3	0.0704	0.0638
Evaluation	0.0728	0.0597

REFERENCE

[1] HAN X, ZHANG Z, DING N, et al. Pre-Trained Models: Past, present and Future[EB/OL].arXiv,2021[2021-10-5].https://arxiv.org/abs/2106.07139.

[2] DEVLIN J, CHANG M, LEE K, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[C].NAACL HLT 2019 – 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies – Proceedings of the Conference.2019,1:4171-4186.

[3] Allied Market Research. Embedded Computing Market by Type (Microprocessor, Microcontroller, Digital Signal Processor, ASIC, FPGA) and End User (Automotive, Industrial, Healthcare, Energy, Communications, Banking, Transport, Government, Robotics, Defense) - Global Opportunity Analysis and Industry Forecast, 2015 – 2022[R/OL].Valuates Reports,2019[2021-12-18].https://reports.valuates.com/market-reports/ALLI-Auto-2G51/embedded-computing.

[4] Markets and Markets. Embedded System Market by Hardware (MPU, MCU, Application-specific Integrated Circuits, DSP, FPGA, and Memories), Software (Middleware, Operating Systems), System Size, Functionality, Application, Region - Global Forecast to 2025[R/OL].Markets and Markets,2020[2021-12-18].https://www.marketsandmarkets.com/Market-Reports/embedded-system-market-98154672.html.

[5] HENDRYCKS D, GIMPEL K. Gaussian Error Linear Units (GELUs)[EB/OL].arXiv,2020[2021-12-18].https://arxiv.org/abs/1606.08415.

[6] LAN Z, CHEN M, GOODMAN S, et al. ALBERT A Lite BERT for Self-Supervised Learning of Language Representations[EB/OL].arXiv,2020[2021-12-28].https://arxiv.org/abs/1909.119

42.

[7] TAMBE T, HOOPER C, PENTECOST L, et al. EdgeBERT: Sentence-Level Energy Optimizations for Latency-Aware Multi-Task NLP Inference[EB/OL].arXiv,2021[2021-12-18].https://arxiv.org/abs/2011.14203.

[8] WOLF T, DEBUT L, SANH V, et al. HuggingFace's Transformers: State-of-the-art Natural Language Processing[C].EMNLP 2020 – 2020 Conference on Empirical Methods in Natural Language Processing.2020:38-45.

[9] RAJPURKAR P, JIA R, LIANG P. Know What You Don't Know: Unanswerable Questions for SQuAD[C].ACL 2018 – Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics.2018,2:784-789.

[10] NVIDIA Corporation. Jetson TX2 Module[EB/OL].NVIDIA Developer,2021[2021-12-28].https://developer.nvidia.com/embedded/jetson-tx2.

[11] HAN S, POOL J, TRAN J, et al. Learning both Weights and Connections for Efficient Neural Networks[C].NIPS 2015 – 29th Annual Conference on Neural Information Processing Systems.2015,1:1135-1143.

[12] SANH V, WOLF T, RUSH A M. Movement Pruning: Adaptive Sparsity by Fine-Tuning[EB/OL].arXiv,2020[2021-12-28].https://arxiv.org/abs/2005.07683.

[13] HAN S, MAO H, DALLY W J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding[EB/OL].arXiv,2016[2021-12-28].https://arxiv.org/abs/1510.00149.

[14] SUKHBAATAR S, GRAVE E, BOJANOWSKI P, et al. Adaptive Attention Span in Transformers[C].ACL 2019 – Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics.2019:331-335.

[15] KINGMA D P, BA J L. Adam: A Method for Stochastic Optimization[C].ICLR 2015 – 3rd International Conference on Learning Representations.2015.

[16] LOSHCHILOV I, HUTTER F. Decoupled Weight Decay Regularization[C].ICLR 2019 – 7th International Conference on Learning Representations.2019.

[17] HANSON S J, PRATT L Y. Comparing Biases for Minimal Network

Construction with Back-Propagation[C].NIPS 1988 – Proceedings of the 1st International Conference on Neural Information Processing Systems.1988,1:177-185.

[18] TIELEMAN T, HINTON G. Lecture 6.5 - RMSProp: Divide the Gradient by a Running Average of Its Recent Magnitude[R].Mountain View:Coursera,2012.

[19] HINTON G, VINYALS O, DEAN J. Distilling the Knowledge in a Neural Network[EB/OL].arXiv,2015[2021-12-28].https://arxiv.org/abs/1503.02531.

[20] MACKEY D J. Information Theory, Inference and Learning Algorithms[M].Cambridge:Cambridge University Press,2003.

[21] Torch Contributors. KLDIVLOSS[EB/OL].PyTorch Documentation,2021[2021-12-28].https://pytorch.org/docs/1.10.1/generated/torch.nn.KLDivLoss.html.

[22] DOZAT T. Incorporating Nesterov Momentum into Adam[C].ICLR 2016 Workshop.2016.

[23] LOSHCHILOV I, HUTTER F. SGDR: Stochastic Gradient Descent with Warm Restarts[EB/OL].arXiv,2017[2021-12-29].https://arxiv.org/abs/1608.03983.

Acknowledgements

Throughout the capstone design, we have received a lot of support and encouragement.

We greatly appreciate the instructor Prof. An Zou from University of Michigan – Shanghai Jiao Tong University Joint Institute. He gave us a lot of support and encouragement. He is very friendly and hard-working on students' affairs. We also thank to Prof. Manuel Charlemagne for putting great workload on another course to train our perseverance.

We appreciate Yimiao Zhang and Yorushika (suis and n-buna) for the name "Elma", which is the name of our capstone design.

We would thank to our parents, who always silently support us. We would thank to our friends, without whom we may not overcome physical and psychological difficulties we met.

上海交通大学 毕业设计（学士学位论文）单独工作报告

SHANGHAI JIAO TONG UNIVERSITY

CAPSTONE DESIGN (BACHELOR'S THESIS)

INDIVIDUAL CONTRIBUTION REPORT

Student Name：Yihua Liu 刘翊华

Student ID Number：518021910998

Major: Electrical and Computer Engineering 电子与计算机工程

In our capstone design, I am the leader of our group, and I did most of the work. Generally, I am responsible for coding and deployment on the embedded system. The job duty, workload, and input are divided as:

1. I start from EdgeBERT and develop the whole question answering application based on SQuAD 2.0 dataset. I also refer to the coding architecture and implementation of HuggingFace's Transformers. I wrote the programs for this project. I set up most of the environment on NVIDIA Jetson TX2. During coding, I trained and evaluated some different models on the server for debugging.
2. Shuocheng Chen is responsible for trying to start from ALBERT TensorFlow model and testing on the server. He is also responsible for measuring the power consumption. However, TensorFlow model is highly coupled, so it is difficult to separate different layers or layer groups. Thus, it is difficult to implement early-exit from TensorFlow model. After that, he also tried ot start from HuggingFace's Transformers. HuggingFace's Transformers provides both PyTorch model and TensorFlow model. He is also the designer of the the design diagram and corresponding descriptions of this project. He fixed some bugs of the programs, and set some environments on NVIDIA Jetson TX2. He trained and evaluated different models on the server.
3. Yiming Ju is responsible for trying to start from HuggingFace's Transformers.

Our idea basically started from the concept of computation offloading. Then, we found that early-exit is a good method to do computation offloading. By exploring, we choose to use ALBERT instead of BERT because ALBERT has less model parameters, i.e., it has smaller model size, which is what we want. Besides, compared with other improvements of BERT like DistilBERT, ALBERT is the most popular one, which means that it has more resources to refer. After that, we found that EdgeBERT provides an approach to entropy-based early-exit, but it is for sequence classification. Thus, we adopted the method of entropy-based early-exit and use it to develop an ALBERT model for question answering. For the details of our application, our original goal is very high. For example, we planned to implement network sockets to

do network communcation and encrypt the data of network communication from the embedded system to the server or vice versa. We also planned to do speech recognition develop an Android application to realize more interesting functions. However, since the time is limited and we lack preliminary knowledge and skills, those insignificant features are given up. The encryption of the data of the network communication is realized by SSH connections. For network connections, we applied a public network IP, but we did not make use of that. For most time, we use a wonderful tool called “croc” to transfer files among NVIDIA Jetson TX2 board, our personal computers, and the VMware Bitfusion server.

For self-reflection, the most important one is to start early. We spend lots of time for concept generation and selection. However, the difficult part is the model training and evaluating. We needed a lot of time to debugging and testing. For training on the server, each epoch takes at least 30 minutes. More epoches are taken for 1 hour. For a good model, 15 * 2 epochs are required, and 30 * 2 epochs are recommended. To save time, we only did few trials with 15 * 2 epochs and more trials are undergoing 5 * 2 epochs. If we have more time, a better grid search could be performed, and the design and implementation could be polished.

Besides, to divide workload is also important. At first, we choose three approaches to our goal, with each of us working on one approach. However, it is not time efficient way. The best way is to work together in a discussion room from the morning to night so that the three group members can work on the same approach simultaneously. It is what we did during the last days. As the leader of our group, I communicate with the other two group members actively, but I can do it better. For example, I should discuss with them more frequently.

The topic of this capstone design is chosen by Prof. An Zou and Shuocheng Chen. At first, both Yiming Ju and I learned little about natrual langauge processing. During working on the capstone design, I learned basic concepts of the natural language processing, especially models based on BERT, which is a transformer in terms of deep learning and neural network. What I learned most is Python coding. During coding, I

learned some knowledge about TensorFlow, but most of my work is based on PyTorch. The programs construct a lot of Python classes that is at least 4-layer nesting and some Python libraries. To design and understand a complex architecture is a challenge. I drawed a huge concept draft diagram on an A4 paper and a lot of notes to help me understand. During debugging, there are a lot of bugs related to size incompatible. Python's grammar is not as clear as C or C++ programming language that I am familiar with. It is confused to identify the type of a variable. Besides, the mixture of tuples, lists, and PyTorch tensors is also a problem. Since Python debuggers are not convenient, I usually directly print the values of variables out.

I learned knowledge with respect to deep learning and neural networks. For example, I learned how did network pruning, optimzers, and warm-up scheduler work. PyTorch has provided functions and libraries of different optimizers, but I also learned the mathematical theories of principles of them. Size and types need extra considerations. This capston design is important to my future study and career, since natural language processing is an important application of deep learning and neural networks, and deep learning is very popular in different fields both academic and industrial.

In brief, as the programmer and the system maintainer of our group, I activately contribute to the project of "early-exit offloading for embedded quesiton answering applications". We worked as a group and successfully implemented a question anwering application based on ALBERT Base version 2 model, SQuAD 2.0 dataset, EdgeBERT, and HuggingFace's Transformers with entropy-based early-exit.

上海交通大学 毕业设计（学士学位论文）单独工作报告

SHANGHAI JIAO TONG UNIVERSITY

CAPSTONE DESIGN (BACHELOR'S THESIS)

INDIVIDUAL CONTRIBUTION REPORT

Student Name：Shuocheng Chen 陈烁丞

Student ID Number：517021911139

Major: Electrical and Computer Engineering 电子与计算机工程

The capstone design was a splendid cooperative work conducted by all three members, me, Yihua Liu, and Yiming Ju, of our group. I was in charge of concept generation from the beginning, while they examine which models fit for our work. At first, we together try to test and verify the performance and the feasibility of the officially released ALBERT model, while Yihua Liu tried to configure the NVIDIA Jetson TX2 into function. Then we pursued different paths to achieve our goal – the functioning ELMA model. Yiming tried to modify and upgrade HuggingFace's Transformers directly to build ELMA model, Yihua tried to learn the architecture of the implementations of EdgeBERT and then to apply it to HuggingFace's Transformers, and I tried to start from the original released code of ALBERT based on Tensorflow. In the end, Yihua's effort paid out the most and all of us focused on this way until we successfully accomplish our task – building a model with various optimizations enabling early-exiting function.

From the very beginning of our project, I played an active role in topic selection. I was the one who brought machine learning and NLP into our plot, for not only did I have a two-year lab experience in the Wireless Communication and Artificial Intelligence Laboratory in the Joint Institute from the freshman year to sophomore year, during which my main responsibility was to code and debug the data mining and machine learning programs, but also do I maintain a special enthusiasm in languages.
My enthusiasm for languages can be traced back to my middle school life when I studied French for more than a year although the only second-language option was English. I considered majoring in linguistics, languages, translations, and interpretations after Gaokao albeit I ended up studying engineering. I took a second major in English Translation and Interpretation from the School of Foreign Languages and enrolled in 4 German courses, 2 Latin courses, and a Sanskrit course during my study in SJTU. I also tried to transfer to the school of foreign languages when I was a sophomore. I'm also considering applying for another undergraduate program in German, dual-majoring in Comparative Historical Linguistics and Digital Humanities

/ Computational Linguistics / Computer Informatics. So, choosing a capstone topic relating to languages in the specified field of NLP seems so attractive and interesting to me.

When we are converting our idea into realistic practice, however, we found it very hard to get started. The vague instruction of the ALBERT code released by Google, the annoying environmental problems of NVIDIA Jetson TX2 because of its ARM architecture, the repetitive malfunctioning of the Bitfusion servers drained our time and energy in the middle of the semester when midterm exams flushed us away. Nevertheless, we moved forward thanks to Prof. ZOU An. He gathered us weekly to group meetings and reports and instructed us with inspiring ideas, and he provided kind-hearted support including an additional computational server. With his advice, guidance, and encouragement, we were able to move forward with a clear direction.

In the following weeks, programming and debugging took up our lives. It was boring yet interesting, from which I found group work so exciting and inspiring. Whenever I or my teammate encountered a coding problem, a solution was typically hard to find on our own. However, when we brought our problems to the group meeting, everything seemed so easy and so intuitive. In the valuable time of discussions, we found the problems unveiled and resolved with almost no effort other than talking and communicating. A sense of achievement and happiness always emerged during such processes.

In this capstone experience, I learned many valuable lessons.

1. Always start early. Almost every deadline is met with reluctance in our groups, for we didn't start early to leave enough time for imprecisions and accidents. For example, before the final defense, we intended to finish measuring the power consumption. However, we only left one day for it, and we found we needed a banana-DC power converter to power our NVIDIA Jetson TX2 with a digital power supply. Such converter is rarely used even in engineering practice and there might not be one single converter on the whole campus, which made us unable to deliver the

power figure in the defense.

2. Plan the project according to the time: I planned so much more at the beginning, including training our model on a vast range of various corpora, designing an audio-text converter to allow users to input with voice to demonstrate on the expo, expanding our model's question-answering ability to daily conversation instead of only SQuAD QA sets and test more optimisation methods for BERT, ALBERT, and transfer learning. In the end, we didn't even have any time to consider any of these, but we wasted much time on literature reviews in these fields

3. Value teamwork: I haven't been enjoying teamwork since my freshman year when I got an A for VG100 Introduction to Engineering. Most teamwork after that brought me only discomforts, for none of them acted really like a team. However, during this capstone experience, I found teamwork exceptionally inspiring and appealing, for there existed actual progress when we sat down together and discussed the problems we encountered.

4. Don't be afraid to bother the professors: I was always the last one to look for help from TAs or professors, for I was always afraid to bother people with my personal problems, to bring them inconvenience, or to show my incapability and weakness. However, Professor ZOU An was so kind-hearted. He was always eager to solve our problems whatever the problems are. He convinced me subconsciously that professors can be reached out for help.

5. Always seek joy in darkness: there was a time in this semester when I was devoured by the side effects of Seroquel, a medicine for Bipolar II which would result in severe drowsiness, and the disease itself. I couldn't find any hope to complete the capstone design or avoid being forced to drop out from SJTU for a lower-than-Mariana GPA. However, everything didn't go that bad as I in my worst condition predicted, which makes me believe that, next time when I am in that bottom-of-the-sea mental condition, I can believe that things can go better and find some joy and expectations.

上海交通大学 毕业设计（学士学位论文）单独工作报告

SHANGHAI JIAO TONG UNIVERSITY

CAPSTONE DESIGN (BACHELOR'S THESIS)

INDIVIDUAL CONTRIBUTION REPORT

Student Name：Yiming Ju 鞠易茗

Student ID Number：518370910059

Major: Electrical and Computer Engineering 电子与计算机工程

In our capstone design of “Early-Exit Offloading for Embedded Question Answering Applications”, I together with my two teammates: Yihua Liu and Shuocheng Chen, have worked for a large amount of time this semester. We have fulfilled an implementation of computation offloading for embedded systems on the task of question answering, which has the early-exit function. Actually, we have no idea about our capstone design at first. It was Professor Zou that put forward the feasibility of the project and guided us in a meaningful direction. In the following months, we hold our group meeting twice a week and communicated with each other about our own ideas. Besides we also have meetings once a week with Professor Zou to report the latest progress and discussed with professor about the questions we have met in this week. Thus in these three months, our professor provided us with great help.

Personally, my main work is to try different pre-trained models and add the function of early exit. At first, I clarified the requirements and plan of our projects and divided them into detailed tasks. Then I studied on the performance and parameters of popular model of recent years, like BERT, ALBERT, EdgeBERT, etc. After doing the literature review, I evaluated them from four aspects, performance, space occupation, flexibility and extensibility so that we could finally determine the pre-trained model. As introduced before, transformers of hugging face is an open free library for NLP models. After some choices of model, we chose the ALBERT with PyTorch. Actually, at the beginning, we have a quite tough time. There are dozens of files in the project. Since the code contributors used some tricky skills and python is not a very readable language, it was hard to begin understanding the realization of the code. To visualize the complex structure of the whole model, I drew a structure table of the model. After that, it was to design the algorithm of early exit. There were some similar designs of early exit for other applications. I learned about the implementation of these implementation and designed our own algorithm. To be honest, it seemed not too difficult to put the algorithm into realization. However, I miscalculated the complexity of the model. I had to print the output of some critical variables and study their relationships. Then through the clues and details I determine where I needed to

change. Finally, it was to design a control group to highlight our results. Since the training process was a quite long process, we tried to train multiple sets of data at the same time. Then according to the results of this grid search, the parameters are finally determined. Along with the joy of overcoming the difficulties and the excitement of seeing the effect, I became more and more enjoyable in this process. Meanwhile I gained confidence that I could achieve anything with enough passion.

From a technical perspective, our project has achieved the basic function we expected at first. Nevertheless, there still exist a lot of space for improvement. First, to realize the function of remote communication, I proposed to apply SSH security shell to guarantee the security of data during the communication. Although it provides a complete solution for small projects like our project due to its great efficiency for development, it is not suitable for commercial use. A new and better method is needed if our design concept is applied in reality. Second, there are still many methods we have not yet try to optimize the model for a better performance. From my perspective, the most meaningful attempt is to adjust the characteristics of parameter sharing. Actually, there is a concept called layer group in the model. I would like to try whether the performance will be improved if parameters are only shared in the same layer group rather than the whole model. This improvement will enable the project to be more complete and have better performance.

I think in this process, from a beginner who didn't know anything about this field to one who is gradually familiar with this field, I have learnt a lot. In the initial stage, even the environment configuration seemed intractable and cost me a lot of time to deal with a variety of strange errors. For example, different models need different running environment, like different versions of libraries. Since for some models, the contributors have already stopped maintenance of code availability and code quality, I have to find conflicts between different versions and resolve them by consulting the official documentation. In the later, I could clearly point out the method of important implementation of each function in the model. In this way, I learned a lot of related skills and knowledge. I think this project is an examination of what I have learned

over the years, especially in the course of advanced embedded systems. And learning is a process of accumulation. I see my own shortcomings and areas that need to be improved, and at the same time, I learned a lot of experience from other students' performance. Besides, another most valuable things about this experience was that it provided me with the opportunity to enter the world of not only NLP but also embedded systems. It pushes me to a higher level of understanding in this field. It also taught me to remain enthusiastic and patient when doing such a project.

In conclusion, I have a deeper understanding of natural language processing and embedded systems in this semester. During the learning and practicing, I saw the promising market of the combination of NLP and embedded system in the future, like the chat bots. In addition, I learned about not only how to use the ready-made model but also how to revise the model and add our own functionality. It was a precious experience to learn new things in such a way. In the cooperation with my teammates, I understood the meaning of teamwork and skills of communication. I believe it will help me a lot in my future study and career.