BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

EFFECTIVE TRAINING OF NEURAL NETWORKS FOR AUTOMATIC SPEECH RECOGNITION

EFEKTIVNÍ TRÉNOVÁNÍ NEURONOVÝCH SÍTÍ PRO AUTOMATICKÉ ROZPOZNÁVÁNÍ ŘEČI

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **MATEJ HORNÍK**
AUTOR PRÁCE

SUPERVISOR **Ing. ALEXANDER POLOK,**
VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

164401

Institut:	Department of Computer Graphics and Multimedia (DCGM)
Student:	**Horník Matej, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Machine Learning
Title:	**Effective Training of Neural Networks for Automatic Speech Recognition**
Category:	Speech and Natural Language Processing
Academic year:	2024/25

Assignment:

1. Familiarize yourself with the fundamentals of machine learning and automatic speech recognition.
2. Train an Encoder-Decoder model for speech recognition using one of the freely available datasets (e.g., VoxPopuli).
3. Utilize pre-trained acoustic models (e.g., XLS-R) and language models (e.g., GPT-2) to initialize the network and compare whether it is advantageous to initialize the model from pre-trained models.
4. Determine which network parameters are suitable for updating and which can be left unchanged without significant loss in recognition accuracy.
5. Analyze whether the models converge faster if individual components of the model (the acoustic and language model) are first fine-tuned on the given corpus and only then combined.
6. Publish at least one of the trained models.

Literature:

- Baevski, A., Zhou, Y., Mohamed, A. a Auli, M. Wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. In: Advances in Neural Information Processing Systems. Curran Associates, Inc., 2020, sv. 33, s. 12449–12460.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. et al. Language models are unsupervised multitask learners. In: OpenAI blog. 2019, sv. 1, č. 8, s. 9.

Requirements for the semestral defence:
points 1 and 2 completed, point 3 in progress

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Polok Alexander, Ing.**
Head of Department:	Černocký Jan, prof. Dr. Ing.
Beginning of work:	1.11.2024
Submission deadline:	21.5.2025
Approval date:	12.11.2024

Abstract

This master's thesis focuses on improving the training efficiency and performance of encoder-decoder transformer models for Automatic Speech Recognition (ASR). It investigates the impact of initialization strategies using pre-trained components (Wav2Vec2, BART), the role of convolutional adapters, and Parameter-Efficient Fine-tuning (PEFT) methods like LoRA and DoRA. Experiments on LibriSpeech and VoxPopuli datasets confirmed that full pre-trained initialization is crucial for best Word Error Rate (WER) and convergence. An optimal number of adapters improved performance, while PEFT (especially LoRA) significantly reduced trainable parameters with comparable accuracy. Domain-specific encoder pre-training proved beneficial, and the encoder-decoder model outperformed a CTC baseline in accuracy. Notably, an optimized configuration achieved a Word Error Rate of 8.85% on the VoxPopuli English test set. These findings offer practical insights for efficient ASR training.

Abstrakt

Táto diplomová práca sa zaoberá zlepšením efektivity trénovania a výkonu modelov kodér-dekodér pre automatické rozpoznávanie reči (ASR) s využitím transformer modelov. Skúmal sa vplyv inicializačných stratégií s predtrénovanými komponentmi (Wav2Vec2, BART), úloha konvolučných adaptérov a metódy parametricky efektívneho doladenia (PEFT) ako LoRA a DoRA. Experimenty na dátových sadách LibriSpeech a VoxPopuli potvrdili, že plná predténovaná inicializácia je kľúčová pre najlepšiu slovnú chybovosť (WER) a konvergenciu. Optimálny počet adaptérov zlepšil výkon, zatiaľ čo PEFT (najmä LoRA) výrazne znížilo počet trénovateľných parametrov pri zachovaní porovnateľnej presnosti. Predtrénovanie kodéru na dátach cieľovej domény bolo prínosné a architektúra kodér-dekodér prekonala CTC model v presnosti. Optimalizovaná konfigurácia dosiahla slovnú chybovosť 8.85% na testovacej sade VoxPopuli English. Tieto zistenia poskytujú praktické poznatky pre efektívny tréning ASR.

Keywords

automatic speech recognition, deep learning, transformer models, encoder-decoder models, pre-trained models, parameter-efficient fine-tuning, PEFT, LoRA, DoRA, adapter layers, initialization strategies, Wav2Vec2, BART, word error rate, WER, sequence-to-sequence learning, self-supervised learning, cross-attention, fine-tuning, LibriSpeech, VoxPopuli, computational efficiency, neural networks, language models

Klíčová slova

automatické rozpoznávanie reči, hlboké učenie, transformer modely, modely kódovač-dekóder, predtrénované modely, parametricky efektívne doladenie, PEFT, LoRA, DoRA, adaptéry, inicializačné stratégie, Wav2Vec2, BART, slovná chybovosť, WER, učenie typu sekvencia-na-sekvenciu, učenie s vlastným dohľadom, krížová pozornosť, doladenie, LibriSpeech, VoxPopuli, výpočtová efektivita, neurónové siete, jazykové modely

Reference

HORNÍK, Matej. *Effective Training of Neural Networks for Automatic Speech Recognition*. Brno, 2025. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Alexander Polok,

Rozšířený abstrakt

Táto diplomová práca sa zameriava na optimalizáciu tréningového procesu a zlepšenie výkonnosti modelov hlbokého učenia pre automatické rozpoznávanie reči (ASR), špecificky sa sústredí na architektúry typu kodér-dekodér založené na transformeroch. Súčasné najvýkonnejšie ASR modely, často postavené na architektúre transformer, dosahujú vynikajúce výsledky, merané nízkou slovnou chybovosťou (Word Error Rate — WER), avšak ich tréning na rozsiahlych dátových sadách si vyžaduje enormné výpočtové zdroje. Táto práca skúma stratégie, ako zmierniť túto výpočtovú náročnosť a zároveň zlepšiť presnosť modelov pre špecifické úlohy ASR. Hlavným prístupom je využitie a kombinácia vopred predtrénovaných komponentov – kodéra, ktorý sa naučil robustné reprezentácie zvuku, a dekodéra, predtrénovaného na úlohách jazykového modelovania. Cieľom práce je systematicky preskúmať rôzne inicializačné stratégie, význam prepojovacích mechanizmov medzi kodérom a dekodérom a aplikáciu metód parametricky efektívneho doladenia (Parameter-Efficient Fine-tuning — PEFT) na zefektívnenie procesu adaptácie modelu.

Teoretická časť stručne predstavuje tradičné metódy spracovania reči a ich obmedzenia, následne sa podrobne venuje základom hlbokého učenia pre sekvenčné dáta, s ťažiskom na architektúru Transformer. Detailne rozoberá kľúčové komponenty ako mechanizmus pozornosti, embeddings, pozičné kódovanie a tokenizáciu (BPE). Opisuje funkcie kodérovej a dekodérovej časti, vrátane ich špecifických mechanizmov. Práca ďalej preberá dominantné end-to-end ASR architektúry: Connectionist Temporal Classification (CTC), RNN Transducer (RNN-T) a Attention-based Encoder-Decoder (AED), pričom porovnáva ich princípy a vlastnosti. V kontexte týchto architektúr predstavuje konkrétne predtrénované modely: akustický model Wav2Vec 2.0 a jazykový model BART, ktoré slúžia ako základné stavebné bloky pre experimenty. Zároveň sú predstavené metódy PEFT, ktoré umožňujú adaptovať rozsiahle modely s trénovaním len malého zlomku parametrov, s dôrazom na LoRA (Low-Rank Adaptation) a DoRA (Weight-Decomposed Low-Rank Adaptation).

Experimentálna časť práce sa venuje systematickému vyhodnoteniu navrhovaných prístupov na anglických ASR dátových sadách LibriSpeech ('train-clean-100') a VoxPopuli (anglická časť). Základným modelom je kombinácia predtrénovaného kodéra Wav2Vec2-base a dekodéra BART-base, hodnoteným metrikou WER. Prvý experiment potvrdil, že inicializácia s oboma predtrénovanými časťami dosahuje najlepšiu WER (8.9% LibriSpeech, 10.6% VoxPopuli) a najrýchlejšiu konvergenciu, pričom predtrénovaný kodér má väčší vplyv. Druhý experiment ukázal, že zmrazenie príznakového extraktora v kodéri neškodí výkonu a mierne znižuje výpočtovú náročnosť, preto bol v ďalších experimentoch zmrazený. Tretí experiment identifikoval optimálny počet (2-3) konvolučných adaptačných vrstiev medzi kodérom a dekodérom pre zlepšenie WER (najlepšie 8.08% LibriSpeech, 10.65% VoxPopuli) bez výrazného vplyvu na rýchlosť tréningu.

Štvrtý a piaty experiment hodnotili metódy PEFT. LoRA s vhodne zvoleným rankom a škálovacím faktorom dosiahla výkon blízky plnému doladeniu (8.67% WER LibriSpeech) s výrazne menej trénovateľnými parametrami, aj keď s pomalším tréningovým krokom. DoRA dosiahla mierne horšiu WER a bola výrazne pomalšia ako LoRA v tomto nastavení. Šiesty experiment ukázal, že použitie kodéra Wav2Vec2 predtrénovaného priamo na cieľových dátach (anglický VoxPopuli) signifikantne zlepšilo výsledky na VoxPopuli (WER 9.88%). Finálne optimalizované trénovanie tejto konfigurácie s rozsiahlejšími parametrami doladenia a technikou SpecAugment ďalej znížilo WER na anglickom VoxPopuli na 8.85%. Dodatočné predtrénovanie dekodéra na cieľových textových dátach však neprinieslo zlepšenie. Posledný experiment porovnal model kodér-dekodér s baseline CTC modelom, pričom kodér-dekodér dosiahol výrazne lepšiu WER (napr. 8.26% vs 11.87% LibriSpeech), aj keď bol pomalší.

Záverom, práca potvrdila kľúčový význam využitia predtrénovaných komponentov, prínos konvolučných adaptérov a efektivitu PEFT metód (najmä LoRA) pre tréning ASR modelov typu kodér-dekodér. Zdôraznila dôležitosť doménovej špecificity predtréningu kodéra a vyššiu presnosť architektúry kodér-dekodér oproti CTC. Výsledky poskytujú praktické odporúčania pre efektívnejšie trénovanie a nasadzovanie moderných ASR systémov.

Effective Training of Neural Networks for Automatic Speech Recognition

Declaration

I declare that I have prepared this master's thesis independently under the guidance of Ing. Alexander Polok. During the preparation of the textual parts, specifically for refining language, generating suggestions, and proofreading, I utilized assistance from generative language models. I have cited all literary sources, publications, and other resources used.

.....
Matej Horník
May 18, 2025

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Ing. Alexander Polok, for his valuable guidance, insightful advice, and support throughout the development of this thesis.

Contents

List of Figures

Chapter 1

Introduction

The field of Automatic Speech Recognition (ASR), which focuses on converting spoken language into text, has been fundamentally transformed by deep learning techniques. Replacing traditional approaches with deep neural networks, particularly architectures like transformers [124] have driven substantial improvements in accuracy, often measured by reductions in Word Error Rate (WER) on speech recognition datasets [70, 62]. This performance leap enables the widespread deployment of ASR in applications such as voice assistants, large-scale transcription services, and human-computer interaction systems.

This thesis focuses on improving the training efficiency and performance of deep learning-based speech recognition models. Most of the best-performing models used today for this task are based on transformer architecture, specifically those employing encoder-decoder architecture. These models were trained on large datasets that require massive computing power. Using pre-trained models that were trained on different objectives and combining them can mitigate this issue, and we can quickly fine-tune a model for an ASR task. For example, the encoder can be pre-trained on a task that emphasizes learning robust audio representations. In contrast, the decoder can be pre-trained on a language modeling task to enhance its ability to generate coherent text. With the correct strategy when initializing the model with both parts, we can potentially accelerate training and achieve better performance on specific speech recognition tasks.

The thesis focuses on experiments with different initialization strategies on the effectiveness of encoder-decoder speech recognition models. The primary research objective is to determine the optimal initialization strategy for training encoder-decoder ASR models by using pre-trained encoder and decoder parts of transformer architecture. The performance of each initialization strategy is evaluated on word error rate, training time, and convergence speed.

This thesis is structured as follows: Chapter 2 reviews foundational speech processing concepts, focusing on deep learning techniques and key components of the transformer architecture. Chapter 3 then delves into specific end-to-end techniques for ASR such as CTC, RNN-T, and AED models. Subsequently, Chapter 4 introduces relevant pre-trained acoustic and language models and discusses Parameter-Efficient Fine-tuning (PEFT) methods such as LoRA and DoRA for their adaptation. The core experimental results are presented in Chapter 5, covering used datasets, initialization strategies, convolutional adapters, PEFT comparisons, pre-training domain effects, a CTC baseline comparison, and an optimized run on the VoxPopuli dataset. Finally, Chapter 6 summarizes the main findings, discusses limitations, and outlines potential future work.

Chapter 2

Theoretical Foundations

In this chapter, the theoretical foundations of speech processing are outlined. First, there is a small introduction to traditional models, and then in much more detail, deep learning techniques focused on transformer architecture and its details. Key components of the transformer are examined, including various forms of the attention mechanism, methods for input representation like embeddings and positional encodings, and finally, the specific roles of the encoder and decoder modules.

2.1 Traditional Models For Speech Processing

This section is based on the following articles and surveys for traditional models used in speech processing [82, 97, 8].

Traditional speech processing models, such as **Gaussian Mixture Models** (GMMs) [10], **Hidden Markow Models** (HMMs) [32], and other techniques, have been widely applied in speech recognition. These models rely on hand-crafted features to capture significant characteristics of speech data. While they have achieved notable success in speech recognition, their limitations, such as complexity, reliance on domain expertise, and reduced accuracy in noisy or ambiguous speech data, have become apparent. These traditional models have been increasingly replaced by better representation learning algorithms, particularly deep neural networks, which offer improved performance in speech processing tasks.

These traditional algorithms operate on acoustic features typically extracted beforehand using specialized signal processing techniques, rather than learning representations directly from raw data like deep models do. These techniques aim to capture significant characteristics of speech data through mathematical operations such as Fourier transforms, wavelet transforms, and linear predictive coding (LPC) [89]. Extracted features are then used as inputs primarily for classification models. These shallow models extract useful information from speech signals, subsequently enabling classification and regression models to learn patterns and make accurate predictions.

Traditional ASR systems often relied on Gaussian Mixture Models (GMMs) [10] to model the probability distribution of speech features using a combination of weighted Gaussians, each capturing distinct acoustic traits. Another key method was Hidden Markov Models (HMMs) [32], which modeled the temporal sequence of speech through hidden states linked to observable features. HMMs were typically trained with the Baum-Welch

algorithm [136], a form of Expectation-Maximization for estimating parameters with latent variables.

Despite their early successes, traditional speech processing models exhibit several key limitations. A primary drawback stems from their dependence on hand-crafted features, a process that is not only time-consuming but also requires significant domain expertise, ultimately limiting the scalability and adaptability of these models to ASR [48, 137]. Furthermore, the complexity of these traditional models tends to increase substantially as the intricacy of speech patterns grows, which can become a bottleneck hindering their performance, particularly when dealing with large and complex datasets [33, 137]. These models also often struggle with real-world variability, encountering difficulties when processing noisy or ambiguous speech data, especially in environments characterized by significant background noise [14]. Finally, traditional speech processing typically involved pipelines with separate, distinct components for stages such as acoustic feature extraction, acoustic modeling, and language modeling. This modular separation inherently complicates the overall system architecture and makes holistic optimization across the entire process significantly more challenging [82].

2.2 Deep Learning Architectures

Deep learning architectures have revolutionized the field of speech processing by demonstrating much better performance across various tasks compared to traditional methods. With their ability to automatically learn hierarchical representations from raw speech data, deep learning models have surpassed traditional approaches in speech recognition [82, 97, 8].

One prominent deep learning model architecture for speech processing that is used in most tasks is transformer [124]. Other models like recurrent neural networks (RNNs) [108], gated recurrent neural networks (GRUs) [18], long short-term memory (LSTMs) [50] and bidirectional LSTMs [56] were previously popular in speech processing. They faced limitations in capturing long-term dependencies and efficiently leveraging parallel computing hardware, whereas transformer-based models can effectively model long sequences and take advantage of parallel computing. Convolutional neural networks (CNNs) [31, 64] have also been used in speech recognition, often in combination with other architectures.

In this section, the basics of RNNs for speech processing will be described, and later, details of the transformer model and its attention mechanism will be provided since this method is heavily used for processing sequences.

2.2.1 Neural Networks for Sequence Processing

Tasks in speech processing are expressed as sequence-to-sequence problems. These tasks involve processing sequences where the elements (like tokens or acoustic features) are not independent, but rather, there are dependencies between them that influence the sequence's meaning or structure. Sequence-to-sequence problems can be solved by finding a mapping f from an input sequence of n vectors $\mathbf{X}_{1:n}$ to a sequence of m target vectors $\mathbf{Y}_{1:m}$, where the number of target vectors m is unknown apriori and depends on the input sequence:

$$f : \mathbf{X}_{1:n} \rightarrow \mathbf{Y}_{1:m}. \tag{2.1}$$

Problems with variable number of inputs and outputs makes traditional neural networks unsuitable for this task. This would, therefore, mean that the number of target vectors m

has to be known a priori and would have to be independent of the input $\mathbf{X}_{1:n}$. For example, in ASR, the length of input speech signal can vary as the length of the transcribed text.

Recurrent Neural Networks

Recurrent neural networks [34] are capable of modeling time-varying patterns, which are often difficult for standard neural architectures to capture effectively. A key feature of RNNs is that they include connections that create directed cycles (loops), known as recurrent connections. With this feature, it allows information to persist across time steps.

Recurrent neural networks process input sequences sequentially, maintaining a hidden state that captures information from past inputs. While this describes the basic causal flow, many approaches, especially for offline tasks, also leverage future context using architectures like Bidirectional RNNs. The behavior of a simple (Vanilla) RNN can be expressed by the following equations, where the hidden state h_t and the output y_t at time t are computed as follows:

$$h_t = \Phi(\mathbf{W}_{hh} h_{t-1} + \mathbf{W}_{xh} x_t) \tag{2.2}$$

$$y_t = \mathbf{W}_{hy} h_t \tag{2.3}$$

where $\mathbf{W}_{hh}$, $\mathbf{W}_{xh}$, and $\mathbf{W}_{hy}$ are the weight matrices that govern the transformations between the hidden states and input, as well as from the hidden state to the output. The function (Φ) is a non-linear activation function, such as Tanh, ReLU, or Sigmoid [35], which introduces non-linearity into the model.

The recursive nature of RNNs, highlighted by the term h_t in the equation above, enables them to retain information from past inputs $\{x_k\}_{k=1}^{t-1}$. This capability acts as a form of memory, allowing the network to model dependencies across time steps.

Recurrent neural networks suffer from issues such as vanishing and exploding gradient problems when trying to capture long-term dependencies. To solve these problems, an improved version of RNN was introduced called Long Short Term Memory (LSTM) [50]. A primary characteristic of this network is its capacity to retain information over extended periods. They use a gating mechanism and memory cells to control the flow of information and relieve gradient problems. Many LSTM variations exist [112]. Gated Recurrent Units [19] had also been introduced to combat the gradient problem. They aim to be more computationally efficient than LSTM models.

As stated in 2.2.1, speech processing challenges involve having varying input and output lengths. A special type of architecture was created to tackle the problem of sequence-to-sequence problems [122]. It is made of two modules: an encoder and a decoder [16]. The encoder-decoder architecture represents an approach for sequence-to-sequence tasks where both input and output sequences can have variable lengths. The encoder, in the original paper represented by LSTM, processes the input sequence $\mathbf{X}_{1:n}$ to a context vector c through the mapping:

$$f_{\theta_{enc}} : \mathbf{X}_{1:n} \rightarrow c \tag{2.4}$$

This encoding process is achieved by successively updating the encoder's hidden state. The decoder module then models the label sequence posterior probability given the encoded representation c by the encoder. It functions auto-regressively, meaning each output

depends on previous outputs and the encoded state c. The model sequentially maps the previous inner hidden state c_{i-1} and the previous target vector y_{i-1} to the current inner hidden state c_i and a logit vector $\mathbf{l_i}$:

$$f_{\theta_{dec}}(y_{i-1}, c_{i-1}) \rightarrow \mathbf{l_i}, c_i \tag{2.5}$$

c_0 is thereby defined as c being the output hidden state of the encoder. The decoder defines the probability distribution of a target sequence $\mathbf{Y}_{1:m}$ given the hidden state c, using Bayes rule, this can be expressed as a condition distribution of a single target vector y_i as follows:

$$p_{\theta_{dec}}(Y_{1:m}|c) = \prod_{i=1}^{m} p_{\theta_{dec}}(y_i|Y_{0:i-1}, c) \tag{2.6}$$

where the conditional probability $p_{\theta_{dec}}(y_i|Y_{0:i-1}, c)$ represents the probability of generating each output token, which is modeled using a nonlinear transformation of the current decoder hidden state c_i, the encoder context vector c, and the previously generated token y_{i-1}.

Since this model operates auto-regressively, the decoder requires special tokens to manage sequence boundaries. A Beginning-of-Sequence (`BOS`) token acts as the initial input prompt for the decoder to start generating the first token, while an End-of-Sequence (`EOS`) token indicates when generation should end. These tokens serve as cues to the network. This mechanism ensures coherent sequence generation and is particularly crucial for handling variable-length sequences. A high-level overview of the encoder-decoder model is depicted in Figure 2.1.

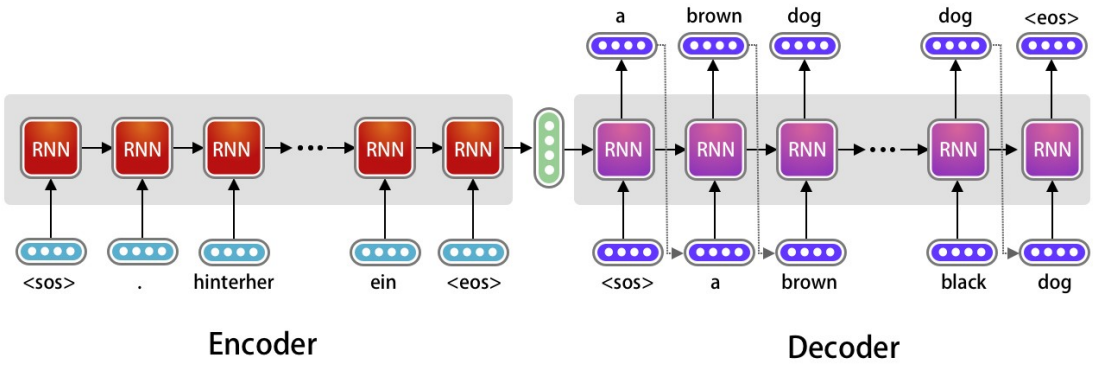


Figure 2.1: Encoder-Decoder Model for machine translation. Adapted from [1]

RNNs and their variants are foundational to speech recognition in the deep learning sphere [82]. Nevertheless, even successful sequence-to-sequence models that leverage these RNN architectures grapple with slow training times and precise capturing of long-term dependencies remains difficult [35].

2.3 Transformer

The introduction of the transformer architecture marked a paradigm shift in sequence-to-sequence modeling. Originally introduced by *Vaswani et al. in Attention is All You Need* [124], the transformer addressed many limitations inherent in RNNs. Significantly accelerating training times, it entirely eliminated recurrence, allowing for parallelization during training. Transformers rely on the attention mechanism, described in more detail in 2.3.1, which allows the model to directly compute relationships between any position in the entire sequence, which enables better capture of long-range dependencies compared to sequential architectures like RNNs. This parallel processing capability offers a critical advantage, particularly during training. Unlike the sequential processing inherent in RNNs, transformers can compute representations for all sequence positions simultaneously, enabling training acceleration on parallel hardware. Additionally, the multi-head attention component captures relationships across different segments of the input sequence, whereas positional encodings inject information about token order to preserve the sequence's structural integrity. This helps address the lack of inherent position representation in traditional sequence-to-sequence models.

As transformers gained prominence, they proved effective in speech recognition and other similar tasks that require processing data as sequences. They surpassed RNN-based models in performance, scalability, and efficiency. The encoder-decoder structure of the transformer, similar to the seq2seq RNN models, enables the transformer models to process and output sequences of different lengths. Given these advantages over RNN-based models, the transformer has become a foundational architecture in many tasks. It often serves as the backbone for state-of-the-art models in speech processing tasks [67]. The architecture of the original transformer is pictured in Figure 2.2.

2.3.1 Attention mechanism

Attention is a method used to determine the importance of parts within a sequence. It is represented by weights assigned to each part in a sequence. In language processing, attention scores are calculated for each token on how significant they are for the currently processed token in a sequence. Models can selectively attend to different parts of the input data, assigning varying degrees of importance or weight to specific elements. Its inspiration comes from the attention of humans, where we prioritize certain stimuli or thoughts to the exclusion of others[20].

The first work that resembled the attention mechanism as we know it today was published in work from 1998 called *A Model of Saliency-Based Visual Attention for Rapid Scene Analysis* [57]. Handcrafted features were used to create saliency maps, which guided another neural network in analyzing image patches based on importance.

A key aspect of attention can be expressed by the following:

$$\sum_i \langle (\text{query})_i, (\text{key})_i \rangle (\text{value})_i \tag{2.7}$$

where, the angled brackets represent dot product between **query** and **key**.

The theoretical foundations of modern attention mechanisms can be traced to earlier architectural paradigms where a dual-network system demonstrated dynamic weight adaptation. In this framework, one network operates as a meta-learner, utilizing gradient descent to generate key-value pairs that control the weight dynamics of a second network.

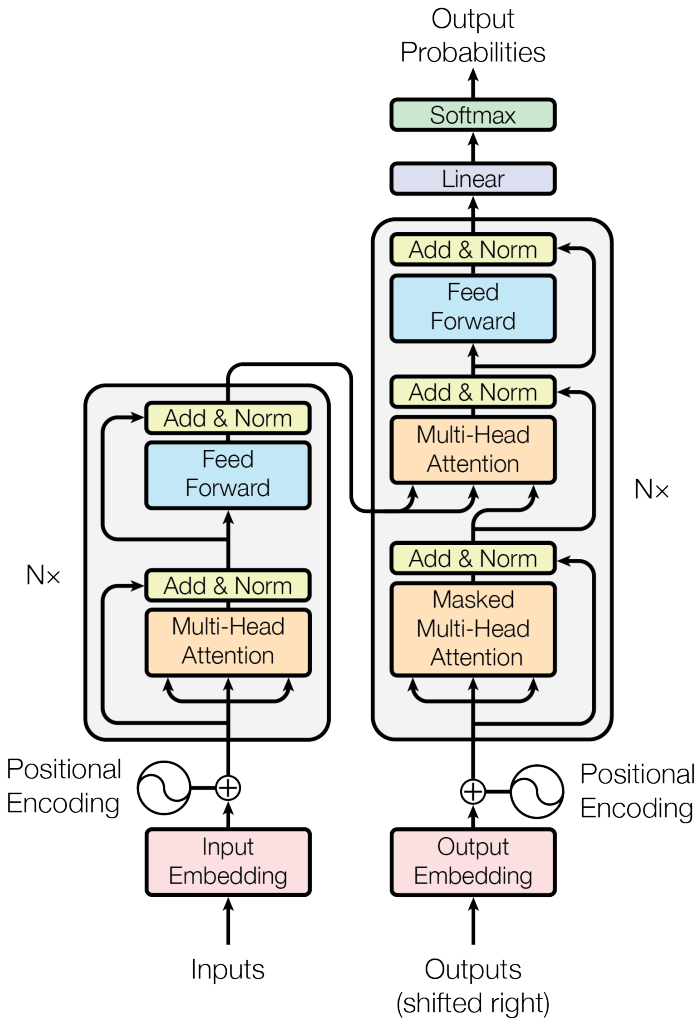


Figure 2.2: Architecture of encoder-decoder based transformer as published in the original paper. Figure taken from [124].

This second network computes answers to queries within the dynamically adjusted space [49, 42, 114]. The mathematical formulation of this weight-modulation mechanism was subsequently proven to be equivalent to unnormalized linear transformer architectures [113].

The attention mechanism was first used to solve tasks in machine translation. It used a sequence-to-sequence model with encoder-decoder architecture. *Bahdanau et al.* [6] identified that using fixed-length vectors created a bottleneck in the basic RNN encoder-decoder architecture. They proposed a novel approach that allowed the model to automatically search for relevant parts of a source sentence when predicting target words without requiring explicit segmentation. This approach achieved state-of-the-art results for machine translation at that time.

The idea of attention was expanded in work by *Xu et al.* in 2015 [131]. They used an attention mechanism on images, where different regions or *patches* with fixed dimensions of an input image were dynamically weighted, allowing the model to focus on relevant parts of the image as it produced each word of the description.

The first innovation in [124] is called self-attention, which fundamentally transformed neural network design by allowing parallel computation of attention weights through matrix multiplications. This approach enabled more efficient sequential data processing by computing attention through dot products of query, key, and value matrices.

The attention operates through three interconnected vector components: a query vector, a key vector, and a value vector. Each vector serves a specific computational purpose in deriving the attention output. When implemented, these vectors are typically arranged into comprehensive matrices representing queries ($\mathbf{Q}$), keys ($\mathbf{K}$), and values ($\mathbf{V}$). Using matrices for query, key, and value vectors enables simultaneous attention computation across multiple tokens, allowing neural networks to capture contextual relationships more efficiently. The scaled dot-product attention characteristic of transformer architectures is mathematically expressed through the formula:

$$\begin{aligned} \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \\ \mathbf{Q} &= \mathbf{W}_Q\mathbf{X} \\ \mathbf{K} &= \mathbf{W}_K\mathbf{X} \\ \mathbf{V} &= \mathbf{W}_V\mathbf{X} \end{aligned} \tag{2.8}$$

where the variables: $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ representing the Query, Key, and Value matrices, respectively; $\mathbf{W}_Q \in \mathbb{R}^{d_{\text{model}} \times d_q}$, $\mathbf{W}_K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\mathbf{W}_V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ are the projection weight matrices for Query, Key, and Value, respectively; and $\mathbf{X} \in \mathbb{R}^{n \times d_{\text{model}}}$ is the input embedding matrix, where each row represents the embedding of a token in the sequence.

The dimensions involved are: n, the sequence length (i.e., the number of tokens in the input); d_{model}, the dimensionality of each token's embedding vector; and d_q, d_k, d_v, the dimensionalities of the Query, Key, and Value vectors, respectively.

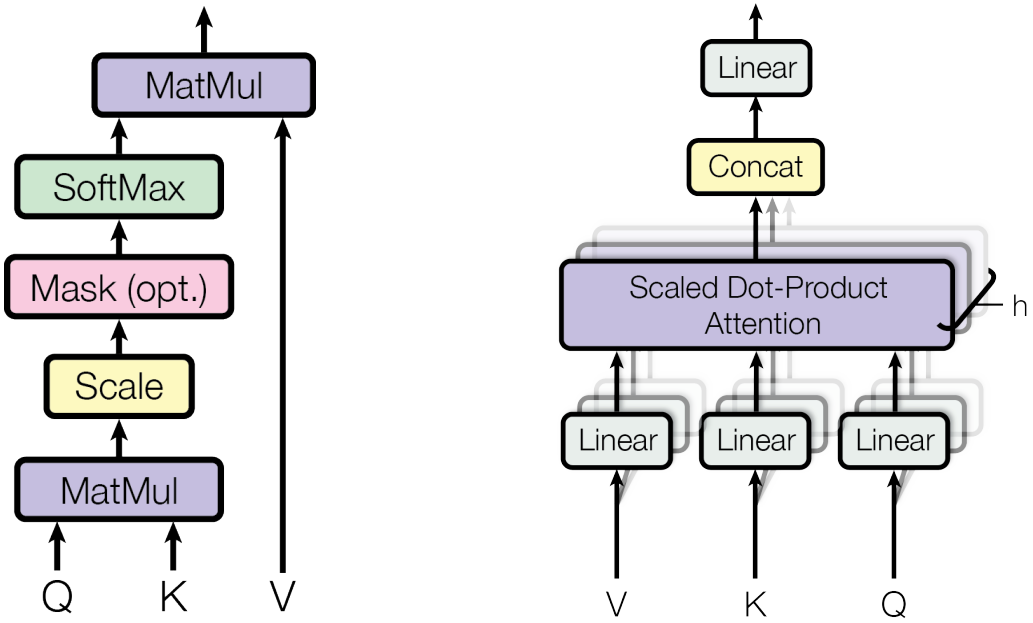
For normalization, the scaling factor $\frac{1}{\sqrt{d_k}}$ prevents excessively small gradients in the softmax computation, which could hinder learning, particularly for long sequences. This ensures stable updates during backpropagation.

A computation diagram of the scaled dot-product attention is shown in Figure 2.3a.

The second innovation that was introduced is a mechanism called Multi-head attention, which improves the model's ability to capture various relationships within the data. In the multi-head attention, input queries, keys, and values are segmented into h parallel processing paths, with each head h_i possessing unique projection matrices $\mathbf{Q}_i$, $\mathbf{K}_i$ and $\mathbf{V}_i$. Instead of using a single attention function with d_{model} dimensional keys, values, and queries, Multi-head attention projects the queries, keys, and values into different subspaces, with dimensions d_k, d_k, and d_v, respectively. Attention is then computed in parallel for each projection, producing outputs that are concatenated and projected to generate the final result. This approach enables the model to attend to multiple aspects of the input simultaneously, as illustrated in Figure 2.3b.

Mutli-head attention is represented by following formula:

$$\begin{aligned} \text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \\ \text{where head}_i &= \text{Attention}(\mathbf{Q}W_i^Q, \mathbf{K}W_i^K, \mathbf{V}W_i^V) \end{aligned} \tag{2.9}$$



(a) Scaled dot-product attention.

(b) Multi-head attention block.

Figure 2.3: Attention Mechanisms in Transformers: Scaled Dot-Product and Multi-Head Attention. **Q**, **K** and **V** denotes the query, key and value inputs. Figures taken from [124].

where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. Dimensions of d_k, and d_v are d_{model}/h. The W^O matrix serves to linearly project the concatenated multi-head attention outputs back to the original model dimension d_{model}, allowing the multi-head attention to integrate and transform the parallel attention computations into a single, coherent matrix representation.

There are small differences in the encoder and decoder attention. In the encoder, self-attention, as described earlier, allows each position in the input sequence to attend to all other positions. This is particularly important in machine translation, where variations in word order and syntax between languages require the model to capture long-range dependencies across the entire input sequence.

The decoder's attention mechanism differs depending on its role. When used for autoregressive sequence generation it employs masked attention, also called causal attention. This is fundamentally important for autoregression because it prevents positions from seeing future tokens in the sequence during training and generation. By blocking attention to subsequent positions, it ensures causality – the current prediction depends only on past information. This masking mechanism in attention is done by matrix M to zero out attention scores for future tokens:

$$M_{ij} = \begin{cases} 0 & \text{if } i < j, \\ -\infty & \text{if } i \geq j. \end{cases} \tag{2.10}$$

Matrix M is then added to the calculation of attention:

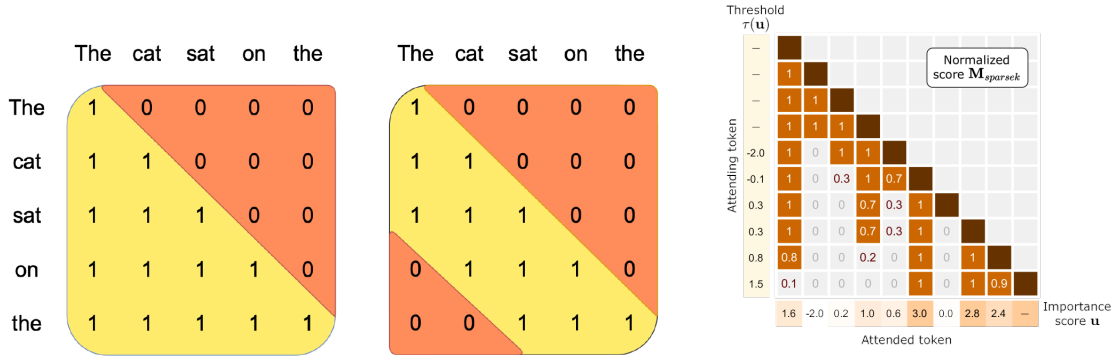
$$\text{Masked Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T + M}{\sqrt{d_k}}\right)\mathbf{V} \tag{2.11}$$

With this small difference, the decoder maintains an autoregressive property that is vital for generating sequences in a step-by-step manner.

In addition to this masked self-attention mechanism, each decoder layer also incorporates a cross-attention mechanism. This is a link that allows the decoder to incorporate information from the encoder. In this cross-attention sub-layer, the **query** vectors (**Q**) originate from the output of the preceding decoder sub-layer. However, the **key** (**K**) and **value** (**V**) vectors are derived directly from the encoder's contextual representations. The computation uses the same scaled dot-product attention formula:

$$\text{Cross-Attention}(\mathbf{Q}_{\text{dec}}, \mathbf{K}_{\text{enc}}, \mathbf{V}_{\text{enc}}) = \text{Softmax}\left(\frac{\mathbf{Q}_{\text{dec}}\mathbf{K}_{\text{enc}}^T}{\sqrt{d_k}}\right)\mathbf{V}_{\text{enc}} \tag{2.12}$$

This mechanism enables the decoder, at each step, to dynamically focus on the most relevant parts of the encoded input sequence. Unlike self-attention, cross-attention computes relationships between two different sequences: the decoder's intermediate representations and the encoder's final output.



(a) Masked attention and sliding window attention. Figure taken from [59].

(b) Sparse attention. Figure taken from [79].

Figure 2.4: Different attention mechanisms used in transformer models.

2.3.2 Improvements in Attention mechanism

Different attention mechanisms have been created to explore more computationally efficient and contextually nuanced alternatives. One strategy called sliding window attention was introduced together with the family of language models from the company *Mistral AI*[1]. It restricts attention computation to only the last n elements in a sequence, which enables more efficient processing of long-range dependencies in sequence [59]. This is shown in Figure 2.4a. Sparse attention also tries to address the efficiency of computational resources by reducing the number of attention connections between tokens [138]. This approach uses intelligent sparsity patterns to remove some connections. Sparse attention is depicted in Figure 2.4b.

Similar to attention mechanism improvements, more efficient techniques have been introduced to improve multi-head attention. Attention mechanism innovations have further

[1]https://mistral.ai/

improved computational efficiency and model performance through techniques like multi-query attention (MQA) and grouped-query attention (GQA). Multi-query attention modifies the multi-head attention approach by sharing key and value projection matrices across all heads, reducing computational complexity together with computational resources. This, however, comes with slightly lower accuracy scores for models [133]. Grouped-query attention [63] represents a better compromise between traditional multi-head and multi-query attention, where a subset of heads share key and value projections, which offers a balanced approach to reducing computational overhead. The grouped-query attention can be mathematically represented as:

$$\begin{aligned} \text{GroupedQueryAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \text{Concat}(\text{group}_1, \ldots, \text{group}_g) W^O \\ \text{where group}_j &= \text{Attention}\left(\mathbf{Q}W_j^Q, \underbrace{\mathbf{K}W_g^K, \mathbf{V}W_g^V}_{\text{shared within group}}\right) \end{aligned} \tag{2.13}$$

where, g represents the number of groups, with each group sharing key and value projections, allowing for a flexible trade-off between computational efficiency and model robustness. It was shown that this improves the model performance while maintaining the same model's ability.

2.3.3 Embeddings

Embeddings are a foundational component within transformer architectures, responsible for converting discrete input tokens into dense, continuous vector representations. This initial transformation is crucial because simpler methods, like one-hot encoding, while uniquely identifying tokens, produce sparse, high-dimensional vectors that fail to capture inherent semantic or syntactic relationships between them [9]. These learned embeddings encode such information so that relationships between tokens (like similarity, analogy, or contextual relatedness) are often captured by the geometric relationships (e.g., distance, orientation) between their corresponding vectors in the embedding space. The dimensionality of these vectors is a key hyperparameter, commonly set between 256 and 4096, influencing the model's capacity and computational cost [7, 123].

Static word embedding methods, including Word2Vec [84], GloVe [94], and FastText [11], generate a single vector representation for each token. This means a word will have the same embedding regardless of its surrounding words or the sense it's used in. While useful for many applications, this static nature limits their ability to model how word meanings shift and adapt across different contexts.

Contextual embeddings, introduced with transformer architecture, address this limitation by generating context-dependent representations. These embeddings are better suited since their meaning is dependent on its surrounding data. During the training process, both embeddings and model parameters collectively form the set of parameters of the model.

The embedding layer is represented as a matrix $\mathbf{E} \in \mathbb{R}^{|V| \times d}$, where $|V|$ is the vocabulary size and d is the embedding dimensionality. Each row of $\mathbf{E}$ corresponds to a vector representation of a token in the vocabulary. Given a token x_i with index i in the vocabulary, its embedding is computed as:

$$\text{Embedding}(x_i) = \mathbf{E}[x_i] \tag{2.14}$$

2.3.4 Positional Encodings

Transformers process input sequences using parallel computation and self-attention, mechanisms that do not inherently encode the position of tokens within the sequence. Positional encodings are crucial components that inject information about the relative or absolute positions of tokens in the sequence. These encodings enable the model to understand and utilize sequential order in its computations.

The original transformer paper introduced sinusoidal positional encodings [124], which use sine and cosine functions of different frequencies to represent positions. For position pos and dimension i in a d-dimensional embedding space, the encodings are computed as:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \end{aligned} \tag{2.15}$$

where pos is the position in the sequence, and i is the dimension index. This formulation handles sequences of arbitrary length.

An alternative approach is learned positional embeddings [24], where the position vectors are treated as model parameters and learned during training. It is simpler to implement; however, learned embeddings are limited to a maximum sequence length defined during training and do not inherently generalize to longer sequences.

Another approach called Attention with Linear Biases (ALiBi) [99] offers a position encoding method that allows models to extrapolate to sequence lengths not seen during training by adding a position-dependent scalar bias to attention scores. The key strength of ALiBi lies in its ability to allow models trained on shorter sequences to generalize to much longer sequences because the bias term is independent of sequence length.

The positional encoding is added to or combined with the token embeddings before being processed by the transformer layers:

$$\text{Input} = \text{Embedding}(x_i) + \text{PositionalEncoding}(pos_i) \tag{2.16}$$

More recently, Rotary Positional Embedding (RoPE) [119] has gained prominence by offering a fundamentally different approach. Instead of adding positional vectors to embeddings or biasing attention scores, RoPE encodes positional information by rotating the query ($\mathbf{Q}$) and key ($\mathbf{K}$) vectors based on their absolute positions within the attention mechanism itself. It takes advantage of geometric properties, applying position-dependent rotation matrices to pairs of dimensions in the $\mathbf{Q}$ and $\mathbf{K}$ vectors before computing the dot product.

The core innovation of RoPE lies in how this rotation affects the query-key interaction. The rotation is designed such that the dot product between the rotated query vector at position m and the rotated key vector at position n depends only on the original query and key vectors and their relative position $m - n$. RoPE's advantages include inherently encoding relative positions and maintaining sequence order via geometric rotations within the attention mechanism. This yields better performance and generalization to longer sequences than additive methods.

2.3.5 Tokenization

End-to-end systems process input data as raw inputs, meaning speech signals are directly captured from a microphone or text strings for natural language processing. The input

sentence is split into discrete parts, called tokens. This step enables transformer models to handle the input data by converting it into a sequence of manageable, discrete units. This preprocessing step directly impacts model performance and efficiency [111].

World-level tokenization is the simplest approach. It splits text on whitespace and punctuation. It is intuitive but struggles with out-of-vocabulary words and requires a large vocabulary to cover most words in a language.

Character-level tokenization represents text as individual characters, which solves out-of-vocabulary problems. Using characters as tokens also means that vocabulary will be minimal. However, this approach generates longer sequences and fails to capture meaningful semantic units.

Byte-Pair Encoding (BPE) [115] is the dominant tokenization method in modern transformer architectures. BPE makes a balance between word and character-level approaches by learning subword units from the training data. The algorithm iteratively merges the most frequent pair of consecutive bytes or characters to create a new token, thus creating a vocabulary of optimal subword units. The training process is formally described at 1.

Algorithm 1 Byte-Pair Encoding (BPE) algorithm

Require: Training corpus C, target vocabulary size V
Initialize:
  Vocab $\leftarrow$ set of individual characters in C
  Split each word in C into character sequences
while $|\text{Vocab}| < V$ **do**
  $(x, y) \leftarrow \arg\max_{(a,b)\in\text{pairs}(C)} \text{count}((a, b))$
  Add merged token xy to Vocab
  Replace all occurrences of (x, y) with xy in C
end while
return Vocab

Other variants include WordPiece [128] and Unigram [65]. These approaches rely on probabilistic approaches to token merging. Similar to BPE, SentencePiece [66] processes text as unicode sequences and applies this algorithm language-agnostically. This ensures consistent tokenization across different scripts. Unigram is used in conjunction with SentencePiece.

Tokenizers employ several special tokens that provide context and structural guidance to the model. These special tokens serve specific roles during training and model inference. They mark the beginning and end of sequences, indicate padding, separate different input components, or act as special instructions in order for the model to understand the structure of the input data. Commonly, these special tokens are represented using formats like `[PAD]`, `<EOS>`, and `[MASK]`, respectively representing sequence padding, end-of-sequence marking, and enabling masked language modeling during pre-training.

The tokenization method and vocabulary size are fundamental design decisions in transformer architectures. Vocabulary size impacts the model's computational performance, where larger vocabularies reduce sequence lengths but increase the parameter count in the following embedding layers. Meanwhile, smaller vocabularies result in longer sequences with fewer parameters.

2.3.6 Encoder

Encoder is the first part of the transformer that processes the input sequence and produces a continuous representation or embeddings of the input sequence. Visual representation of an encoder part of a transformer model is shown in figure 2.2. The encoder captures the context of each token with respect to the entire sequence.

Transformer models often employ more than one encoder layer. In the original architecture, one encoder layer comprises of two sub-modules: a multi-head self-attention mechanism and a fully connected feed-forward network. Additionally, it incorporates residual connections around each sub-module, which are then followed by a normalization layer. In the original paper [124], the normalization layer was applied after the attention and feed-forward layers. Current transformer architectures use a normalization layer before attention and a feed-forward layer. This small change has been shown to make the training of transformer models more stable [129]. For the normalization layer, LayerNorm was used, but in recent transformer models, it is more common to use Root Mean Square Layer Normalization (RMSNorm) instead. It is more computationally efficient and often shows comparable or slightly better performance [141].

Within each encoder layer, multi-head self-attention is utilized. This allows the encoder to process the input sequence by relating each token to every other token, effectively capturing the contextual information from the entire sequence in parallel across different representation subspaces ('heads').

2.3.7 Decoder

Decoder is the second part of the transformer. It is responsible for generating the output sequence based on the encoder's representations. As shown in figure 2.2, the decoder processes the encoder's output and generates tokens sequentially.

Each decoder layer typically comprises three main sub-layers, along with residual connections and layer normalization, similar in structure to the encoder layers but with distinct functions. The first sub-layer is a masked multi-head self-attention mechanism. This masking ensures the autoregressive property, preventing the model from seeing future tokens during generation. The second sub-layer is the multi-head cross-attention mechanism. This is where the decoder interacts with the encoder's output: the query vectors ($\mathbf{Q}$) originate from the decoder's state, while the key ($\mathbf{K}$) and value ($\mathbf{V}$) vectors are taken directly from the final representation sequence produced by the encoder. This allows the decoder, at each step, to focus on the most relevant parts of the input sequence's representation when predicting the next output token. The third sub-layer is a standard position-wise feed-forward network. The decoder operates autoregressively, beginning with a start token and generating tokens sequentially, using previously generated outputs and the context derived from the encoder via cross-attention, until an end token is produced.

One layer comprises of three sub-modules: masked attention 2.11, cross attention, and feed-forward network. Cross attention mixes data from the encoder and decoder part; queries originate from the results of the preceding decoder sub-module, while keys and values stem from the outputs of the encoder. This aligns the encoder's input with the decoder's input.

Finally, within the decoder architecture specifically configured for ASR, a linear layer followed by a softmax function acts as the output layer. This linear layer projects the decoder's final hidden state representation into a vector whose size matches the target vocabulary. It produces raw logit scores for each potential token. The subsequent softmax

layer then transforms these logits into a probability distribution over the entire vocabulary, indicating the likelihood of each token being the next element in the generated transcription. During inference, the model selects the next token based on these probabilities to build the output text sequence word by word (or token by token). The chosen token is then fed back together with the previous input as a new input for the decoder. Decoding process continues until the model produces a token that signals end of generation for the model. Because of this auto-regressive process, it is sequential and time-consuming compared to the encoder.

Chapter 3

End-to-End Architectures for Automatic Speech Recognition

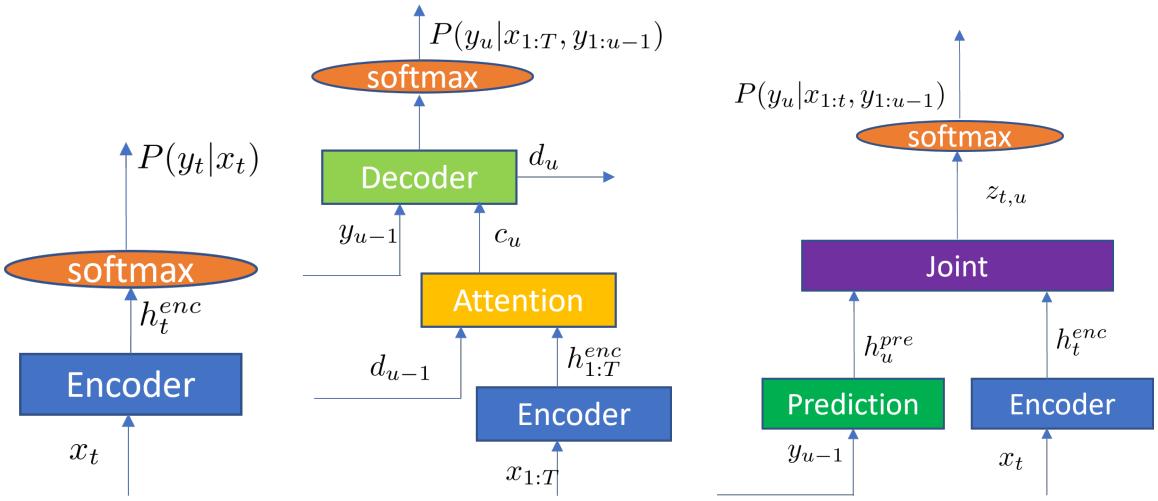
Building upon the foundational concepts of the transformer architecture discussed in Chapter 2, this chapter delves into specific end-to-end architectures that have revolutionized the field of Automatic Speech Recognition (ASR). These architectures aim to directly map an input audio signal, either in its raw form or as processed features, to an output sequence of text.

3.1 Introduction to End-to-End ASR

Traditional ASR systems often involved a complex pipeline of separate modules for acoustic feature extraction, acoustic modeling, and language modeling. These modular systems were often difficult to optimize as a whole. In contrast, end-to-end ASR architectures simplify this process by learning a direct mapping from the input audio to the output text sequence using a single neural network [37]. This paradigm shift has been enabled by advances in deep learning, particularly sequence-to-sequence models capable of handling variable-length inputs and outputs. The goal of end-to-end ASR is to train a single model that can learn all the necessary components, from extracting relevant acoustic features to generating grammatically correct and semantically meaningful transcriptions.

The three most popular end-to-end architectures dominate ASR [70, 62]. The approaches are Connectionist Temporal Classification (CTC) [38, 39], Attention-based Encoder-Decoder (AED) models [17, 13], and the Recurrent Neural Network Transducer (RNN-T) [37]. These architectures differ fundamentally in how they align the audio input with the text output and generate the final sequence, as illustrated in Figure 3.1. Each architecture presents distinct advantages and challenges, which will be discussed in the following sections.

Regardless of the specific end-to-end architecture, the ultimate goal during inference is to decode the input audio into the most probable text sequence. This decoding process typically involves strategies such as greedy search, which rapidly selects the most likely output token at each step based on the model's probabilities. However, to achieve higher accuracy, beam search is more commonly employed. Beam search explores a limited set of the most promising candidate transcriptions simultaneously by keeping track of a „beam“ of hypotheses, offering a better approximation of the optimal sequence, although with increased computational demand.



(a) CTC Architecture (b) AED Architecture (c) RNN-T Architecture

Figure 3.1: Diagrams of the three most common End-to-End ASR architectures: (a) Connectionist Temporal Classification (CTC), (b) Attention-based Encoder-Decoder (AED), and (c) Recurrent Neural Network Transducer (RNN-T). These illustrate their core components and information flow. Adapted from [70].

3.2 Connectionist Temporal Classification (CTC)

Connectionist Temporal Classification (CTC) [38] is an end-to-end architecture primarily used for sequence labeling tasks, including ASR. It is characterized by its encoder-only structure, where the input audio is processed by an encoder to produce a sequence of frame-level predictions.

3.2.1 Architecture and Loss Function

The CTC architecture consists of an acoustic encoder that takes the input audio features and transforms them into a sequence of higher-level representations. A linear layer followed by a softmax activation is then applied to each time step of the encoder's output, producing a probability distribution over the target vocabulary plus an additional 'blank' token. This special token serves as a hard boundary between groups of characters and also makes it possible to filter out duplicate characters.

The CTC loss function is designed to train the model without requiring explicit frame-level alignments between the audio and the text. For a given audio input and its corresponding transcription, there can be multiple possible frame-level label sequences (paths) that are consistent with the transcription. A path is considered consistent if it can be reduced to the transcription by first removing consecutive repeated labels and then removing all blank tokens. The CTC loss is the negative logarithm of the sum of probabilities of all such consistent paths.

Mathematically, let $X = (x_1, x_2, ..., x_T)$ be the input audio sequence of length T, and $Y = (y_1, y_2, ..., y_U)$ be the target transcription of length U. The model outputs a probability distribution $p(a_t|X)$ over the vocabulary $\mathcal{V} \cup \{\langle blank \rangle\}$ at each time step t, where a_t is a label from the vocabulary or the blank token. The probability of a path $\pi = (\pi_1, \pi_2, ..., \pi_T)$,

where each $\pi_t \in \mathcal{V} \cup \{\langle blank \rangle\}$, is given by $p(\pi|X) = \prod_{t=1}^{T} p(\pi_t|X)$. The CTC loss for a single input-output pair (X, Y) is then:

$$\mathcal{L}_{CTC}(X, Y) = -\log \sum_{\pi \in \mathcal{B}(Y)} p(\pi|X) \tag{3.1}$$

where $\mathcal{B}(Y)$ is the set of all paths π that, after removing consecutive repeated labels and then all blank tokens, result in the transcription Y. The summation over all possible paths is computed using a dynamic programming algorithm based on the forward-backward algorithm.

3.2.2 Characteristics

CTC is the first E2E architecture widely used in ASR [43, 83, 117, 21]. CTC is known for its non-autoregressive nature, as the prediction at each time step is independent of other time steps given the input. This allows for fast decoding. However, a key limitation of CTC is the assumption of conditional independence between the output labels, which restricts its ability to model complex language structures.

3.3 RNN Transducer (RNN-T)

The RNN Transducer (RNN-T) [37] is another end-to-end architecture designed to address some of the limitations of CTC, particularly the conditional independence assumption. RNN-T is an encoder-decoder architecture with a joiner network that learns to align the input audio with the output text in a monotonic fashion. It is important to note that when originally proposed, an RNN-Transducer uses a recurrent network as the encoder, hence the name RNN-T. Nowadays, Transducers usually adopt more sophisticated networks involving self-attention in their encoder [130].

3.3.1 Architecture and Mechanism

As illustrated in Figure 3.1c, the RNN-T architecture consists of three main components:

1. **Acoustic Encoder**: Similar to CTC and AED models, this network processes the input audio features $X = (x_1, \ldots, x_T)$ into a sequence of high-level acoustic representations $h^{\text{enc}} = (h_1^{\text{enc}}, \ldots, h_T^{\text{enc}})$, where $h_t^{\text{enc}} \in \mathbb{R}^{d_{\text{enc}}}$.

2. **Prediction Network**: This is an autoregressive network (e.g., RNN, LSTM, or Transformer Decoder) that models the history of the predicted output sequence. It takes the previously emitted non-blank symbol y_{u-1} (where y_0 is a special start-of-sequence token) and its previous hidden state h_{u-1}^{pre} to generate the current prediction network hidden state $h_u^{\text{pre}} \in \mathbb{R}^{d_{\text{pred}}}$.

$$h_u^{\text{pre}} = \text{PredictionNetwork}(y_{u-1}, h_{u-1}^{\text{pre}}) \tag{3.2}$$

3. **Joint Network**: This network, typically a feed-forward network, combines the acoustic information from the encoder at time step t and the contextual information from the prediction network at output step u. It projects both hidden states and combines them, often through addition followed by a non-linear activation function ψ to

produce a joint representation $z_{t,u} \in \mathbb{R}^{d_{\text{joint}}}$.

$$z_{t,u} = \psi(\mathbf{W}_Q h_t^{\text{enc}} + \mathbf{W}_V h_u^{\text{pre}} + \mathbf{b}_z) \tag{3.3}$$

where $\mathbf{W}_Q \in \mathbb{R}^{d_{\text{joint}} \times d_{\text{enc}}}$ and $\mathbf{W}_V \in \mathbb{R}^{d_{\text{joint}} \times d_{\text{pred}}}$ are learnable weight matrices, and $\mathbf{b}_z$ is a bias vector.

The joint representation $z_{t,u}$ is then passed through a final linear layer and a softmax function to compute the probability distribution over the target vocabulary $\mathcal{V}$ augmented with a special 'blank' token $\langle blank \rangle$, denoted as $\mathcal{V}' = \mathcal{V} \cup \{\langle blank \rangle\}$.

$$h_{t,u} = \mathbf{W}_y z_{t,u} + \mathbf{b}_y \tag{3.4}$$

$$P(k|t,u) = \text{Softmax}(h_{t,u})_k \tag{3.5}$$

where $P(k|t,u)$ is the probability of emitting symbol $k \in \mathcal{V}'$ given the acoustic context up to time t and the label context up to step u. $\mathbf{W}_y \in \mathbb{R}^{|\mathcal{V}'| \times d_{\text{joint}}}$ and $\mathbf{b}_y$ are the weights and bias of the output layer.

The core mechanism of RNN-T operates on a conceptual grid defined by the input time steps $t \in [1, T]$ and the output sequence steps $u \in [0, U]$, where U is the length of the target sequence $Y = (y_1, ..., y_U)$. At each grid point (t, u), the model uses $P(k|t,u)$ to decide the next step:

- If it emits a non-blank symbol $k = y_u \in \mathcal{V}$, it advances in the output sequence (moves from u to $u+1$) and feeds y_u to the Prediction Network. The time step t remains the same.
- If it emits the blank symbol $k = \langle blank \rangle$, it advances in the input time sequence (moves from t to $t+1$) while staying at the same output step u. The Prediction Network state is retained.

This process generates an alignment path $\mathbf{a}$ through the $T \times (U+1)$ grid, starting at $(t = 1, u = 0)$ and ending when $t = T$ and $u = U$. An alignment path consists of a sequence of symbols from $\mathcal{V}'$.

3.3.2 Loss Function

Similar to CTC, the RNN-T loss function marginalizes over all possible alignment paths $\mathbf{a}$ that correctly correspond to the target sequence Y after removing the blank symbols. Let $\mathcal{A}(X, Y)$ be the set of all valid alignment paths for the input X and target Y. The probability of a single path $\mathbf{a} = (a_1, a_2, ..., a_{|\mathbf{a}|})$ is the product of the conditional probabilities at each step along the path:

$$P(\mathbf{a}|X) = \prod_{i=1}^{|\mathbf{a}|} P(a_i|t_i, u_i) \tag{3.6}$$

where (t_i, u_i) is the grid position corresponding to the i-th step of the path $\mathbf{a}$.

The total probability of the target sequence Y given the input X is the sum of probabilities of all valid alignment paths:

$$P(Y|X) = \sum_{\mathbf{a} \in \mathcal{A}(X,Y)} P(\mathbf{a}|X) \tag{3.7}$$

The RNN-T loss is the negative log-likelihood of this probability:

$$\mathcal{L}_{RNNT}(X, Y) = -\log P(Y|X) \tag{3.8}$$

This summation is computed efficiently using a dynamic programming approach, specifically a forward-backward algorithm adapted for the transducer lattice [37]. Efficient implementations often employ techniques like loop skewing [5] or function merging [109] to optimize computation and memory usage during training.

3.3.3 Characteristics

A key advantage of RNN-T is its ability to operate in a streaming fashion, making it suitable for real-time ASR applications. By processing the audio input incrementally and generating outputs based on local acoustic and label context, it can produce transcriptions with low latency [45]. Compared to CTC, RNN-T explicitly models dependencies between output labels via the Prediction Network, often leading to better performance, especially on tasks requiring strong language modeling. However, the architecture is more complex than CTC, involving three interacting networks and a more intricate loss computation, which increases training complexity and computational cost.

3.4 Attention-Based Encoder-Decoder (AED)

Attention-Based Encoder-Decoder (AED) architectures have become the dominant approach for many sequence-to-sequence tasks, including ASR [140, 27, 26]. These models leverage the attention mechanism, as detailed in Chapter 2, to dynamically weigh the importance of different parts of the input sequence when generating the output sequence.

3.4.1 Architecture and Key Components

An AED architecture for ASR consists of three main components, as illustrated in Figure 3.1b:

1. **Acoustic Encoder**: Processes the input audio features and generates high-level representations.
2. **Attention Mechanism**: Computes alignment scores between the encoder representations and the decoder state that produces a context vector.
3. **Autoregressive Decoder**: Generates the output text sequence one token at a time, conditioned on the encoder representations (via the context vector) and previously generated tokens.

Encoder Role

The primary role of the acoustic encoder is to transform the input audio sequence $X = (x_1, \ldots, x_T)$ into a sequence of high-level hidden representations $h^{\text{enc}} = (h_1^{\text{enc}}, \ldots, h_{T'}^{\text{enc}})$. The inputs X are typically raw waveforms, although some models expect spectral features like log-Mel filterbanks. The output length T' might differ from T due to downsampling operations commonly used in the initial layers of the encoder.

In modern AED architectures for ASR, the encoder is usually built on the transformer family. Representative variants include Conformer [40], Wav2Vec2 [4], HuBERT [52], Branchformer [93], Emformer [116] or more efficient version of Conformer called Fast-Conformer [107].

Regardless of the specific architecture, the output sequence serves as the key ($\mathbf{K}$) and value ($\mathbf{V}$) inputs for cross-attention.

Cross-Attention

The key component connecting the encoder and decoder is the cross-attention mechanism. This allows the decoder, at each generation step, to query the encoder's output representations and focus on the most relevant acoustic information for predicting the next token, using the attention principles detailed in Section 2.3.1.

Decoder Role

The decoder generates the output text sequence $Y = (y_1, \ldots, y_U)$ one token at a time. The prediction of the token y_u depends on the previously generated tokens $y_{<u} = (y_1, \ldots, y_{u-1})$ and the context vector $\mathbf{c}_u$ derived from the encoder outputs. The process starts with a special beginning-of-sequence ($\langle bos \rangle$) token y_0.

The decoder's final output is then used to compute the probability distribution over the target vocabulary $\mathcal{V}$ (including an end-of-sequence token $\langle eos \rangle$):

$$P(y_u|y_{<u}, X) = \text{Softmax}(\mathbf{W}_{\text{out}} s_u + \mathbf{b}_{\text{out}}) \tag{3.9}$$

where $\mathbf{W}_{\text{out}}$ and $\mathbf{b}_{\text{out}}$ are parameters of the final output layer.

The overall probability of the target sequence Y given the input X is:

$$P(Y|X) = \prod_{u=1}^{U} P(y_u|y_{<u}, X) \tag{3.10}$$

The model is trained by minimizing the negative log-likelihood (cross-entropy loss) of the correct target sequence:

$$\mathcal{L}_{AED}(X, Y) = -\log P(Y|X) \tag{3.11}$$

3.4.2 Characteristics

AED architectures excel at ASR thanks to the attention mechanism and the use of transformer based encoders and decoders. They often achieve high accuracy in ASR tasks due to their strong ability to model both acoustic and linguistic information. However, the autoregressive nature of the decoder can make decoding slower compared to CTC and RNN-T.

Current state-of-the-art performance on benchmarks like the Open ASR Leaderboard [118] by HuggingFace is primarily driven by models based on either the transformer (typically within AED architectures) or the Transducer framework. While transformer-based AED models often achieve high accuracy, Transducer models demonstrate better efficiency for real-time scenarios, achieving a higher throughput or Real-Time Factor X (RTFx), calculated as the duration of audio processed divided by the computation time.

Chapter 4

Pre-trained Models and Parameter-Efficient Fine-tuning in ASR

Building upon the end-to-end ASR architectures in Chapter 3, this chapter explores different pre-trained language and acoustic models and discusses how they can be used for ASR. While pre-trained models offer significant advantages, they require adaptation to perform optimally on specific downstream tasks. Fine-tuning is the process of further training these models on task-specific data. However, full fine-tuning of large-scale models can be computationally expensive and memory intensive. This has motivated the development of parameter-efficient fine-tuning (PEFT) techniques, which aim to achieve comparable performance while training only a small fraction of the model's parameters. These PEFT techniques will be discussed at the end of the chapter.

4.1 Pre-trained Language Models

In the context of ASR, language models play a crucial role when they are integrated as the decoder in a sequence-to-sequence architecture. Language models estimate the probability of token sequences and capture linguistic dependencies. Pre-trained on vast text corpora, these language models enhance transcription quality by effectively modeling syntax and semantics. These models are predominantly trained using self-supervised learning. In the last three years, there has been a significant increase in the popularity of language models. These models can be broadly categorized based on their architecture into three categories [92].

4.1.1 Encoder-Only Language Models

Encoder-only models, such as BERT [23] and RoBERTa [77], are based on the transformer's [124] encoder part. These models process input sequences bidirectionally, which allows each token to be attended to both its left and right contexts through self-attention. This bidirectional nature makes them particularly effective for tasks such as text classification, named entity recognition, and semantic similarity, where understanding of the whole context is necessary.

The primary pre-training objective for encoder-only models is Masked Language Modeling (MLM). In MLM, a certain percentage of tokens in the input sequence are randomly masked, and the model is trained to predict these masked tokens based on the surrounding context. For example, BERT masks 15% of tokens during pre-training, replacing them with a special `[MASK]` token with 80% probability, a random token with 10% probability, or leaving them unchanged with 10% probability, to avoid the dataset shift problem [23]. RoBERTa, which is an improved variant of BERT, enhances this by using dynamic masking (where tokens are masked differently at each epoch) and removing BERT's next-sentence prediction task, focusing solely on MLM. It also trains on a larger dataset (160GB of text compared to 13GB used for BERT) with larger mini-batch sizes and learning rates that improves performance on benchmarks like GLUE and SQuAD [77]. A recent advancement, ModernBERT [126], builds on encoder-only architectures by incorporating techniques pioneered in decoder-only models, achieving state-of-the-art performance across multiple benchmarks.

4.1.2 Decoder-Only Language Models

Decoder-only language models are built on the transformer's [124] decoder part and are primarily designed for generative tasks. They utilize a unidirectional (causal or masked) attention mechanism, where each token can only attend to the preceding tokens in the sequence. This architecture naturally lends itself to sequentially generating text, one token at a time.

The standard pre-training objective for decoder-only models is Causal Language Modeling (CLM), also known as next-token prediction. The model is trained to predict the next token in a sequence given all the preceding tokens. This objective enables the model to learn the relationships between words and generate fluent and coherent text, possibly even resembling this paragraph.

The scale required for this pre-training, especially for state-of-the-art models, is immense. For instance, training large-scale decoder-only language models like Meta's Llama 3.1 405B involves substantial computational resources. This model was trained on over 15 trillion tokens sourced from publicly available data. The training process utilized a cluster of 16,384 NVIDIA H100 80GB GPUs, accumulating to approximately 31 million GPU hours [36].

Decoder-only models have demonstrated strong generative capabilities, making them suitable for tasks like text generation, chatbots, and even as the language model component in ASR decoders. Prominent examples include the GPT series of models (GPT [104], GPT-2 [105], GPT-3 [12], GPT-4 [87], GPT-4o [85], OpenAI o1 [86]), as well as open-source alternatives like the Llama series with the new Llama 4[1] or Mistral [59].

4.1.3 Encoder-Decoder Language Models

Encoder-decoder models, such as BART [69] and T5 [106], combine a bidirectional encoder with an autoregressive decoder. The encoder processes the input sequence bidirectionally, capturing contextual information, while the decoder generates the output sequence autoregressively, using cross-attention to attend to the encoder's output. This architecture is particularly effective for sequence-to-sequence tasks, such as machine translation, text

[1] https://ai.meta.com/blog/llama-4-multimodal-intelligence/

summarization, and conditional text generation, where both understanding and generation are required.

Encoder-decoder models are typically pre-trained on denoising sequence-to-sequence tasks. For example, BART is trained by corrupting the input text with various noising functions and learning to reconstruct the original text. T5, on the other hand, frames all tasks as text-to-text problems, pre-training on a mixture of supervised and unsupervised tasks with a unified objective. The specific corruption methods in pre-training are:

- **Token Masking**: Randomly masking tokens in the input sequence, similar to MLM.
- **Token Deletion**: Randomly deleting tokens from the input sequence.
- **Text Infilling**: Masking contiguous spans of tokens and training the model to fill in the missing spans.
- **Sentence Permutation**: Shuffling the order of sentences in a document.
- **Document Rotation**: Choosing a random token as the start of the document.

By learning to invert these varied forms of textual corruption, the models develop effective representations that capture complex linguistic phenomena that provide a foundation for strong performance on sequence-to-sequence tasks after fine-tuning.

The following Figure 4.1 illustrates the core architectural differences between these three categories of language models.

4.1.4 Bidirectional and Auto-Regressive Transformer (BART)

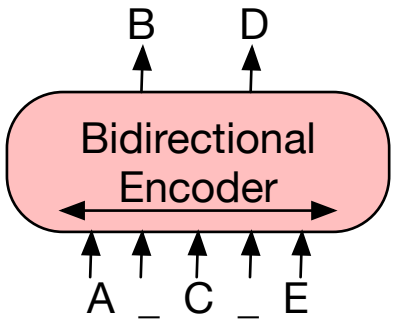
BART (Bidirectional and Auto-Regressive Transformers) [69] is an encoder-decoder language model designed for sequence-to-sequence tasks. It conceptualizes pre-training as a denoising autoencoder task. BART is trained using text corruption and reconstruction, as detailed above in 4.1.3.

Architecturally, BART utilizes a standard transformer encoder-decoder setup [124]. It employs a bidirectional encoder to process the potentially corrupted input sequence. This is coupled with an autoregressive decoder (similar to GPT) which generates the output sequence.

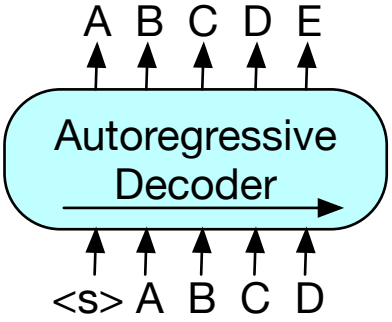
An innovation in BART's pre-training is the flexibility of the noising schemes applied to the input text. The authors experimented with several corruption strategies, finding the best performance often came from combining multiple approaches. By learning to reverse these diverse corruptions, BART develops robust representations suitable for a wide range of tasks. While particularly effective for generative tasks like abstractive summarization, dialogue, and machine translation due to its autoregressive decoder, BART also demonstrates strong performance on comprehension tasks like SQuAD and GLUE [69]. In the context of ASR, the decoder component of BART can be leveraged for its strong language modeling capabilities but also because its architecture inherently includes the cross-attention mechanism necessary for conditioning on acoustic encoder representations.

4.1.5 Generative Pre-trained Transformer (GPT)

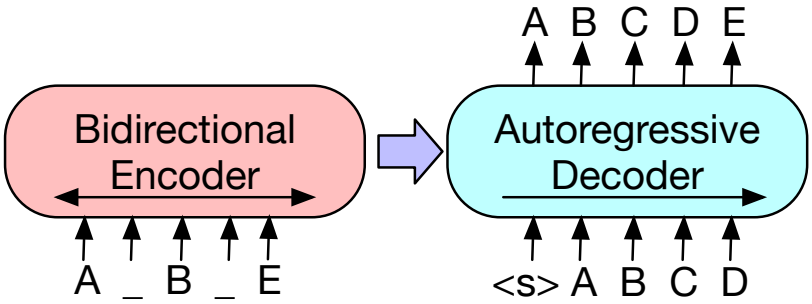
The Generative Pre-trained Transformer (GPT) series of models represents a family of decoder-only language models. These models are built upon the decoder part of the standard transformer architecture [124].



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.



(b) GPT: Tokens are predicted autoregressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.



(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitrary noise transformations. Here, a document has been corrupted by replacing spans of text with mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

Figure 4.1: A schematic comparison of BART [69] with BERT ([23]) and GPT ([104]). Cited from [69].

The fundamental pre-training objective for GPT models is Causal Language Modeling (CLM), or next-token prediction. Given a sequence of tokens $s_1, \ldots, s_{n-1}$, the model is trained to maximize the likelihood of the next token s_n:

$$L_{\text{CLM}} = \sum_i \log P(s_i | s_1, \ldots, s_{i-1}; \Theta) \tag{4.1}$$

where Θ denotes the trainable parameters of the language model.

The GPT series has demonstrated a clear trend: increasing model size, dataset size, and training compute leads to improved performance. The original GPT [104] introduced the effectiveness of the generative pre-training approach followed by task-specific fine-tuning for various NLP benchmarks. Subsequently, GPT-2 [105] scaled the model size significantly (up to 1.5 billion parameters) and was trained on a large, curated web text dataset (WebText); it demonstrated remarkable zero-shot learning capabilities, performing various tasks without any explicit fine-tuning, simply by conditioning the model with appropriate prompts.

Further scaling occurred with GPT-3 [12], which reached up to 175 billion parameters and was trained on even more extensive data. Its key contribution was demonstrating powerful few-shot (or „in-context“) learning; by providing just a few examples of a task within the prompt at inference time, GPT-3 could often match or exceed the performance of state-of-the-art fine-tuned models on various benchmarks without requiring any gradient updates or parameter changes. Subsequent models, such as GPT-4 [87] and GPT-4o [85], have continued this scaling trend, incorporating multimodal capabilities and enhanced reasoning abilities; however, these models remain closed-source.

4.2 Pre-trained Acoustic Models

Similar to the advances seen in natural language processing, pre-trained acoustic models have become essential to speech processing. Leveraging the idea of self-supervised learning, these models are initially trained to extract meaningful, high-level representations from large quantities of unlabeled audio data. Subsequently, the pre-trained model is fine-tuned on a smaller, labeled dataset for a specific downstream task, such as automatic speech recognition (ASR), speaker identification, emotion recognition, or language identification. This two-stage approach is advantageous when labeled data is scarce, which is common for many of the world's languages and specialized domains. This section will focus on the influential Wav2Vec 2.0 architecture and more recent models.

4.2.1 Wav2Vec 2.0

Wav2Vec 2.0 represents a milestone in self-supervised learning for speech representations. It demonstrated that learning powerful representations directly from raw audio, followed by fine-tuning, outperforms previous state-of-the-art methods. The idea is to learn speech representations by solving a contrastive task in the latent space, analogous to masked language modeling in NLP.

Architecture

The Wav2Vec 2.0 model architecture as illustrated in Figure 4.2, consists of three main components:

1. **Feature Encoder**: This module consists of a multi-layer 1D convolutional neural network (CNN) $f : \mathcal{X} \mapsto \mathcal{Z}$. It takes the raw audio waveform $\mathcal{X}$ as input and produces a sequence of latent speech representations $\mathbf{Z} = (z_1, \ldots, z_T)$ at a lower temporal frequency (every 20ms). The encoder captures local acoustic features. In the BASE (smaller) configuration, this part has seven blocks with temporal convolutions, resulting in a receptive field of 25 ms.

2. **Context Network (Transformer)**: The sequence of latent representations $\mathbf{Z}$ is then fed into a standard transformer network $g : \mathcal{Z} \mapsto \mathcal{C}$. This network applies multi-head self-attention to capture contextual dependencies over the entire sequence, which then produces context-aware representations $\mathbf{C} = (c_1, \ldots, c_T)$. Unlike standard transformers, Wav2Vec 2.0 replaces fixed positional embeddings with a convolutional layer to learn relative positional information. The BASE model uses 12 transformer blocks, while the larger variant (LARGE) uses 24 blocks.

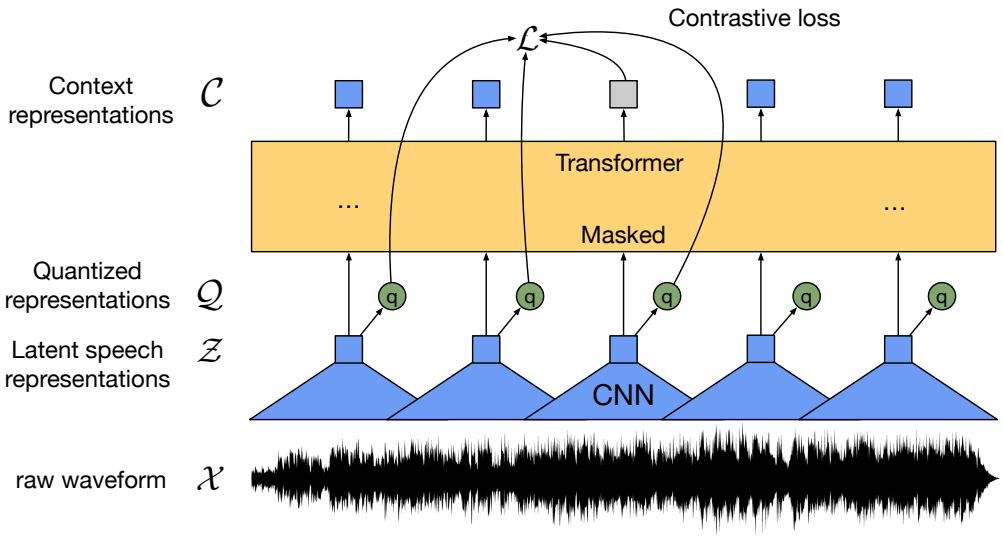


Figure 4.2: The Wav2Vec 2.0 framework architecture, showing the feature encoder (CNN), transformer network, and quantization module used during pre-training. Adapted from [4].

3. **Quantization Module**: This module discretizes the output of the feature encoder $\mathbf{Z}$ into a finite set of codebook entries $\mathbf{Q} = (q_1, \ldots, q_T)$. This is achieved using product quantization, where each latent vector z_t is mapped to G codebooks, each containing V entries. The selected entries are concatenated and linearly transformed to form q_t. Crucially, this module is only used during the self-supervised pre-training phase to generate targets for the contrastive task. Wav2Vec 2.0 employs Gumbel-Softmax [58] to allow differentiable selection of codebook entries during training. The probability of choosing the v-th codebook entry for group g is given by:

$$p_{g,v} = \frac{\exp(l_{g,v} + n_v)/\tau}{\sum_{k=1}^{V} \exp(l_{g,k} + n_k)/\tau} \tag{4.2}$$

where l are the logits computed from z_t, $n = -\log(-\log(u))$ with $u \sim U(0, 1)$ are Gumbel noise samples, and τ is a non-negative temperature parameter that is annealed during training.

Self-Supervised Pre-training

Wav2Vec 2.0 is pre-trained on large amounts of unlabeled speech audio using a contrastive task combined with a diversity loss.

1. **Masking**: Similarly to BERT, a portion of the latent feature encoder outputs $\mathbf{Z}$ is masked before being fed into the transformer context network. Specifically, a proportion p ($p = 0.065$) of time steps is randomly selected as starting indices. Each selected index initiates a mask over the subsequent M time steps ($M = 10$). These masked latent vectors z_t are replaced with a single learned feature vector representation before entering the transformer. This masking strategy results in approximately 49% of the total time steps being masked [4].

2. **Contrastive Loss**: For each masked time step t, the model uses the corresponding contextualized representation c_t from the transformer output to predict the correct

quantized representation q_t (derived from the unmasked z_t via the quantization module). The prediction is framed as a contrastive task: identifying the true target q_t from a set of K distractor quantized representations $\tilde{q}$. These distractors are uniformly sampled from other masked time steps within the same audio utterance. The contrastive loss $\mathcal{L}_m$ is defined as:

$$\mathcal{L}_m = -\log \frac{\exp(\text{sim}(c_t, q_t)/\kappa)}{\sum_{\tilde{q}\sim Q_t} \exp(\text{sim}(c_t, \tilde{q})/\kappa)} \tag{4.3}$$

where $\text{sim}(a, b) = \mathbf{a}^\top \mathbf{b}/(\|\mathbf{a}\|\|\mathbf{b}\|)$ is the cosine similarity, Q_t is the set containing q_t and the K distractors ($K = 100$), and κ is a temperature parameter [4].

3. **Diversity Loss**: To encourage the model to utilize the codebook entries effectively, a diversity loss $\mathcal{L}_d$ is added. This loss maximizes the entropy of the averaged softmax distribution of codebook usage across batches for each codebook group G. Let $\bar{p}_{g,v}$ be the average probability of using the v-th entry in the g-th codebook. The diversity loss is:

$$\mathcal{L}_d = \frac{1}{GV} \sum_{g=1}^{G} -H(\bar{p}_g) = \frac{1}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} \bar{p}_{g,v} \log \bar{p}_{g,v} \tag{4.4}$$

This loss term encourages the model to explore and utilize the full capacity of the quantization codebooks.

The overall pre-training objective is a weighted sum of the contrastive loss and the diversity loss:

$$\mathcal{L} = \mathcal{L}_m + \alpha \mathcal{L}_d \tag{4.5}$$

where α is a hyperparameter controlling the weight of the diversity loss ($\alpha = 0.1$) [4].

Fine-tuning for ASR

After pre-training on unlabeled data, the Wav2Vec 2.0 model is fine-tuned for ASR using labeled data. A randomly initialized linear layer is added on top of the transformer output $\mathbf{C}$ to predict the vocabulary tokens (characters or sub-words). The model is then trained using the CTC 3.2 loss. During fine-tuning, the parameters of the transformer and the new linear layer are updated. The quantization module is not used during fine-tuning.

The success of Wav2Vec 2.0 comes from its ability to learn representations directly from raw audio waveforms in a self-supervised manner, which reduces the need for large amounts of transcribed data.

4.2.2 Whisper

Representing a different paradigm and architecture compared to self-supervised models such as Wav2Vec2, Whisper [103], developed by OpenAI, leverages large-scale weakly supervised pretraining for multilingual and multitask speech recognition. Instead of learning from unlabeled audio, Whisper v1 is trained on a dataset of 680,000 hours of audio paired with noisy transcripts collected from the internet. The core goal of Whisper was to build a robust model capable of strong zero-shot generalization across different datasets, tasks, and languages without the need for task-specific fine-tuning.

Architecture

Whisper uses a transformer encoder-decoder architecture. Input is a raw audio waveform, which is resampled to 16 kHz. An 80-channel log-magnitude Mel spectrogram is computed using 25-millisecond windows with a 10-millisecond stride, serves as the input to the encoder. The encoder architecture features a small convolutional *stem* that consists of two 1D convolutional layers with GELU activation [47] where the second layer has a stride of two, to initially process the spectrogram. Sinusoidal positional embeddings are added to the output of this *stem* before the features are passed into the main transformer blocks, which utilize pre-activation residual connections. The decoder uses learned positional embeddings and employs tied input-output token representations [100]. For tokenization, they utilize a byte-level Byte-Pair Encoding tokenizer, identical in principle to the one used for GPT-2 but specifically refitted for multilingual data. OpenAI has released this architecture in several sizes, ranging from a `tiny` model with 39 million parameters up to a `large` version containing 1.55 billion parameters. The architecture is illustrated in Figure 4.3.

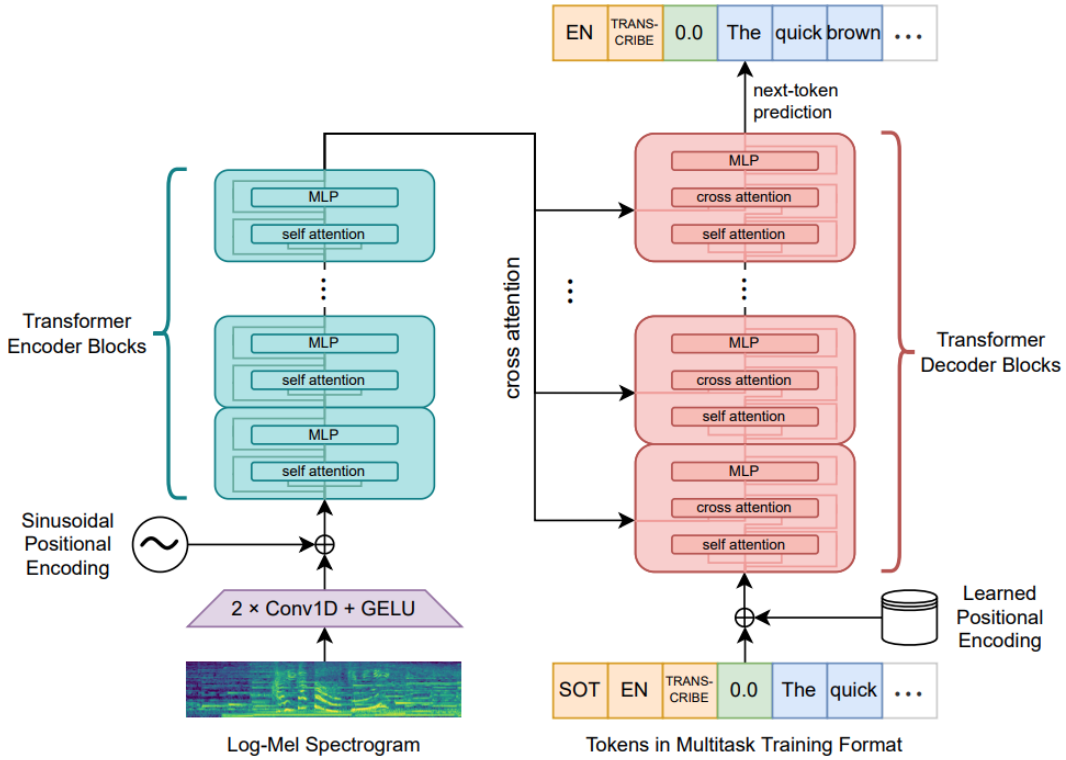


Figure 4.3: Overview of the Whisper architecture. Figure adapted from [103].

Recent Developments and Relevance to Encoder-Decoder ASR

Since its initial release, Whisper has seen several iterations and inspired community-driven improvements. Whisper v2 incorporated longer training with regularization techniques (SpecAugment [91], Stochastic Depth [55] and BPE Dropout [101]. Subsequently, Whisper v3 was trained on a substantially larger and more diverse dataset, leveraging 1 million hours of weakly labeled audio and 4 million hours of pseudo-labeled audio. Along with

minor architectural refinements (128 Mel frequency bins), this extensive data foundation contributed to further WER reductions of 10-20% over v2, particularly by improving its multilingual capabilities [88].

Fine-tuned variants have also emerged. CrisperWhisper [147] focuses on verbatim transcription by employing an adjusted tokenizer and a custom attention loss during fine-tuning to achieve accurate word-level timestamps, differentiate spoken fillers („um," „uh"), and transcribe speech exactly as uttered. CrisperWhisper shows stronger performance on the ASR leaderboard [118] than Whisper v3.

Whisper serves as a prominent example of an encoder-decoder architecture for ASR that integrates acoustic encoding with autoregressive textual generation. This architectural choice differs from encoder-only approaches. An understanding of the capabilities and structure of models like Whisper is relevant, as this thesis investigates techniques relevant to the effective training and adaptation of encoder-decoder models for speech recognition tasks.

4.2.3 Canary

The model that achieves the highest scores across many datasets is called Canary [102] and was released by NVIDIA. They challenge the reliance on web-scale datasets for training high-performance ASR models. This encoder-decoder model exceeds Whisper-large-v3 on benchmarks for English, French, Spanish, and German, while using considerably less training data (around 86,000 hours versus Whisper's initial 680,000+ hours). Architecturally, Canary utilizes a FastConformer encoder paired with a standard transformer decoder. Key training differences contributing to its performance with less data include the use of pseudo-labeled translation data for AST, data balancing and dynamic bucketing techniques (reducing computational overhead and speeding up training compared to fixed-length padding used in Whisper training), initializing the encoder from a pre-trained ASR checkpoint, and incorporating noise-robust fine-tuning which reduces model hallucinations. Canary achieves strong accuracy with less data and faster training, completing in under 48 hours on 128 A100 GPUs.

4.3 Parameter-Efficient Fine-tuning

Training large-scale models, such as the transformer-based architectures prevalent in both acoustic and language modeling for ASR, demands substantial computational resources and memory. Full fine-tuning, where all model parameters are updated on a downstream task, becomes increasingly impractical as model sizes grow into billions of parameters. Parameter-Efficient Fine-tuning (PEFT) techniques have emerged as a crucial solution to this challenge [25]. PEFT methods aim to adapt large pre-trained models to specific tasks or domains by training only a small fraction of the total parameters or only a small number of additional parameters. This significantly reduces the computational cost, memory requirements, and storage overhead associated with fine-tuning, while often achieving performance comparable to full fine-tuning [73]. They preserve the knowledge captured during pre-training and reduce the risk of catastrophic forgetting and overfitting, especially when the fine-tuning dataset is small [132].

In the context of ASR, state-of-the-art ASR relies on massive pre-trained acoustic models and potentially large language models. Adapting these models to specific languages, accents, domains (medical dictation, call center conversations), or even individual speakers

using full fine-tuning is, in many cases, impractical. PEFT allows for efficient specialization of these powerful pre-trained models.

The landscape of PEFT methods is diverse and rapidly evolving. These techniques can be broadly categorized based on how they modify the model during fine-tuning. PEFT methods can be classified into several major groups: additive, partial, reparameterized, hybrid, and unified fine-tuning [132].

4.3.1 Additive Fine-tuning

These methods introduce new, trainable parameters or modules into the architecture while keeping the original parameters frozen.

- **Adapter-based Fine-tuning**: This approach involves inserting small, task-specific neural network modules, known as adapters, into the layers of the pre-trained model. Typically, these are small feed-forward networks (often with a bottleneck structure: down-projection -> non-linearity -> up-projection) placed after the attention and feed-forward layers in each transformer block [51]. Variations include placing adapters in parallel [44], using residual connections [74], dynamically dropping adapters during inference [110], fusing knowledge from multiple adapters [95], or using hypernetworks to generate adapter parameters [46].
- **Soft Prompt-based Fine-tuning**: Instead of modifying internal model weights, these methods prepend trainable continuous vectors (soft prompts or prefixes) to the input embedding sequence or intermediate hidden states. Prompt Tuning [68] adds learnable prompt tokens directly to the input embeddings. Prefix Tuning [71] adds trainable prefix vectors to the hidden states within each layer's attention mechanism (affecting keys and values). Other methods explore variations like adversarial prompts or multi-task prompt transfer. In this variant, only the prompt/prefix vectors are trained.
- **Other Additive Methods**: This category includes techniques like Ladder Side-Tuning [120], which adds a separate side network or $(IA)^3$, that learns scaling vectors for activations [75].

4.3.2 Partial Fine-tuning

These methods fine-tune only a selected subset of the original pre-trained parameters, keeping the rest frozen.

- **Bias Update**: Techniques like BitFit [139] propose fine-tuning only the bias terms within the transformer layers and the classification head. This is extremely parameter-efficient but it has limited expressivity.
- **Pretrained Weight Masking**: Methods like Threshold-Mask [145] or FISH Mask [121] identify important pre-trained weights based on criteria like magnitude or Fisher information and create a binary mask. Only unmasked weights are active or used. Typically, the weights remain frozen, and the mask selects them.
- **Delta Weight Masking**: These methods focus on masking the updates (delta weights) applied to the pre-trained weights. LT-SFT [2] identifies parameters with

large changes after initial full fine-tuning and only tunes those in subsequent steps. Diff Pruning [41] learns a sparse mask for the delta weights directly. SAM [30] uses a second-order approximation to select delta weights.

4.3.3 Reparameterized Fine-tuning

This category, which includes LoRA and its derivatives, modifies the model by reparameterizing the weight update matrices, often using low-rank matrices.

- **Low-rank Decomposition**: This involves approximating the weight update matrix ΔW as a product of smaller matrices.
  - **LoRA (Low-Rank Adaptation)** [53]: This has become a very popular and effective technique. It hypothesizes that the change ΔW needed to adapt a pre-trained weight matrix $W \in \mathbb{R}^{d\times k}$ is low-rank. LoRA introduces two small, trainable matrices $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$, where the rank $r \ll \min(d, k)$, such that the update is $\Delta W = BA$. The effective weight matrix used during fine-tuning is $W' = W + \Delta W$, often with a scaling factor:
    $$W' = W + \frac{\alpha}{r}BA \tag{4.6}$$
    Only A and B are trained, while the original W is frozen. This drastically reduces the number of trainable parameters to $r(d + k)$. A significant advantage is that after training, ΔW can be merged with W ($W_{\text{merged}} = W + \frac{\alpha}{r}BA$), resulting in zero additional inference latency compared to the original model. LoRA is commonly applied to attention weight matrices (Query, Key, Value, Output) and sometimes Feed-Forward Network layers.
    However, LoRA introduces potential limitations. The low-rank constraint may restrict the expressiveness of the weight updates, potentially limiting adaptation to tasks requiring significant deviations from the pre-trained model. Additionally, the choice of hyperparameters r and α can affect stability and performance, with excessively large updates risking training divergence [53].
  - **KronA (Kronecker Adapter)** [28]: Uses Kronecker products instead of standard matrix multiplication for the decomposition $\Delta W = B \otimes A$.
- **LoRA Derivatives**: Building on the success and simplicity of LoRA, numerous derivative techniques have emerged to address its limitations or enhance its capabilities in specific scenarios. Key directions include improving efficiency through quantization and optimizing parameter allocation.
  - *Quantization-Aware Adaptation*: QLoRA [22] reduces the memory footprint required for fine-tuning. QLoRA achieves this by quantizing the pre-trained model weights to very low precision, typically 4-bit. While the base model is quantized, the LoRA adapter weights are kept in a higher precision format. During the forward pass, the quantized base weights are de-quantized on-the-fly to the compute precision, the LoRA update is computed and added, and then the computation proceeds. To further save memory, QLoRA employs techniques like double quantization and paged optimizer (offloading optimizer states to CPU RAM). Other related work, like QA-LoRA [134] and LOFTQ [72], explores integrating quantization more deeply into the LoRA process for potentially faster inference or better initialization.

- *Adaptive Rank Allocation (AdaLoRA)*: Standard LoRA uses a fixed rank r for all adapted matrices. However, different layers or matrices might benefit from different adaptation capacities. AdaLoRA [144] addresses this by dynamically allocating the parameter budget (the rank) during training based on the importance of weight matrices. It parameterizes the LoRA update using Singular Value Decomposition (SVD) and adaptively prunes less important singular values based on a sensitivity-based importance score. This allows the model to allocate more parameters (higher rank) to weight matrices that contribute more significantly to the task, while reducing the rank for less important ones.
- *Other Variations*: Further research has explored aspects like pruning within LoRA (LoRAPrune [143]), incorporating Bayesian principles (Laplace-LoRA [135]), or modifying the update mechanism itself, such as freezing one of the LoRA matrices to save activation memory (LoRA-FA [142]) or decomposing the update differently (DoRA [76]).

4.3.4 Weight-Decomposed Low-Rank Adaptation (DoRA)

DoRA [76] is a recent technique that refines LoRA by explicitly decomposing the pre-trained weight matrix W_0 into magnitude and direction components before applying the low-rank update. The idea is to separate the adaptation of how much a neuron fires (magnitude) from what pattern it responds to (direction). First, W_0 is decomposed:

$$W_0 = m \odot \frac{V}{||V||_c} \approx m \odot \frac{W_0}{||W_0||_c} \tag{4.7}$$

where m is a learnable vector representing the column-wise magnitude, V is the directional component (initialized as W_0), $||\cdot||_c$ is the column-wise norm, and $\odot$ denotes element-wise multiplication. Fine-tuning involves updating both m and V. Importantly, the low-rank update $\Delta V = BA$ (using LoRA matrices A and B with rank r) is applied only to the directional component V:

$$W' = m \odot \frac{V + \Delta V}{||V + \Delta V||_c} = m \odot \frac{W_0 + BA}{||W_0 + BA||_c} \tag{4.8}$$

Here, the trainable parameters are the magnitude vector m and the LoRA matrices B and A. By explicitly learning the magnitude and applying LoRA to the normalized direction, DoRA aims for more stable and effective adaptation compared to standard LoRA, potentially capturing changes that LoRA might struggle with. However, unlike LoRA, the DoRA update cannot be easily merged back into W_0 for inference due to the dependence on the learned magnitude m and the normalization, which may introduce slight inference overhead.

4.3.5 Hybrid Fine-tuning

These methods manually combine different PEFT techniques to leverage their complementary strengths. Examples include MAM Adapter [44] that combines parallel adapters and prefix-tuning) or UniPELT [80] that integrates sequential adapters, prefix-tuning, and LoRA with a gating mechanism.

4.3.6 Unified Fine-tuning

These approaches aim to create frameworks that can encompass or automatically select/-configure different PEFT strategies. AutoPEFT [146] uses Bayesian optimization to search

for optimal configurations of adapters and prefix-tuning across layers. S^3Delta-M [54] performs differentiable structure search over combinations of methods like LoRA, Compacter, BitFit, etc.

4.3.7 Summary of PEFT Approaches

The choice of PEFT technique involves trade-offs between parameter efficiency, downstream task performance, training stability and speed, memory usage, implementation complexity, and potential inference overhead. Additive methods like adapters and prompt/prefix tuning are conceptually simple but might add inference latency or have limited capacity. Partial methods like BitFit are extremely efficient but may underperform. Reparameterized methods, particularly LoRA and its advanced derivatives like QLoRA and DoRA, offer a strong balance by significantly reducing trainable parameters while often matching full fine-tuning performance and allowing zero-cost inference merging. Hybrid and unified methods attempt to get the best of multiple worlds but can increase complexity.

Chapter 5

Experimental Evaluations

This chapter details all the experiments in chronological order.

5.1 Model Selection

For the experiments, I take advantage of *transformers* library from huggingface and their class (SpeechEncoderDecoder [29]) to directly use any encoder and decoder part. I choose the following transformer architectures for each part:

5.1.1 Encoder

The encoder is based on the *Wav2Vec2* architecture described in Section 4.2.1, which showed that it can derive robust audio features from raw waveforms through self-supervised learning. Pre-training leveraged tens of thousands of hours of unlabeled speech, primarily sourced from LibriVox [60] dataset (providing over 53,000 hours), which allowed the model to develop robust representations effective across diverse real-world audio environments. The specific version chosen is *base*, which is the smallest version, containing 12 transformer blocks, 768 model dimensions, inner dimension of size 3072 and 8 attention heads. The total size of the encoder is 105M parameters.

5.1.2 Decoder

The decoder is based on *BART* architecture, as described in Section 4.1.4, which is an encoder-decoder model created for sequence-to-sequence generation tasks. The total size of the decoder is 96M parameters.

The selection of *BART* as the decoder, paired with the *Wav2Vec2* encoder, was made after considering alternative architectures, including decoder-only models (e.g., GPT-2 variants) and other encoder-decoder frameworks (e.g., T5). A primary constraint in this selection was the requirement for the decoder's parameter count to be comparable to that of the *Wav2Vec2* encoder. The employed *BART* model satisfies this constraint. Furthermore, this specific *Wav2Vec2-BART* integration directly aligns with the methodology used by von Platen et al. [96], which demonstrated the efficacy of leveraging these pre-trained components for ASR. This architectural choice capitalizes on the complementary strengths of *Wav2Vec2* for robust acoustic feature extraction and *BART*'s proficiency in generating coherent textual output. From *BART* architecture, only the decoder part is used. Convolutional adapter layers are added in the encoder to align the output with the decoder.

5.2 Dataset and Preprocessing

The performance, robustness, and generalization capabilities of the techniques explored in this thesis are reliant on the data used for training and evaluation. High-quality and sufficiently large datasets are key. Speech itself presents significant variability, influenced by factors that include language, regional accents, speaker demographics, speaking style, and recording conditions. This diversity demands various specialized datasets, each tailored to specific ASR tasks, languages, or acoustic environments.

The experiments in this thesis utilize two English language datasets: LibriSpeech and VoxPopuli. A summary of their key characteristics is presented in Table 5.1, followed by more detailed descriptions.

Dataset	**Language (s)**	**Duration (hours)**
LibriSpeech	English	1,000 (Total Labeled)
VoxPopuli	English (Used Subset)	1,800 (Labeled EN) / 400,000 (Total Unlabeled)

Table 5.1: Key characteristics of the primary ASR datasets utilized in this thesis.

5.2.1 Datasets Used in Experiments

LibriSpeech

LibriSpeech [90] is a widely adopted benchmark dataset for ASR research, particularly for evaluating models on clean, read English speech. It comprises recordings derived from audiobooks available through the LibriVox project. The audio is sampled at 16 kHz and is accompanied by high-quality transcriptions. Its large size and clean nature make it suitable for training and evaluating robust ASR models, although it primarily represents a single domain (audiobooks). For the experiments in this thesis, the ‘train-clean-100‘ subset (100 hours of the cleanest speech) is used for training, and the standard ‘dev-clean‘ and ‘test-clean‘ subsets are used for evaluation.

VoxPopuli

VoxPopuli [125] offers a large-scale multilingual speech corpus sourced from European Parliament proceedings recorded between 2009 and 2020. It is significantly larger and more diverse than LibriSpeech in terms of languages and potentially speaking styles. The full dataset contains approximately 400,000 hours of unlabeled audio across 23 European languages. For supervised tasks, it includes 1,800 hours of transcribed speech data along with aligned text translations in up to 15 languages. This makes VoxPopuli valuable for multilingual ASR and tasks like speech translation and cross-lingual transfer learning. The experiments in this thesis focus on the labeled English portion of the dataset for training and evaluation.

5.2.2 Data Preprocessing

The audio data from both LibriSpeech and VoxPopuli is resampled to $16\,kHz$ to meet the input requirements of the *Wav2Vec2* model. Text transcriptions are tokenized using the *BART* tokenizer. This tokenizer employs a vocabulary of 50,265 tokens, which includes special tokens for start-of-sequence (`<s>`, ID: 0), end-of-sequence (`</s>`, ID: 2), and padding (`<pad>`, ID: 1).

5.3 Evaluation Metric: Word Error Rate (WER)

The Word Error Rate (WER) is a widely used metric to evaluate the performance of automatic speech recognition systems by measuring the discrepancy between the predicted transcription and the reference text. It accounts for three types of errors: substitutions, where an incorrect word is predicted in place of the correct one; deletions, where a word present in the reference is omitted in the prediction; and insertions, where an extra word is added to the prediction that does not appear in the reference. The WER is calculated using the following formula:

$$\text{WER} = \frac{S + D + I}{N}$$

Where, S represents the number of substitutions, D the number of deletions, I the number of insertions, and N the total number of words in the reference text. A lower WER indicates better performance, as it reflects fewer errors relative to the reference transcription.

The figure below illustrates the components used to calculate the Word Error Rate by comparing Ground Truth text with Model output.

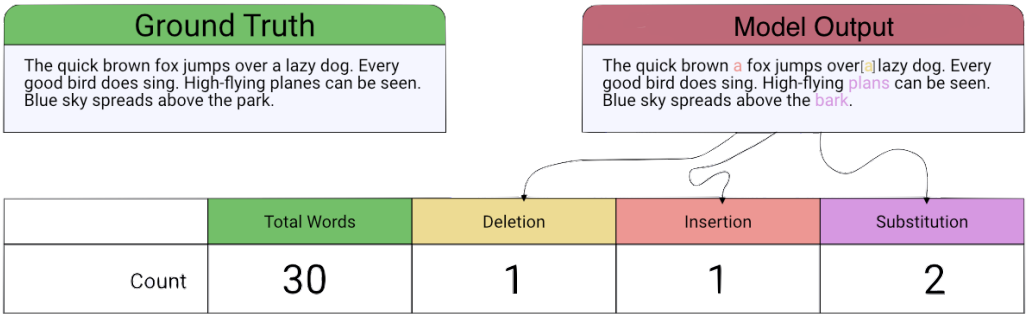


Figure 5.1: Comparison of Ground Truth and Model Output showing counts for Deletions (D), Insertions (I), and Substitutions (S). Figure adapted from Martinez [81].

Based on the counts shown in Figure 5.1: there were two substitutions, with 'planes' recognized as 'plans' and 'park' as 'bark'. There was one deletion, where the word 'a' before 'lazy' in the Ground Truth was missing in the Model output. Additionally, there was one insertion, as the word 'a' after 'brown' appeared in the Model output but not in the Ground Truth.

5.4 Experimental Setup 1

For the first experiment, I evaluate different initialization strategies for training a SpeechEncoderDecoder model for automatic speech recognition (ASR). The primary objective is to determine the impact of using pre-trained encoder and decoder components on model performance and training efficiency. Four different initialization strategies are evaluated:

1. **Both Pretrained**: Both the encoder and decoder are initialized with pre-trained weights (*Wav2Vec2-base* and *BART-base*, respectively). Adapter layers in the encoder and Final linear layer (`lm_head`) in the decoder are randomly initialized.

2. **Pretrained Encoder**: The encoder is initialized with pre-trained *Wav2Vec2-base* weights, while the decoder is randomly initialized.

3. **Pretrained Decoder**: The decoder is initialized with pre-trained *BART-base* weights, while the encoder is randomly initialized.

4. **From Scratch**: Both the encoder and decoder are initialized at random.

5.4.1 Training Details

The models are trained using the *Seq2SeqTrainer* from the Hugging Face *transformers* library. Training involves the use of the AdamW [78] optimizer, paired with a cosine annealing scheduler that includes a linear warmup phase. Different learning rates are applied depending on the dataset and model configuration. For the LibriSpeech dataset, higher learning rates are used when both the encoder and decoder are pretrained; this reflects a fine-tuning scenario where the goal is primarily to adapt the robust pretrained representations and train the smaller, randomly initialized layers. However, lower learning rates are applied for cases with a pre-trained encoder, pre-trained decoder, or training from scratch. This provides the stability required to learn meaningful representations from scratch or to integrate a randomly initialized component with a pre-trained one without disrupting the latter's learned knowledge. Similarly, for the VoxPopuli dataset, learning rates vary across configurations with pretrained components or training from scratch. Batch sizes were set specifically for each dataset: 64 for LibriSpeech and 96 for the more extensive VoxPopuli dataset. Gradient accumulation is performed with a single step. The primary evaluation metric for assessing model performance was the WER.

5.4.2 Results

The performance of each initialization strategy on both datasets is summarized in the following tables:

LibriSpeech (clean.100)

Initialization	WER [%] ↓	S	D	I	Eval Loss	Total Steps	Epochs
Both Pret.	**8.9**	7.2	0.9	0.8	0.57	**2230**	5
Pret. Encoder	10.2	7.7	1.7	0.8	0.81	66900	150
Pret. Decoder	106	81.9	10.9	13.5	5.74	67000	150
Scratch	115	85.4	6.6	23.4	5.29	22300	50

Table 5.2: Comparison of initialization strategies for the Wav2Vec2-BART ASR model trained on LibriSpeech (clean.100). Performance metrics include Word Error Rate (WER %), Substitution (S), Deletion (D), and Insertion (I) percentages, final evaluation loss, total training steps, and epochs completed.

VoxPopuli (English)

Where, **WER**: Word Error Rate, **S**: Substitutions, **D**: Deletions, **I**: Insertions.

Initialization	WER [%] ↓	S	D	I	Eval Loss	Total Steps	Epochs
Both Pret.	**10.6**	6.1	2.4	2.1	0.44	**10446**	6
Pret. Encoder	11.2	6.1	2.6	2.5	0.55	30990	17.8
Pret. Decoder	35.3	23.4	4.7	7.3	0.99	60935	35
Scratch	34.3	22.8	4.9	6.7	0.98	60935	35

Table 5.3: Comparison of initialization strategies for the Wav2Vec2-BART ASR model trained on VoxPopuli (English). Performance metrics include Word Error Rate (WER %), Substitution (S), Deletion (D), and Insertion (I) percentages, final evaluation loss, total training steps, and epochs completed.

Conclusion

The results clearly demonstrate the advantages of initializing the encoder-decoder model with pre-trained components. On both datasets, this strategy achieves the lowest WER and converges significantly faster than other strategies. This highlights the importance of leveraging prior knowledge encoded in pre-trained models for both audio feature extraction (encoder) and text generation (decoder).

Furthermore, the experiments reveal that pre-training the encoder has a more substantial impact on performance compared to pre-training the decoder. The *Pretrained Encoder* strategy consistently outperforms the *Pretrained Decoder* strategy, especially on the larger VoxPopuli dataset. This suggests that learning robust audio representations is crucial for ASR performance and that pre-trained Wav2Vec2 provides a strong foundation for this task.

A likely hypothesis for the poor performance when only the decoder is pre-trained is the mismatch between the expected input of the pre-trained decoder and the actual input it receives from the randomly initialized encoder. The pre-trained decoder (BART) is adept at language modeling but expects structured, meaningful hidden states as input via the cross-attention mechanism. However, a randomly initialized encoder produces noisy and unstructured representations of the audio signal, especially in the early stages of training. The cross-attention layers must then learn to map these poor-quality acoustic features to the decoder's latent space, while the encoder simultaneously tries to learn meaningful audio features. This dual challenge likely hinders effective learning. In contrast, when the encoder is pre-trained, it provides meaningful audio features from the start, giving the randomly initialized decoder a stable foundation to learn the mapping necessary for text generation.

Analysis of the error components (S, D, I) highlights a key challenge, substitution errors significantly outnumber both deletions and insertions. This implies that accurate word recognition within the audio signal poses a greater difficulty for these models than correctly segmenting the speech stream (which typically impacts deletion and insertion rates).

5.5 Experimental Setup 2

In the second experiment, I investigate the effect of freezing versus fine-tuning the Wav2Vec2 feature extractor. This part of the model is responsible for processing the raw input waveform through a series of seven convolutional layers to generate initial feature representations. These layers progressively reduce the temporal resolution while increasing the feature

dimensionality. The output is then fed into the transformer blocks of the encoder. The feature extractor module contains approximately 4.2 million parameters.

The goal is to determine whether fine-tuning these low-level feature extraction layers, originally optimized during Wav2Vec2's self-supervised pre-training, provides benefits for the downstream ASR task compared to keeping their weights fixed. This experiment builds on the combination of the pre-trained Wav2Vec2-base encoder and the pre-trained BART-base decoder.

5.5.1 Training Details

The models in this experiment, as well as the subsequent experiments, are trained using a common framework and hyperparameters based on the findings of Experiment 5.4.1.

The AdamW optimizer [78] and a linear learning rate scheduler that includes a linear warmup phase is used. The specific settings vary slightly per dataset but remain consistent across experiments for that dataset:

- For LibriSpeech, the training uses a per-device batch size of 64, a learning rate of 3×10^{-4}, 5 epochs, and 400 warm-up steps.
- For VoxPopuli, the training uses a larger per-device batch size of 96, a lower learning rate of 1×10^{-4}, 6 epochs, and 100 warm-up steps.

Gradient accumulation is set to a single step for both datasets, and mixed precision training (`bf16`) [61] is enabled to improve efficiency.

To assess computational efficiency consistently across experiments, I measure the average time per forward and backward pass on a single, 10-second audio sample from the test set of the LibriSpeech dataset. This measurement involves 15 warm-up iterations followed by 500 measurement iterations to ensure stability.

5.5.2 Results and Analysis

The performance impact of freezing the feature extractor is evaluated on both the LibriSpeech and VoxPopuli datasets. The results, including WER components are presented in Tables 5.4 and 5.5. The computational cost is in Table 5.6.

To ensure robustness and mitigate the effects of random variations during training, each configuration is trained three times independently. The performance metrics reported in the tables represent the average values across these three runs.

Feature Extractor	Dev WER [%]	Test WER [%] ↓	S	D	I
Frozen	**8.1**	**8.3**	6.8	0.6	0.9
Trained	8.5	8.6	7.1	0.6	0.9

Table 5.4: Impact of freezing the Wav2Vec2 feature extractor versus fine-tuning it on ASR performance for the LibriSpeech (clean.100) dataset, using the pretrained Wav2Vec2-BART model. Metrics show dev and test Word Error Rate (WER %), along with test set error components (S, D, I).

Feature Extractor	Dev WER [%]	Test WER [%] ↓	S	D	I
Frozen	**10.47**	**10.81**	6.4	2.4	2.0
Trained	10.57	10.82	6.4	2.4	2.0

Table 5.5: Impact of freezing the Wav2Vec2 feature extractor versus fine-tuning it on ASR performance for the VoxPopuli (English) dataset, using the pretrained Wav2Vec2-BART model. Metrics show dev and test Word Error Rate (WER %), along with test set error components (S, D, I).

Feature Extractor	Time [ms] ↓
Frozen	52.4 ± 2
Trained	57.8 ± 2

Table 5.6: Computational cost comparison: Average time per forward and backward pass (ms) for the Wav2Vec2-BART model with a frozen versus fine-tuned feature extractor. Measurements were taken on a representative 10-second LibriSpeech audio sample.

Conclusion

The results presented in Tables 5.4 and 5.5 show that freezing the Wav2Vec2 feature extractor yields performance comparable to, or even slightly better than, fine-tuning it alongside the rest of the model.

Based on the results, audio features learned during Wav2Vec2's self-supervised pretraining do not require further adaptation for ASR tasks within this encoder-decoder framework. Freezing these initial convolutional layers offers the benefit of reducing the number of trainable parameters by 4.2 million.

As shown in Table 5.6, freezing the feature extractor slightly decreased the average computation time per step compared to full fine-tuning, which is expected. For the rest of the experiments, the feature extractor is frozen.

5.6 Experimental Setup 3

In the third experiment, I investigate the impact of adding convolutional adapter layers to the Wav2Vec2 encoder within the encoder-decoder model. Specifically, these layers act as temporal subsampling layers, reducing the sequence length of the encoder's output. The objective is to determine the optimal number of adapter layers and to evaluate their effect on model performance, parameter efficiency, and training efficiency.

The Wav2Vec2 encoder architecture consists of a feature extractor, feature projection, and transformer encoder blocks. By default, a single output vector of Wav2Vec2-base has a receptive field of approximately 25 ms [4]), representing a temporal span slightly less than a single character. In contrast, the BART decoder employs a sentence-piece tokenizer as its input processor, where a single hidden vector of BART-base represents approximately 4 characters. To better align the receptive field of the Wav2Vec2 output vectors with BART's inputs, I experiment with a convolution-based *Wav2Vec2Adapter* module that subsamples Wav2Vec2's output along the time dimension. This adapter extends the encoder by transforming its output representations through a series of 1D convolutional layers with

a kernel size of 3, stride of 2, and padding of 1, mapping the input dimensionality of 768 to an intermediate dimensionality of 1536 before projecting back to the required output dimensionality for compatibility with the decoder. The number of adapter layers varies from 0 (no adapters) to 6.

The experiment utilizes the same datasets as described earlier with the same data processing.

5.6.1 Training Details

Based on the findings from Experiment 5.7.1, the Wav2Vec2 feature extractor is kept frozen throughout this experiment. The training configuration otherwise follows the common setup detailed in Section 5.5.1, utilizing the AdamW optimizer, linear scheduler with warmup, specified learning rates, batch sizes, epochs, and mixed precision training (`bf16`). The variable under investigation is the number of convolutional adapter layers added between the encoder's transformer blocks and the decoder's cross-attention mechanism. Computational efficiency is measured using the standardized timing methodology described in Section 5.5.1.

5.6.2 Parameter Efficiency and Performance Analysis

This experiment investigates the trade-off between model parameters (total and trainable), training efficiency (average time per step), and performance (Word Error Rate — WER). Figures 5.2 (LibriSpeech) and 5.3 (VoxPopuli) visualize this trade-off as Pareto fronts. These plots show the total and trainable parameters on the y-axis against the achieved WER on the x-axis for each adapter configuration. This shows a direct comparison of how different adapter configurations impact both the size and accuracy of the model.

Furthermore, WER progress is tracked during training on the validation sets of both datasets. The progression of WER is illustrated in Figure 5.4 for LibriSpeech and VoxPopuli, providing information on how the number of adapters affects the learning of the model over epochs.

Tables 5.7 and 5.8 provide a detailed breakdown of the WER, including its components (substitutions, deletions, and insertions), on both the validation and test sets for each dataset and adapter configuration.

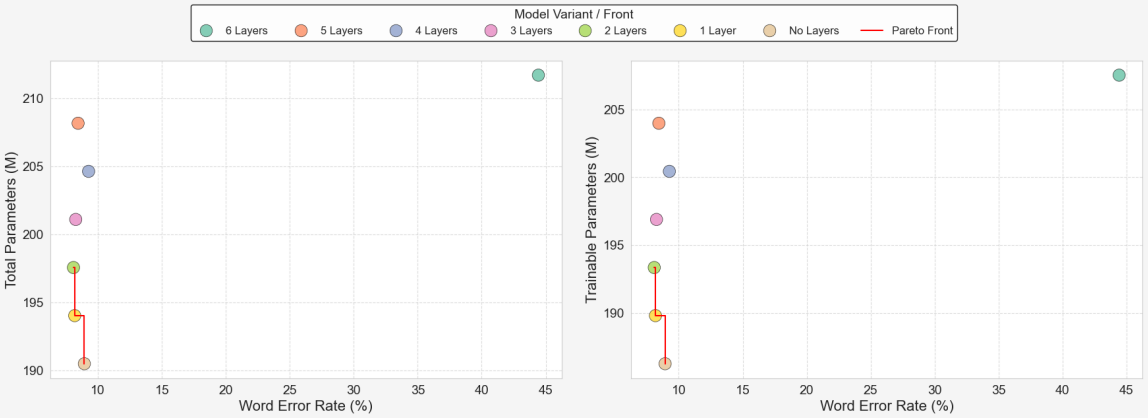


Figure 5.2: Pareto front of total parameters/trainable parameters versus Word Error Rate (WER) for LibriSpeech.

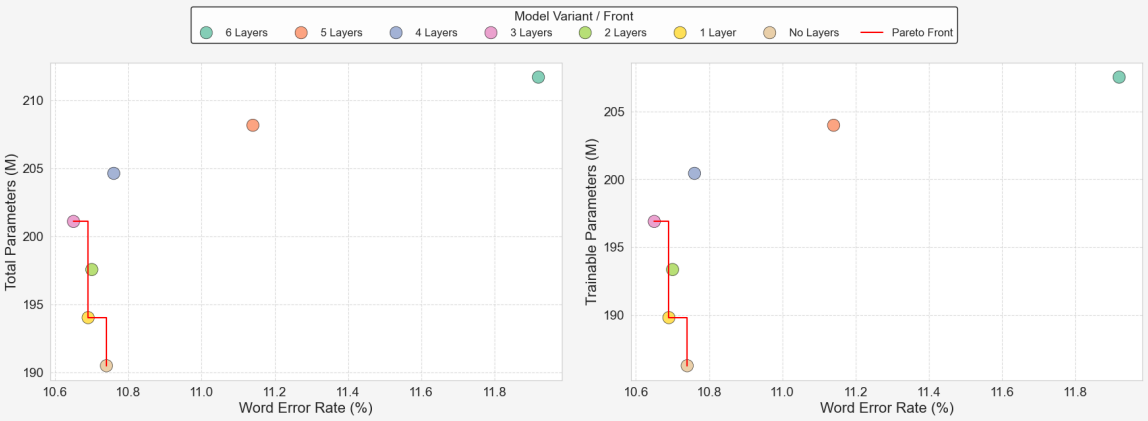


Figure 5.3: Pareto front of total parameters/trainable parameters versus Word Error Rate (WER) for VoxPopuli.

Figure 5.4: Progression of WER on validation part for both datasets.

Conclusion

The third experiment explores the impact of convolutional adapter layers in the Wav2Vec2 encoder of the encoder-decoder model. For LibriSpeech, 2 adapters yield the lowest WER, as shown in Table 5.7, but performance drops significantly with 6 adapters, likely due to over-compression because with 6 adapters, the sequence length gets 64-times shorter. For VoxPopuli, configurations with 1 to 4 adapters yield the best results.

WER progression (Figure 5.4) shows faster convergence with 1–3 adapters. Efficiency plots (Figures 5.2, 5.3) reveal that more parameters do not mean the model will be linearly slower, that is because with more adapter layers the input length for the decoder becomes shorter. Time measurements reveal minimal variation in the average forward and backward pass time across different adapter configurations, ranging from 48.10ms to 50.90ms (Table 5.9). This indicates that adding convolutional adapter layers, within the tested range, has a negligible impact on processing speed. It's important to note that these measurements are specific to the utilized hardware and software environment, and different implementations or hardware might yield varying results, more details in Appendix A.

In summary, incorporating 1 to 3 adapter layers effectively improves performance by facilitating better alignment between the encoder and decoder representations. However, an excessive number of layers negatively impacts accuracy.

Adapters	Dev WER [%] ↓	Test WER [%] ↓	S	D	I	Adapter Params [M]
0	8.5	8.9	7.3	0.7	1.0	0
1	8.1	8.2	6.7	0.7	0.8	3.54
2	**7.9**	**8.1**	6.6	0.6	0.9	7.08
3	8.3	8.6	6.9	0.6	0.8	10.6
4	9.1	9.3	7.7	0.6	1.0	14.1
5	8.3	8.4	6.9	0.6	0.9	17.7
6	45	44	34	6.2	4.5	21.2

Table 5.7: Effect of varying the number of convolutional adapter layers between the Wav2Vec2 encoder and BART decoder on ASR performance for the LibriSpeech (clean.100) dataset. Metrics include dev and test Word Error Rate (WER %), test set error components (S, D, I), and the number of parameters in the adapter layers (M). The feature extractor is frozen. Numbers are rounded to 1 decimal place.

Adapters	Dev WER [%] ↓	Test WER [%] ↓	S	D	I	Adapter Params [M]
0	10.44	10.74	6.2	2.4	2.1	0
1	10.30	10.69	6.2	2.4	2.1	3.54
2	10.31	10.70	6.3	2.3	2.1	7.08
3	10.29	**10.65**	6.3	2.4	2.0	10.6
4	**10.26**	10.76	6.3	2.4	2.0	14.1
5	10.58	11.14	6.5	2.5	2.1	17.7
6	11.21	11.92	7.0	2.6	2.2	21.2

Table 5.8: Effect of varying the number of convolutional adapter layers between the Wav2Vec2 encoder and BART decoder on ASR performance for the VoxPopuli (English) dataset. Metrics include dev and test Word Error Rate (WER %), test set error components (S, D, I), and the number of parameters in the adapter layers (M). The feature extractor is frozen.

5.7 Experimental Setup 4

In the fourth experiment, I investigate the integration of Low-Rank Adaptation (LoRA) adapters [53] into the SpeechEncoderDecoder model to enhance parameter-efficient fine-tuning. Building on the findings from Experimental Setup 5.7.1 and Experimental Setup 5.6, where three convolutional adapter layers are shown to be optimal for aligning the Wav2Vec2 encoder outputs with the BART decoder, this experiment aims to further reduce the computational resources of training, by using LoRA adapters. These adapters are applied to both the encoder and decoder, targeting key linear layers within the transformer architecture. Besides LoRA adapters, the convolutional adapters and the last linear layer (`lm_head`) need to be trained.

5.7.1 Experimental Design

In this setup, LoRA adapters are applied to the linear layers within the attention and feed-forward blocks of both the Wav2Vec2 encoder and the BART decoder. These correspond to the query, key, value, and output projections in the multi-head attention mechanism, as

Number of Adapters	Time [ms] ↓
0	49.4 ± 1
1	50.0 ± 1
2	48.1 ± 1
3	50.9 ± 1
4	50.7 ± 1
5	49.3 ± 1
6	49.7 ± 1

Table 5.9: Computational cost comparison: Average time per forward and backward pass (ms) for the Wav2Vec2-BART model with varying numbers of convolutional adapter layers (0 to 6). Measurements were taken on a representative 10-second LibriSpeech audio sample.

well as the intermediate and output dense layers in the feed-forward networks. The number of trainable LoRA parameters depends on the chosen rank value.

In addition to the LoRA adapters, the convolutional adapter layers (10.6M parameters) and the `lm_head` (38.6M parameters) are trained. Together, these two components dominate the trainable parameter count and comprise roughly 25% of the total parameters in the model.

For the initial LoRA experiments, I focus on varying the rank r and the scaling factor α. For α value, I explore values that are significantly higher than commonly used heuristics (e.g., $\alpha = r$ or $\alpha = 2r$). The rationale is to investigate whether a stronger adaptation signal, achieved by increasing the magnitude of the LoRA update ($\frac{\alpha}{r} \cdot BA$), is necessary to effectively fine-tune the pre-trained model for the ASR task. This approach aims to potentially achieve a lower Word Error Rate by allowing for more substantial deviations from the original weights. However, this choice risks training instability, as large α/r ratios amplify gradient updates to the effective weights, potentially leading to divergence and training instability.

Training Details

The training configuration mirrors that of Experimental Setup 2; only thing that changed was the number of epochs on the VoxPopuli [125] dataset from 6 to 5. I made this decision since I have not seen much improvement in training for longer, and also, in some cases, the model got worse.

Performance is evaluated using the same approach as in previous experiments, with results reported in Tables 5.10 and 5.11. These tables include the LoRA trainable parameters (excluding convolutional adapters and `lm_head`) alongside WER for each configuration. Computational efficiency is measured the exact same way as in previous experiment 5.6 and is summarized in Table 5.12. Figures 5.5 and 5.6 show the trade-off between different LoRA configurations, visualized as Pareto fronts.

Conclusion

LoRA proves effective for parameter-efficient fine-tuning in this setup. An increase in the rank r consistently leads to improved performance, as indicated by a reduction in the WER. This improvement is attributed to the fact that a higher rank allows the LoRA layers more degrees of freedom to adapt the original weights.

Rank	Alpha	Dev WER [%] ↓	Test WER [%] ↓	S	D	I	LoRA Params [M]
Full FT	—	8.29	8.26	6.9	0.6	0.8	—
16	128	10.4	10.7	8.7	0.9	1.0	4.6
	192	10.2	10.5	8.7	0.8	1.1	
	256	10.4	10.5	8.8	0.8	1.0	
	384	10.5	10.9	8.8	1.1	1.0	
	512	158	161	82.1	9.5	70.4	
32	128	10.1	10.2	8.3	0.9	1.0	9.1
	192	9.8	9.9	8.2	0.8	1.0	
	256	9.4	9.6	8.0	0.6	1.0	
	384	9.6	9.6	8.0	0.6	1.0	
	512	9.6	9.8	8.3	0.6	1.0	
64	128	9.7	9.9	8.1	0.8	1.0	18.3
	192	9.3	9.4	7.7	0.7	0.9	
	256	9.4	9.5	7.8	0.7	1.0	
	384	9.0	9.0	7.5	0.6	0.9	
	512	9.1	9.2	7.6	0.7	0.9	
128	128	9.4	9.8	8.0	0.8	1.0	36.6
	192	9.1	9.2	7.7	0.6	0.9	
	256	8.9	9.1	7.5	0.8	0.9	
	384	8.8	9.0	7.4	0.7	0.9	
	512	**8.6**	**8.7**	7.2	0.7	0.8	

Table 5.10: Performance evaluation of Low-Rank Adaptation (LoRA) for fine-tuning the Wav2Vec2-BART ASR model on LibriSpeech (clean.100), varying rank (r) and alpha (α) values. The model includes 3 convolutional adapters and a frozen feature extractor. Metrics include dev and test WER (%), error components (S, D, I), and number of LoRA-specific parameters (M). The full fine-tuning (Full FT) baseline with 3 adapters is included for comparison. Numbers are rounded to 1 decimal place.

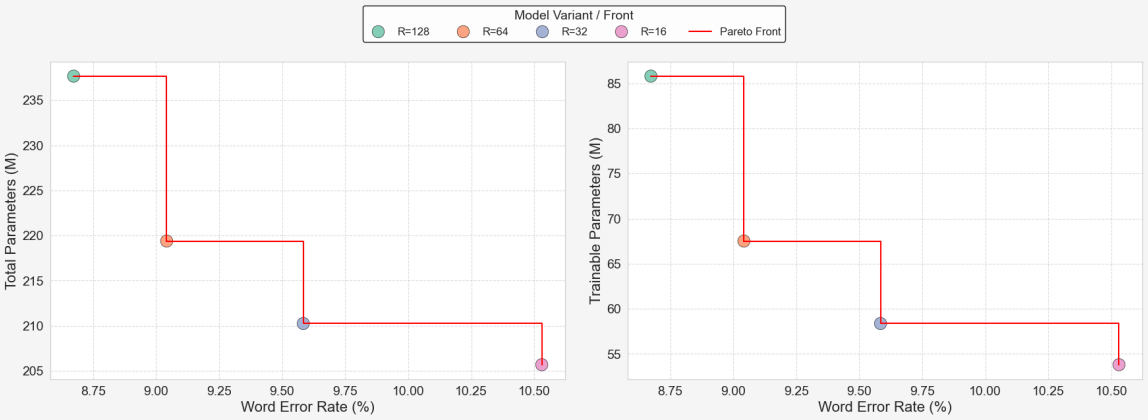


Figure 5.5: Pareto front of total parameters/trainable parameters versus Word Error Rate for LibriSpeech on circa 10 seconds long sample for different LoRA configurations.

Rank	Alpha	Dev WER [%] ↓	Test WER [%] ↓	S	D	I	LoRA Params [M]
Full FT	—	10.29	10.65	6.3	2.4	2.0	—
16	128	12.9	13.5	8.2	2.8	2.5	4.6
	192	16.1	16.4	8.6	5.4	2.4	
	256	12.6	12.9	7.9	2.5	2.4	
	384	12.3	12.8	7.7	2.8	2.2	
	512	12.7	13.1	7.9	2.8	2.4	
32	128	12.5	13.2	7.9	2.9	2.5	9.1
	192	12.3	12.8	7.7	2.7	2.4	
	256	12.9	13.6	8.1	3.1	2.3	
	384	11.7	12.3	7.5	2.5	2.3	
	512	11.6	12.2	7.4	2.5	2.3	
64	128	13.1	13.6	8.1	3.0	2.4	18.3
	192	12.1	12.7	7.6	2.6	2.4	
	256	11.9	12.4	7.6	2.5	2.3	
	384	11.6	12.0	7.2	2.5	2.3	
	512	11.7	**11.9**	7.2	2.5	2.1	
128	128	16.4	16.7	9.2	4.5	3.1	36.6
	192	11.9	12.7	7.7	2.6	2.5	
	256	11.7	12.5	7.5	2.6	2.4	
	384	**11.3**	12.1	7.3	2.5	2.3	
	512	12.1	12.5	7.4	2.8	2.3	

Table 5.11: Performance evaluation of Low-Rank Adaptation (LoRA) for fine-tuning the Wav2Vec2-BART ASR model on VoxPopuli (English), varying rank (r) and alpha (α) values. The model includes 3 convolutional adapters and a frozen feature extractor. Metrics include dev and test WER (%), error components (S, D, I), and number of LoRA-specific parameters (M). The full fine-tuning (Full FT) baseline with 3 adapters is included for comparison. Numbers are rounded to 1 decimal place.

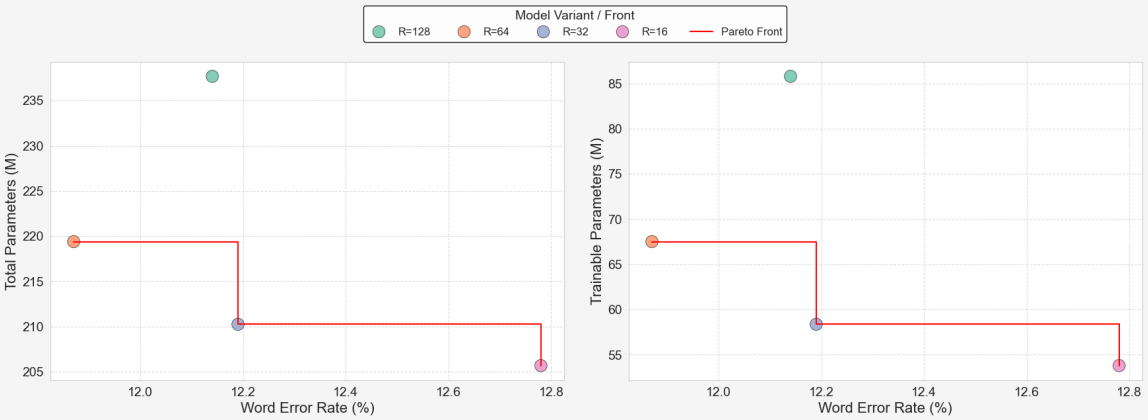


Figure 5.6: Pareto front of total parameters/trainable parameters versus Word Error Rate for VoxPopuli on circa 10 seconds long sample for different LoRA configurations.

Rank	Time [ms] ↓
16	101 ± 2
32	101 ± 2
64	103 ± 3
128	102 ± 3

Table 5.12: Computational cost comparison: Average time per forward and backward pass (ms) for the Wav2Vec2-BART model using LoRA adapters with varying ranks (r). Measurements were taken on a representative 10-second LibriSpeech audio sample.

The scaling factor α has a more complex effect. I used high α values to try to get the best WER, but very high α (like 512 with r=16 on LibriSpeech) made training unstable. From the results, we see that the best LoRA setup is different for each dataset.

LoRA uses many fewer trainable parameters than fine-tuning the whole model. Even the biggest LoRA setup (r=128) adds only 36.6M parameters. Adding the convolutional adapters (10.6M) and `lm_head` (38.6M), the total count is still much smaller than the full fine-tuning. Table 5.12 shows that changing the LoRA rank doesn't change the training time much (101-103ms per forward and backward pass).

The Pareto fronts (Figures 5.5 and 5.6) show the trade-off between parameters and WER. Choosing the right rank and alpha gives good performance without adding many parameters. Compared to Experiment , LoRA had slightly worse performance on LibriSpeech and on VoxPopuli. The focus was on using few resources, not just getting the absolute best WER.

In short, LoRA is good for fine-tuning this encoder-decoder model because it reduces parameters a lot while keeping good performance. It is important to say that this parameter efficiency becomes even more advantageous for larger sized models, where the number of parameters added by LoRA is much smaller relative to the total model size. The best settings depend on the dataset. The training time stays about the same for different ranks but are 2 times slower than full-finetuing.

5.8 Experimental Setup 5

In the fifth experiment, I use Weight-Decomposed Low-Rank Adaptation (DoRA) [76], which is a modified version of LoRA. This experiment continues to use the optimal configuration identified in Experimental Setup 5.7.1 and 5.6, which includes three convolutional adapter layers and frozen feature extractor.

In this experimental setup, DoRA adapters are applied to the same linear layers within the Wav2Vec2 encoder and BART decoder as in Experimental Setup 5.7. I vary the rank r and α values for the LoRA component within DoRA. The number of trainable parameters in DoRA will be slightly higher than in the corresponding LoRA configurations due to the inclusion of the trainable magnitude vector m. For each linear layer with a weight matrix of size $d \times k$, the magnitude vector introduces an additional k trainable parameters.

The training details, including optimizer (AdamW), learning rate scheduler, batch sizes, and gradient accumulation, are identical to those used in Experimental Setup 5.7. Performance is evaluated using Word Error Rate (WER). Computational efficiency is measured

using the average time per forward and backward pass. The measurements are summarized in Table 5.15.

Rank	Alpha	Dev WER [%] ↓	Test WER [%] ↓	S	D	I	DoRA Params [M]
Full FT	—	8.29	8.26	6.9	0.6	0.8	—
16	256	10.1	10.2	8.3	0.9	1.0	4.7
	384	10.4	10.5	8.6	0.9	1.0	
	512	10.4	10.6	8.7	0.8	1.1	
32	256	9.6	9.7	8.0	0.8	0.9	9.3
	384	9.5	9.6	7.9	0.7	1.0	
	512	9.5	9.5	7.8	0.8	0.9	
64	256	9.1	9.1	7.5	0.6	0.9	18.4
	384	9.2	9.2	7.6	0.7	0.9	
	512	196	201	82	4.6	113	
128	256	9.0	9.1	7.5	0.8	0.8	36.7
	384	8.9	9.1	7.4	0.8	0.9	
	512	**8.7**	**8.9**	7.3	0.7	0.9	

Table 5.13: Performance evaluation of Weight-Decomposed Low-Rank Adaptation (DoRA) for fine-tuning the Wav2Vec2-BART ASR model on LibriSpeech (clean.100), varying rank (r) and alpha (α) values. The model includes 3 convolutional adapters and a frozen feature extractor. Metrics include dev and test WER (%), error components (S, D, I), and number of DoRA-specific parameters (M). The full fine-tuning (Full FT) baseline with 3 adapters is included for comparison. Numbers are rounded to 1 decimal place.

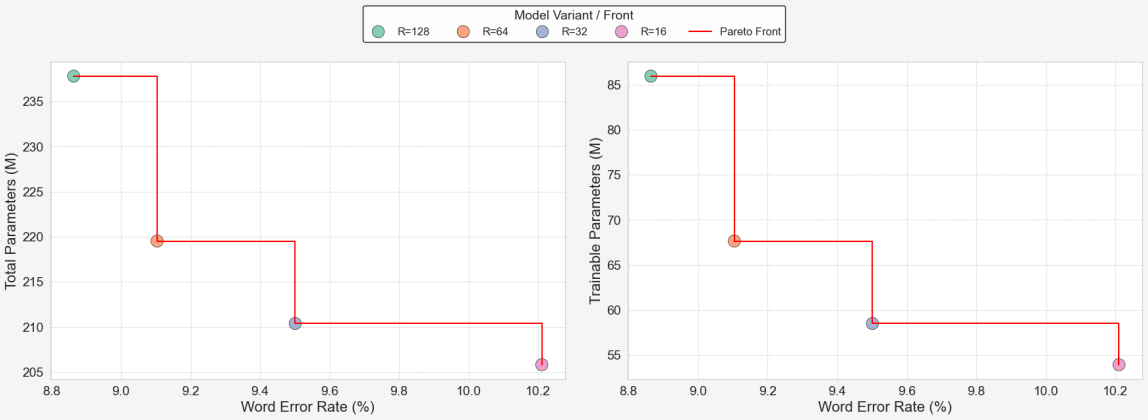


Figure 5.7: Pareto front of total parameters/trainable parameters versus Word Error Rate for LibriSpeech on a circa 10-second long sample for different DoRA configurations.

Conclusion

The DoRA experiments evaluate an alternative parameter-efficient fine-tuning technique that tries to enhance LoRA. Comparing these results to those from Experimental Setup 5.7 in order to assess DoRA's relative effectiveness in this task.

Rank	Alpha	Dev WER [%] ↓	Test WER [%] ↓	S	D	I	DoRA Params [M]
Full FT	—	10.3	10.7	6.3	2.4	2.0	—
16	256	12.3	12.9	7.9	2.5	2.4	4.7
	384	12.0	12.4	7.5	2.7	2.3	
	512	12.5	12.9	7.6	3.0	2.3	
32	256	12.6	13.2	8.0	2.7	2.5	9.3
	384	11.9	12.5	7.4	2.8	2.3	
	512	11.6	12.0	7.2	2.5	2.3	
64	256	11.9	12.4	7.4	2.6	2.3	18.4
	384	12.9	13.4	7.8	3.0	2.5	
	512	**11.4**	**11.9**	7.1	2.7	2.2	
128	256	12.5	13.3	7.7	3.2	2.4	36.7
	384	11.9	12.4	7.3	2.7	2.4	
	512	12.6	12.8	7.4	3.1	2.3	

Table 5.14: Performance evaluation of Weight-Decomposed Low-Rank Adaptation (DoRA) for fine-tuning the Wav2Vec2-BART ASR model on VoxPopuli (English), varying rank (r) and alpha (α) values. The model includes 3 convolutional adapters and a frozen feature extractor. Metrics include dev and test WER (%), error components (S, D, I), and number of DoRA-specific parameters (M). The full fine-tuning (Full FT) baseline with 3 adapters is included for comparison. Numbers are rounded to 1 decimal place.

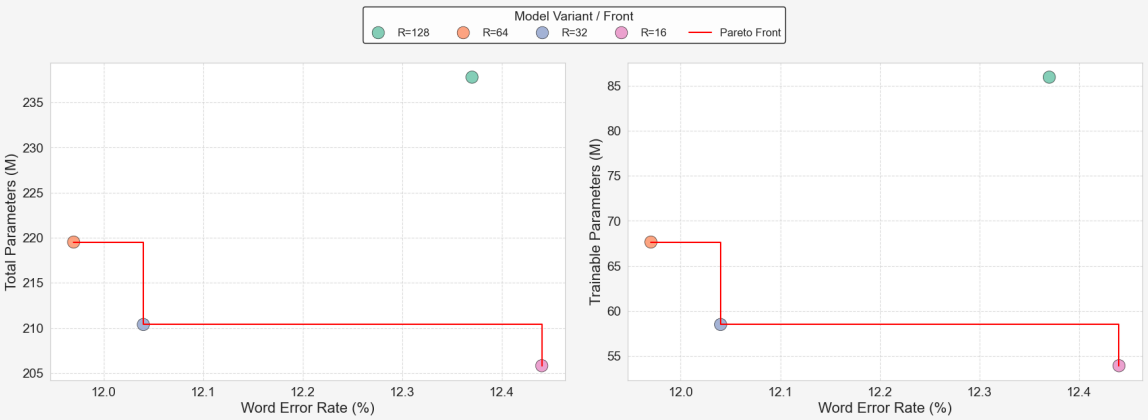


Figure 5.8: Pareto front of total parameters/trainable parameters versus Word Error Rate for VoxPopuli on a circa 10-second long sample for different DoRA configurations.

Similar to LoRA, increasing the rank r within DoRA generally tends to improve performance (lower WER), although this trend is not perfectly consistent, particularly on the VoxPopuli dataset (Table 5.14). The scaling factor α continues to show a complex relationship with performance, with higher values often yielding better results but also increasing the risk of training instability, as seen with the $r = 64, \alpha = 512$ configuration on LibriSpeech (Table 5.13). In terms of absolute performance, the best DoRA configurations achieved slightly higher WER compared to the best LoRA configurations on both datasets.

Rank	Time [ms] ↓
16	184 ± 3
32	181 ± 4
64	185 ± 3
128	184 ± 4

Table 5.15: Computational cost comparison: Average time per forward and backward pass (ms) for the Wav2Vec2-BART model using DoRA adapters with varying ranks (r). Measurements were taken on a representative 10-second LibriSpeech audio sample.

Regarding parameter efficiency, DoRA introduces a marginal increase in trainable parameters compared to LoRA due to the addition of the magnitude vector m. For instance, with $r = 128$, DoRA adds 36.7M parameters compared to LoRA's 36.6M. DoRA maintains a similar high level of parameter efficiency as LoRA compared to full fine-tuning.

A significant difference emerges in computational cost. Table 5.15 shows that the average time per forward and backward pass for DoRA ranges from 181ms to 185ms, irrespective of rank. This is substantially slower than LoRA, which ranged from 101ms to 103ms, representing an approximate 80% increase in step time. Compared to full fine-tuning (51ms from Table 5.9), DoRA is about 3.6 times slower per step. This overhead comes from the additional computations involved in the weight decomposition and normalization specific to DoRA.

5.9 Experimental Setup 6

This sixth experiment investigates the influence of encoder's and decoder's pre-training data domain strategies on downstream ASR performance, specifically when targeting the VoxPopuli dataset because the pre-trained wav2vec2 base model includes LibriSpeech in its pre-training The hypothesis is that domain-specific encoder pre-training or decoder continuous pre-training might provide better initialization and lead to improved performance when fine-tuning the full encoder-decoder model on the labeled VoxPopuli English subset.

5.9.1 Experimental Design: Encoder Pre-training

This part of the experiment builds directly upon the optimal configuration identified in previous setups. The only variable changed here is the specific pre-trained checkpoint used to initialize the Wav2Vec2 encoder component. I use Wav2Vec2 checkpoints pre-trained on varying amounts and subsets of the VoxPopuli corpus. The following checkpoints are evaluated:

- `facebook/wav2vec2-base-en-voxpopuli-v2`: Pre-trained on 24.1k hours of unlabeled English VoxPopuli data.
- `facebook/wav2vec2-base-10k-voxpopuli-ft-en`: Pre-trained on 10k hours of unlabeled VoxPopuli, then fine-tuned on labeled English VoxPopuli dataset.
- `facebook/wav2vec2-base-100k-voxpopuli`: Pre-trained on 100k hours of unlabeled multilingual VoxPopuli data.

- `facebook/wav2vec2-base-10k-voxpopuli`: Pre-trained on 10k hours of unlabeled multilingual VoxPopuli data.

The performance of these models is compared against the baseline model from Experimental Setup 5.6 when trained and evaluated on the VoxPopuli English dataset. The models are trained exclusively on the VoxPopuli (English) dataset using the same training framework detailed in Section 5.5.1.

5.9.2 Experimental Design: Decoder Continuous Pre-training

For this part of the experiment, I apply additional pre-training steps to the decoder, using the VoxPopuli English training text data. The goal is to adapt the decoder to the target domain's linguistic style using self-supervised objectives similar to the original paper [69]. Since the original BART pre-training recipe wasn't fully detailed by Facebook, I use and compare two common strategies based on their paper and available code examples:

- **Text Infilling**: This objective aims to teach the model to predict missing text spans. Spans of text are sampled (lengths drawn from a Poisson distribution, $\lambda = 3.0$), and each chosen span is replaced by a single `[MASK]` token. The decoder is trained to reconstruct the original, uncorrupted text sequence. This differs slightly from some interpretations where multiple mask tokens might be used, but captures the core idea of predicting variable-length missing content.
- **Denoising + Sentence Permutation**: This objective combines text infilling with sentence-level shuffling, mimicking the combination reported in the original BART paper. It involves:
    1. *Sentence Permutation*: Documents are segmented into sentences (based on `EOS` tokens), and the order of these sentences is randomly shuffled with a certain probability.
    2. *Text Infilling*: Similar to the above, text spans/whole words are masked based on a mask ratio (`mask_ratio=0.15`) and Poisson-sampled lengths ($\lambda = 3.0$), are replaced by `[MASK]` tokens. The decoder must reconstruct the original text from this doubly corrupted input.

These decoder pre-training steps were performed on the VoxPopuli English text data. Both are continuously pre-trained on additional 500 epochs on training data.

5.9.3 Results and Analysis

The performance results for each encoder and decoder pre-training configuration are presented in Table 5.16. The table includes the Word Error Rate (WER) and its components (substitutions, deletions, insertions) for the VoxPopuli English test set.

Conclusion

For the encoder, the results reveal a significant benefit from domain-specific pre-training. The Wav2Vec2 encoder pre-trained specifically on English VoxPopuli data achieves the lowest WER (9.88% test), substantially outperforming the standard `wav2vec2-base` baseline (10.65% test). This indicates that aligning the encoder's pre-training data domain

Pre-training Configuration	Dev WER [%] ↓	Test WER [%] ↓	S	D	I
Baseline (53k h)	10.29	10.65	6.3	2.4	2.0
Encoder Pre-training Variations					
VoxPopuli EN (24.1k h)	9.20	9.88	5.6	2.2	2.1
VoxPopuli (10k h) + FT EN	10.55	10.88	6.5	2.3	2.1
VoxPopuli Multi (100k h)	10.62	11.22	6.8	2.4	2.0
VoxPopuli Multi (10k h)	11.15	12.03	7.2	2.5	2.3
Decoder Continuous Pre-training					
Text Infilling	10.26	10.71	6.4	2.3	2.1
Denoising + Sent. Perm.	10.21	10.80	6.4	2.3	2.1

Table 5.16: Impact of varying encoder pre-training data domain and decoder continuous pre-training strategy on ASR performance on the VoxPopuli (English) dataset. Compares different Wav2Vec2 encoder checkpoints and two BART decoder pre-training objectives against the baseline model (3 adapters, frozen feature extractor). Metrics include dev and test WER (%), and test set error components (S, D, I).

and language with the target task provides a considerable advantage. Notably, other VoxPopuli-based encoders, including those pre-trained on multilingual data or incorporating fine-tuning, did not surpass the baseline performance.

Conversely, for the decoder, applying continuous pre-training on domain-specific data using BART-inspired objectives does not yield improvements. Both the Text Infilling (10.71% test WER) and the Denoising + Sentence Permutation (10.80% test WER) strategies resulted in slightly higher error rates compared to the baseline model. The variance in the results does not show any difference from the baseline and is not significant enough. In this setup, adapting the pre-trained BART decoder further using self-supervised objectives on the target text domain does not show that it should increase the performance of encoder decoder for ASR task.

These experiments utilize a fixed set of fine-tuning hyperparameters based on the baseline model. The optimal hyperparameters might differ for each pre-trained encoder and decoder checkpoint. Specific hyperparameter tuning for each checkpoint might yield different performance results.

5.10 Experimental Setup 7

This final experiment aims to compare the performance of the encoder-decoder model against a Connectionist Temporal Classification (CTC) based model. CTC is explained in section 3.2.

5.10.1 CTC Model Configuration

The CTC model uses the same pre-trained Wav2Vec2-base encoder, which is used with the previous experiments for the encoder-decoder model. The CTC model architecture uses a standard CTC setup where a linear classification head is added directly on top of the Wav2Vec2 encoder's output sequence. This head projects the encoder's final hidden states

into logit vectors, where the dimension of each vector corresponds to the size of the target vocabulary.

A difference from the encoder-decoder setup is also in the vocabulary construction. Rather than using the large BART vocabulary, a much smaller vocabulary is created specifically for each dataset based on the characters present in their training transcriptions. This involves extracting all unique characters from the target text, sorting them, and assigning unique integer IDs. Special tokens are included: a padding token, an unknown token, and a word delimiter token (used to represent spaces).

Consistent with the findings from Experiment 5.7.1, the Wav2Vec2 feature extractor layers are kept frozen during training.

5.10.2 Training Details

The CTC models are trained using CTC loss function. Standard optimization techniques are used, including the AdamW optimizer and a learning rate scheduler. The training parameters for the CTC models (such as learning rate, number of epochs, and scheduler details) are determined empirically. A few training runs are conducted for each dataset to identify settings that yield reasonable convergence and representative performance for the CTC architecture on the given task.

5.10.3 Results and Analysis

The performance comparison between the CTC model and the baseline encoder-decoder model on both datasets is presented in Table 5.17.

Dataset	Model	Dev WER [%] ↓	Test WER [%] ↓	S	D	I
LibriSpeech	Encoder-decoder	8.29	8.26	6.9	0.6	0.8
	CTC	11.87	11.87	5.7	0.4	5.8
VoxPopuli	Encoder-decoder	10.29	10.65	6.3	2.4	2.0
	CTC	16.61	17.19	8.5	2.8	5.9

Table 5.17: Performance comparison between a Connectionist Temporal Classification (CTC) model (Wav2Vec2-base encoder + CTC head) and the baseline Wav2Vec2-BART encoder-decoder model (3 adapters, frozen feature extractor) on the LibriSpeech (clean.100) and VoxPopuli (English) datasets. Metrics include dev and test Word Error Rate (WER %), and test set error components (S, D, I).

Conclusion

The results clearly show that the encoder-decoder model significantly outperforms the CTC model in terms of Word Error Rate on both datasets.

Looking at the error components reveals that the CTC model's primary weakness lies in its high insertion rate (I) on both LibriSpeech (5.8% vs 0.8%) and VoxPopuli (5.9% vs 2.0%). While CTC shows fewer substitutions and deletions on LibriSpeech, the excessive insertions increase its overall WER. On VoxPopuli, CTC errors are higher across all categories. This pattern suggests that the simpler CTC architecture, which lacks the sequence-level language modeling provided by the decoder, struggles more to correctly determine word boundaries or handle silence.

In terms of computational efficiency, the average forward and backward pass per sample (measured under the same conditions as previous experiments, using the same 10-second LibriSpeech sample) for the CTC model is approximately **30ms ± 1ms**. Compared to the baseline encoder-decoder model, the CTC model is roughly 38% faster (encoder-decoder model is aproximately 48ms for best setup). This speed advantage is primarily due to the non-autoregressive nature of CTC decoding, which avoids the sequential generation process inherent in the encoder-decoder architecture. Secondly, the CTC architecture is smaller overall, as it omits the decoder component of the transformer model.

In conclusion, this experiment highlights a clear trade-off between accuracy and speed when comparing the CTC and encoder-decoder architectures for ASR task.

5.11 Model Optimization and Performance Benchmarking

The culmination of the experimental work involves a dedicated training run to maximize performance using the optimal configuration identified in previous experiments: a Wav2Vec2-base encoder pre-trained on English VoxPopuli, a BART-base decoder, three convolutional adapter layers, and a frozen feature extractor. To achieve the best possible Word Error Rate on the VoxPopuli English dataset, this final fine-tuning stage incorporates several enhancements: the number of training epochs is increased to 20, a weight decay of 0.01 and a label smoothing factor of 0.05 are introduced. Furthermore, SpecAugment [91] is utilized with a time masking probability of 0.25 (length 30, minimum 2 masks) and a feature masking probability of 0.3 (length 30, minimum 1 mask) to improve model robustness. The learning rate scheduler was changed to `cosine_with_min_lr` (with a minimum learning rate of $5e-9$), and the warmup period was extended to 2000 steps.

This refined training strategy resulted in a test WER of 8.8% on the VoxPopuli English dataset. The corresponding model checkpoint has been made publicly available on the Hugging Face Hub[1].

To contextualize this achieved performance, Table 5.18 presents a comparative analysis against several other Automatic Speech Recognition (ASR) models of varying architectures and parameter scales. The closest comparable model is the ESPnet.

Model	**Size (Parameters)**	**Test WER [%] ↓**
Whisper (large-v2) [103]	1.54 B	7.3
Whisper (small) [103]	244 M	8.5
XLS-R (1B) [3]	1.0 B	8.8
Mu2SLAM-spm [15]	∼0.7 B	9.2
XLS-R (300M) [3]	0.3 B	10.2
MMS (1B FLANG) [98]	1.0 B	10.3
ESPnet (WavLM + mBART large) [127]	∼0.4 B	11.3
Wav2Vec2-base + BART-base	**201 M**	**8.8**

Table 5.18: VoxPopuli test set WER comparison across different ASR models. Note: Model sizes are approximate for some entries based on reported figures.

[1] https://huggingface.co/BUT-FIT/wav2vec2-base_bart-base_voxpopuli-en

Chapter 6

Conclusion

The primary goal of this thesis was to investigate the training of encoder-decoder transformer models for Automatic Speech Recognition, focusing on initialization strategies, adapter layers, and parameter-efficient fine-tuning. This goal was achieved through systematic experiments on the LibriSpeech and VoxPopuli datasets.

Experiments conclusively showed that initializing both the Wav2Vec2 encoder and BART decoder with pre-trained weights yields superior Word Error Rate and convergence compared to other strategies. The work established a strong baseline using this approach, achieving WERs of 8.08% on LibriSpeech test-clean and 10.65% on VoxPopuli English test with an optimal adapter configuration. Investigating the encoder-decoder connection, the thesis demonstrated that incorporating a small number of convolutional adapter layers effectively improved WER by better aligning the temporal resolutions of the encoder and decoder. The evaluation of PEFT methods confirmed their effectiveness. Low-Rank Adaptation (LoRA) significantly reduced the number of trainable parameters while achieving performance close to full fine-tuning (8.67% WER on LibriSpeech with rank 128), offering a balance between efficiency and accuracy. Weight-Decomposed Low-Rank Adaptation (DoRA) provided similar parameter savings but resulted in slightly lower accuracy and substantially slower training step times in these experiments. The work revealed the benefit of domain-specific pre-training for the encoder; using a Wav2Vec2 model pre-trained on English VoxPopuli data significantly improved results on the VoxPopuli task (9.88% test WER). Additional decoder pre-training on the target text domain did not yield benefits in this setup. A final optimized training run, incorporating extensive fine-tuning parameters and SpecAugment, further pushed the performance on VoxPopuli English to a test WER of 8.85%. A comparison with a Connectionist Temporal Classification (CTC) baseline confirmed that the encoder-decoder architecture achieves better accuracy.

My personal contribution covered the design and execution of these experiments, including the implementation of adapter layers, the systematic evaluation of configurations, and the analysis of the resulting trade-offs between accuracy, efficiency, and computational cost.

Looking ahead, this work can be extended by exploring more advanced PEFT methods, investigating different adapter architectures, and applying these strategies to newer, larger foundation models. Replacement of the BART decoder with a more recent decoder could further improve the results.In summary, this thesis provides evidence and practical guidance on initializing and efficiently fine-tuning encoder-decoder ASR models. By systematically evaluating the roles of pre-training, adapters, and PEFT, this work contributes valuable insights for building accurate and resource-conscious speech recognition systems.

Bibliography

[1] Jana, A. *Machine Translation using Recurrent Neural Network and PyTorch* online. 2020-10-24. Available at: https://adeveloperdiary.com/data-science/deep-learning/nlp/machine-translation-recurrent-neural-network-pytorch/#conclusion. [cit. 2024-11-06].

[2] Ansell, A.; Ponti, E.; Korhonen, A. and Vulić, I. Composable Sparse Fine-Tuning for Cross-Lingual Transfer. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2022, p. 1778–1796. Available at: http://dx.doi.org/10.18653/v1/2022.acl-long.125.

[3] Babu, A.; Wang, C.; Tjandra, A.; Lakhotia, K.; Xu, Q. et al. XLS-R: Self-supervised Cross-lingual Speech Representation Learning at Scale. In: *Proceedings of Interspeech 2022*. Incheon, Korea: ISCA, Sep 2022, p. 2278–2282. Available at: http://dx.doi.org/10.21437/interspeech.2022-143.

[4] Baevski, A.; Zhou, H.; Mohamed, A. and Auli, M. *Wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations*. 2020. Available at: https://arxiv.org/abs/2006.11477.

[5] Bagby, T.; Rao, K. and Sim, K. Efficient Implementation of Recurrent Neural Network Transducer in Tensorflow. In:. December 2018, p. 506–512.

[6] Bahdanau, D.; Cho, K. and Bengio, Y. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014.

[7] Bai, J.; Bai, S.; Chu, Y.; Cui, Z.; Dang, K. et al. *Qwen Technical Report*. 2023.

[8] Basak, S.; Agrawal, H.; Jena, S.; Gite, S.; Bachute, M. et al. Challenges and Limitations in Speech Recognition Technology: A Critical Review of Speech Signal Processing Algorithms, Tools and Systems. *Computer Modeling in Engineering & Sciences*, october 2022, vol. 135, p. 1–37.

[9] Bengio, S. and Heigold, G. Word Embeddings for Speech Recognition. In: *Proceedings of the 15th Conference of the International Speech Communication Association, Interspeech*. 2014.

[10] Bishop, C. Pattern Recognition and Machine Learning. In:. January 2006, vol. 16, p. 140–155.

[11] Bojanowski, P.; Grave, E.; Joulin, A. and Mikolov, T. Enriching Word Vectors with Subword Information. *Transactions of the Association for*

Computational Linguistics. MIT Press, december 2017, vol. 5, p. 135–146. ISSN 2307-387X. Available at: http://dx.doi.org/10.1162/tacl_a_00051.

[12] Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. et al. *Language Models are Few-Shot Learners*. 2020. Available at: https://arxiv.org/abs/2005.14165.

[13] Chan, W.; Jaitly, N.; Le, Q. and Vinyals, O. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, March 2016. Available at: http://dx.doi.org/10.1109/ICASSP.2016.7472621.

[14] Chavan, K. and Gawande, U. Speech recognition in noisy environment, issues and challenges: A review. In: *2015 International Conference on Soft-Computing and Networks Security (ICSNS)*. 2015, p. 1–5.

[15] Cheng, Y.; Zhang, Y.; Johnson, M.; Macherey, W. and Bapna, A. Mu2SLAM: multitask, multilingual speech and language models. In: *Proceedings of the 40th International Conference on Machine Learning*. JMLR.org, 2023. ICML'23.

[16] Cho, K.; Merrienboer, B. van; Gulcehre, C.; Bahdanau, D.; Bougares, F. et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. Available at: https://arxiv.org/abs/1406.1078.

[17] Chorowski, J.; Bahdanau, D.; Serdyuk, D.; Cho, K. and Bengio, Y. *Attention-Based Models for Speech Recognition*. 2015.

[18] Chung, J.; Gulcehre, C.; Cho, K. and Bengio, Y. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014.

[19] Chung, J.; Gulcehre, C.; Cho, K. and Bengio, Y. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. Available at: https://arxiv.org/abs/1412.3555.

[20] Corbetta, M. and Shulman, G. L. Control of goal-directed and stimulus-driven attention in the brain. *Nature Reviews Neuroscience*, march 2002, vol. 3, no. 3, p. 201–215.

[21] Das, A.; Li, J.; Ye, G.; Zhao, R. and Gong, Y. Advancing Acoustic-to-Word CTC Model With Attention and Mixed-Units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. Institute of Electrical and Electronics Engineers (IEEE), december 2019, vol. 27, no. 12, p. 1880–1892. ISSN 2329-9304. Available at: http://dx.doi.org/10.1109/TASLP.2019.2933325.

[22] Dettmers, T.; Pagnoni, A.; Holtzman, A. and Zettlemoyer, L. *QLoRA: Efficient Finetuning of Quantized LLMs*. 2023.

[23] Devlin, J.; Chang, M.-W.; Lee, K. and Toutanova, K. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018.

[24] Devlin, J.; Chang, M.-W.; Lee, K. and Toutanova, K. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. Available at: https://arxiv.org/abs/1810.04805.

[25] Ding, N.; Qin, Y.; Yang, G.; Wei, F.; Yang, Z. et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*. Nature Publishing Group UK London, 2023, vol. 5, no. 3, p. 220–235.

[26] Dong, L.; Wang, F. and Xu, B. Self-attention Aligner: A Latency-control End-to-end Model for ASR Using Self-attention Network and Chunk-hopping. In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, May 2019, p. 5656–5660. Available at: http://dx.doi.org/10.1109/ICASSP.2019.8682954.

[27] Dong, L.; Xu, S. and Xu, B. Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, p. 5884–5888.

[28] Edalati, A.; Tahaei, M.; Kobyzev, I.; Nia, V. P.; Clark, J. J. et al. *KronA: Parameter Efficient Tuning with Kronecker Adapter*. 2022.

[29] Face, H. *Transformers Library - SpeechEncoderDecoder Documentation*. 2025. Available at: https://huggingface.co/docs/transformers/main/en/model_doc/speech-encoder-decoder. Accessed: 2025-01-09.

[30] Fu, Z.; Yang, H.; So, A. M.-C.; Lam, W.; Bing, L. et al. *On the Effectiveness of Parameter-Efficient Fine-Tuning*. 2022.

[31] Fukushima, K. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1988, vol. 1, no. 2, p. 119–130. ISSN 0893-6080. Available at: https://www.sciencedirect.com/science/article/pii/0893608088900147.

[32] Gales, M. and Young, S. 2008.

[33] Ghai, W. and Singh, N. Literature review on automatic speech recognition. *International Journal of Computer Applications*. Citeseer, 2012, vol. 41, no. 8.

[34] Giles, C. L.; Kuhn, G. M. and Williams, R. J. Dynamic recurrent neural networks: Theory and applications. *IEEE Transactions on Neural Networks*. IEEE, 1994, vol. 5, no. 2, p. 153–156.

[35] Goodfellow, I.; Bengio, Y. and Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[36] Grattafiori, A.; Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A. et al. *The Llama 3 Herd of Models*. 2024. Available at: https://arxiv.org/abs/2407.21783.

[37] Graves, A. *Sequence Transduction with Recurrent Neural Networks*. 2012.

[38] Graves, A.; Fernández, S.; Gomez, F. and Schmidhuber, J. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: *Proceedings of the 23rd international conference on Machine learning*. 2006, p. 369–376.

[39] Graves, A. and Jaitly, N. Towards end-to-end speech recognition with recurrent neural networks. In: PMLR. *International conference on machine learning*. 2014, p. 1764–1772.

[40] Gulati, A.; Qin, J.; Chiu, C.-C.; Parmar, N.; Zhang, Y. et al. Conformer: Convolution-augmented Transformer for Speech Recognition. In: *Proceedings of the Annual Conference of the International Speech Communication Association (Interspeech 2020)*. ISCA, October 2020, p. 5036–5040. Available at: https://doi.org/10.21437/Interspeech.2020-3015.

[41] Guo, D.; Rush, A. and Kim, Y. Parameter-Efficient Transfer Learning with Diff Pruning. In: Zong, C.; Xia, F.; Li, W. and Navigli, R., ed. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, August 2021, p. 4884–4896. Available at: https://aclanthology.org/2021.acl-long.378/.

[42] Ha, D.; Dai, A. and Le, Q. V. *HyperNetworks*. 2016.

[43] Hannun, A.; Case, C.; Casper, J.; Catanzaro, B.; Diamos, G. et al. *Deep Speech: Scaling up end-to-end speech recognition*. 2014.

[44] He, J.; Zhou, C.; Ma, X.; Berg Kirkpatrick, T. and Neubig, G. *Towards a Unified View of Parameter-Efficient Transfer Learning*. 2021.

[45] He, Y.; Sainath, T. N.; Prabhavalkar, R.; McGraw, I.; Alvarez, R. et al. Streaming End-to-end Speech Recognition for Mobile Devices. In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, May 2019. Available at: http://dx.doi.org/10.1109/ICASSP.2019.8682336.

[46] He, Y.; Zheng, H. S.; Tay, Y.; Gupta, J.; Du, Y. et al. *HyperPrompt: Prompt-based Task-Conditioning of Transformers*. 2022.

[47] Hendrycks, D. and Gimpel, K. *Gaussian Error Linear Units (GELUs)*. 2016.

[48] Hinton, G.; Deng, L.; Yu, D.; Dahl, G. E.; Mohamed, A.-r. et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 2012, vol. 29, no. 6, p. 82–97.

[49] Hinton, G. E. and Plaut, D. C. Using Fast Weights to Deblur Old Memories. In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. 1987, vol. 9. Available at: https://escholarship.org/uc/item/0570j1dp.

[50] Hochreiter, S. Long Short-term Memory. *Neural Computation MIT-Press*, 1997.

[51] Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; Laroussilhe, Q. de et al. *Parameter-Efficient Transfer Learning for NLP*. 2019.

[52] Hsu, W.-N.; Bolte, B.; Tsai, Y.-H. H.; Lakhotia, K.; Salakhutdinov, R. et al. HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. Institute of Electrical and Electronics Engineers (IEEE), 2021, vol. 29,

p. 3451–3460. ISSN 2329-9304. Available at: http://dx.doi.org/10.1109/taslp.2021.3122291.

[53] Hu, E. J.; Shen, Y.; Wallis, P.; Allen Zhu, Z.; Li, Y. et al. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021.

[54] Hu, S.; Zhang, Z.; Ding, N.; Wang, Y.; Wang, Y. et al. Sparse Structure Search for Delta Tuning. In: Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K. et al., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2022, vol. 35, p. 9853–9865. Available at: https://proceedings.neurips.cc/paper_files/paper/2022/file/4027fc4573ec9114182d1fef37a6321a-Paper-Conference.pdf.

[55] Huang, G.; Sun, Y.; Liu, Z.; Sedra, D. and Weinberger, K. Q. Deep Networks with Stochastic Depth. In: Leibe, B.; Matas, J.; Sebe, N. and Welling, M., ed. *Computer Vision – ECCV 2016*. Cham: Springer International Publishing, 2016, p. 646–661. ISBN 978-3-319-46493-0.

[56] Huang, Z.; Xu, W. and Yu, K. *Bidirectional LSTM-CRF Models for Sequence Tagging*. 2015.

[57] Itti, L.; Koch, C. and Niebur, E. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1998, vol. 20, no. 11, p. 1254–1259.

[58] Jang, E.; Gu, S. and Poole, B. *Categorical Reparameterization with Gumbel-Softmax*. 2016.

[59] Jiang, A. Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D. S. et al. *Mistral 7B*. 2023.

[60] Kahn, J.; Riviere, M.; Zheng, W.; Kharitonov, E.; Xu, Q. et al. Libri-Light: A Benchmark for ASR with Limited or No Supervision. In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, May 2020, p. 7669–7673. Available at: http://dx.doi.org/10.1109/ICASSP40776.2020.9052942.

[61] Kalamkar, D.; Mudigere, D.; Mellempudi, N.; Das, D.; Banerjee, K. et al. *A Study of BFLOAT16 for Deep Learning Training*. 2019.

[62] Kheddar, H.; Hemis, M. and Himeur, Y. Automatic speech recognition using advanced deep learning approaches: A survey. *Information Fusion*, 2024, vol. 109, p. 102422. ISSN 1566-2535. Available at: https://www.sciencedirect.com/science/article/pii/S1566253524002008.

[63] Kitaev, N.; Kaiser, Ł. and Levskaya, A. Reformer: The efficient transformer. *ArXiv preprint arXiv:2001.04451*, 2020.

[64] Krizhevsky, A.; Sutskever, I. and Hinton, G. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems*, january 2012, vol. 25.

[65] Kudo, T. *Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates*. 2018. Available at: https://arxiv.org/abs/1804.10959.

[66] Kudo, T. and Richardson, J. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 2018. Available at: http://dx.doi.org/10.18653/v1/D18-2012.

[67] Latif, S.; Zaidi, A.; Cuayahuitl, H.; Shamshad, F.; Shoukat, M. et al. *Transformers in Speech Processing: A Survey*. 2023. Available at: https://arxiv.org/abs/2303.11607.

[68] Lester, B.; Al Rfou, R. and Constant, N. The Power of Scale for Parameter-Efficient Prompt Tuning. In: Moens, M.-F.; Huang, X.; Specia, L. and Yih, S. W.-t., ed. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, November 2021, p. 3045–3059. Available at: https://aclanthology.org/2021.emnlp-main.243/.

[69] Lewis, M.; Liu, Y.; Goyal, N.; Ghazvininejad, M.; Mohamed, A. et al. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. 2019. Available at: https://arxiv.org/abs/1910.13461.

[70] Li, J. Recent Advances in End-to-End Automatic Speech Recognition. *APSIPA Transactions on Signal and Information Processing*. Now Publishers, 2022, vol. 11, no. 1. ISSN 2048-7703. Available at: http://dx.doi.org/10.1561/116.00000050.

[71] Li, X. L. and Liang, P. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In: Zong, C.; Xia, F.; Li, W. and Navigli, R., ed. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, August 2021, p. 4582–4597. Available at: https://aclanthology.org/2021.acl-long.353/.

[72] Li, Y.; Yu, Y.; Liang, C.; He, P.; Karampatziakis, N. et al. *LoftQ: LoRA-Fine-Tuning-Aware Quantization for Large Language Models*. 2023.

[73] Lialin, V.; Deshpande, V.; Yao, X. and Rumshisky, A. *Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning*. 2023.

[74] Lin, Z.; Madotto, A. and Fung, P. Exploring Versatile Generative Language Model Via Parameter-Efficient Transfer Learning. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, 2020. Available at: http://dx.doi.org/10.18653/v1/2020.findings-emnlp.41.

[75] Liu, H.; Tam, D.; Muqeeth, M.; Mohta, J.; Huang, T. et al. *Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning*. 2022.

[76] Liu, S.-Y.; Wang, C.-Y.; Yin, H.; Molchanov, P.; Wang, Y.-C. F. et al. *DoRA: Weight-Decomposed Low-Rank Adaptation*. 2024.

[77] Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M. et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019.

[78] Loshchilov, I. and Hutter, F. *Decoupled Weight Decay Regularization*. 2019. Available at: https://arxiv.org/abs/1711.05101.

[79] Lou, C.; Jia, Z.; Zheng, Z. and Tu, K. *Sparser is Faster and Less is More: Efficient Sparse Attention for Long-Range Transformers*. 2024.

[80] Mao, Y.; Mathias, L.; Hou, R.; Almahairi, A.; Ma, H. et al. *UniPELT: A Unified Framework for Parameter-Efficient Language Model Tuning*. 2021.

[81] Martinez, J. *What is the OCR Accuracy and How it Can be Improved* https://www.docuclipper.com/blog/ocr-accuracy/. January 2025. Date: January 6, 2025. Viewed 24.4.2025.

[82] Mehrish, A.; Majumder, N.; Bharadwaj, R.; Mihalcea, R. and Poria, S. A review of deep learning techniques for speech processing. *Information Fusion*. Elsevier BV, november 2023, vol. 99, p. 101869. ISSN 1566-2535. Available at: http://dx.doi.org/10.1016/j.inffus.2023.101869.

[83] Miao, Y.; Gowayyed, M. and Metze, F. EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding. In: *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, December 2015, p. 167–174. Available at: http://dx.doi.org/10.1109/ASRU.2015.7404790.

[84] Mikolov, T.; Chen, K.; Corrado, G. and Dean, J. *Efficient Estimation of Word Representations in Vector Space*. 2013. Available at: https://arxiv.org/abs/1301.3781.

[85] OpenAI; :; Hurst, A.; Lerer, A.; Goucher, A. P. et al. *GPT-4o System Card*. 2024.

[86] OpenAI; :; Jaech, A.; Kalai, A.; Lerer, A. et al. *OpenAI o1 System Card*. 2024.

[87] OpenAI; Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L. et al. *GPT-4 Technical Report*. 2023.

[88] OpenAI and Face, H. *Model card for openai/whisper-large-v3* https://huggingface.co/openai/whisper-large-v3. 2023. Accessed: 1.5.2025.

[89] O'Shaughnessy, D. Linear predictive coding. *IEEE potentials*. IEEE, 1988, vol. 7, no. 1, p. 29–32.

[90] Panayotov, V.; Chen, G.; Povey, D. and Khudanpur, S. Librispeech: An ASR corpus based on public domain audio books. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, p. 5206–5210.

[91] Park, D. S.; Chan, W.; Zhang, Y.; Chiu, C.-C.; Zoph, B. et al. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. In: *Proc. Interspeech*. ISCA, September 2019. Available at: http://dx.doi.org/10.21437/Interspeech.2019-2680.

[92] Patil, R. and Gudivada, V. A Review of Current Trends, Techniques, and Challenges in Large Language Models (LLMs). *Applied Sciences*, 2024, vol. 14, no. 5. ISSN 2076-3417. Available at: https://www.mdpi.com/2076-3417/14/5/2074.

[93] Peng, Y.; Dalmia, S.; Lane, I. and Watanabe, S. *Branchformer: Parallel MLP-Attention Architectures to Capture Local and Global Context for Speech Recognition and Understanding*. 2022.

[94] Pennington, J.; Socher, R. and Manning, C. GloVe: Global Vectors for Word Representation. In: Moschitti, A.; Pang, B. and Daelemans, W., ed. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, October 2014, p. 1532–1543. Available at: https://aclanthology.org/D14-1162.

[95] Pfeiffer, J.; Kamath, A.; Rücklé, A.; Cho, K. and Gurevych, I. AdapterFusion: Non-Destructive Task Composition for Transfer Learning. In: Merlo, P.; Tiedemann, J. and Tsarfaty, R., ed. *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Online: Association for Computational Linguistics, April 2021, p. 487–503. Available at: https://aclanthology.org/2021.eacl-main.39/.

[96] Platen, P. von. *Leveraging Pre-trained Language Model Checkpoints for Encoder-Decoder Models*. November 2020. Available at: https://huggingface.co/blog/warm-starting-encoder-decoder. Accessed: 2025-01-09.

[97] Prabhavalkar, R.; Hori, T.; Sainath, T. N.; Schlüter, R. and Watanabe, S. End-to-End Speech Recognition: A Survey. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. Institute of Electrical and Electronics Engineers (IEEE), 2024, vol. 32, p. 325–351. ISSN 2329-9304. Available at: http://dx.doi.org/10.1109/TASLP.2023.3328283.

[98] Pratap, V.; Tjandra, A.; Shi, B.; Tomasello, P.; Babu, A. et al. Scaling speech technology to 1,000+ languages. *J. Mach. Learn. Res.* JMLR.org, january 2024, vol. 25, no. 1. ISSN 1532-4435.

[99] Press, O.; Smith, N. A. and Lewis, M. *Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation*. 2022. Available at: https://arxiv.org/abs/2108.12409.

[100] Press, O. and Wolf, L. Using the Output Embedding to Improve Language Models. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, 2017, p. 157–163. Available at: http://dx.doi.org/10.18653/V1/E17-2025.

[101] Provilkov, I.; Emelianenko, D. and Voita, E. BPE-Dropout: Simple and Effective Subword Regularization. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020. Available at: http://dx.doi.org/10.18653/v1/2020.acl-main.170.

[102] Puvvada, K. C.; Żelasko, P.; Huang, H.; Hrinchuk, O.; Koluguri, N. R. et al. *Less is More: Accurate Speech Recognition & Translation without Web-Scale Data*. 2024.

[103] Radford, A.; Kim, J. W.; Xu, T.; Brockman, G.; McLeavey, C. et al. *Robust Speech Recognition via Large-Scale Weak Supervision*. 2022.

[104] Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I. et al. Improving language understanding by generative pre-training. San Francisco, CA, USA, 2018.

[105] Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D. et al. Language models are unsupervised multitask learners.

[106] Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S. et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2019.

[107] Rekesh, D.; Koluguri, N. R.; Kriman, S.; Majumdar, S.; Noroozi, V. et al. Fast Conformer With Linearly Scalable Attention For Efficient Speech Recognition. In: *2023 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, December 2023, p. 1–8. Available at: http://dx.doi.org/10.1109/ASRU57964.2023.10389701.

[108] Robinson, T.; Hochberg, M. and Renals, S. The use of recurrent neural networks in continuous speech recognition. In: *Automatic Speech and Speaker Recognition: Advanced Topics*. Springer, 1996, p. 233–258.

[109] Rocha, R. C. O.; Petoumenos, P.; Wang, Z.; Cole, M. and Leather, H. Function Merging by Sequence Alignment. In: *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 2019, p. 149–163.

[110] Rücklé, A.; Geigle, G.; Glockner, M.; Beck, T.; Pfeiffer, J. et al. AdapterDrop: On the Efficiency of Adapters in Transformers. In: Moens, M.-F.; Huang, X.; Specia, L. and Yih, S. W.-t., ed. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, November 2021, p. 7930–7946. Available at: https://aclanthology.org/2021.emnlp-main.626/.

[111] Rust, P.; Pfeiffer, J.; Vulić, I.; Ruder, S. and Gurevych, I. How Good is Your Tokenizer? On the Monolingual Performance of Multilingual Language Models. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, 2021. Available at: http://dx.doi.org/10.18653/v1/2021.acl-long.243.

[112] Sak, H.; Senior, A. W. and Beaufays, F. Long short-term memory recurrent neural network architectures for large scale acoustic modeling, 2014.

[113] Schlag, I.; Irie, K. and Schmidhuber, J. *Linear Transformers Are Secretly Fast Weight Programmers*. 2021.

[114] Schmidhuber, J. Learning to Control Fast-Weight Memories: An Alternative to Dynamic Recurrent Networks. *Neural Computation*, january 1992, vol. 4, no. 1, p. 131–139. ISSN 0899-7667. Available at: https://doi.org/10.1162/neco.1992.4.1.131.

[115] Sennrich, R.; Haddow, B. and Birch, A. Neural Machine Translation of Rare Words with Subword Units. In: Erk, K. and Smith, N. A., ed. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, August 2016, p. 1715–1725. Available at: https://aclanthology.org/P16-1162.

[116] Shi, Y.; Wang, Y.; Wu, C.; Yeh, C.-F.; Chan, J. et al. Emformer: Efficient Memory Transformer Based Acoustic Model for Low Latency Streaming Speech Recognition. In: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, June 2021, p. 6783–6787. Available at: http://dx.doi.org/10.1109/ICASSP39728.2021.9414560.

[117] Soltau, H.; Liao, H. and Sak, H. Neural Speech Recognizer: Acoustic-to-Word LSTM Model for Large Vocabulary Speech Recognition. In: *Proceedings of the 18th Annual Conference of the International Speech Communication Association (INTERSPEECH 2017)*. Stockholm, Sweden: ISCA, August 2017, p. 3707–3711. Available at: https://www.isca-archive.org/interspeech_2017/soltau17_interspeech.pdf.

[118] Srivastav, V.; Majumdar, S.; Koluguri, N.; Moumen, A.; Gandhi, S. et al. *Open Automatic Speech Recognition Leaderboard* https://huggingface.co/spaces/hf-audio/open_asr_leaderboard. Hugging Face, 2023.

[119] Su, J.; Lu, Y.; Pan, S.; Murtadha, A.; Wen, B. et al. *RoFormer: Enhanced Transformer with Rotary Position Embedding*. 2023. Available at: https://arxiv.org/abs/2104.09864.

[120] Sung, Y.-L.; Cho, J. and Bansal, M. *LST: Ladder Side-Tuning for Parameter and Memory Efficient Transfer Learning*. 2022.

[121] Sung, Y.-L.; Nair, V. and Raffel, C. *Training Neural Networks with Fixed Sparse Masks*. 2021.

[122] Sutskever, I.; Vinyals, O. and Le, Q. V. *Sequence to Sequence Learning with Neural Networks*. 2014. Available at: https://arxiv.org/abs/1409.3215.

[123] Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A. et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023.

[124] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L. et al. *Attention Is All You Need*. 2017.

[125] Wang, C.; Rivière, M.; Lee, A.; Wu, A.; Talnikar, C. et al. *VoxPopuli: A Large-Scale Multilingual Speech Corpus for Representation Learning, Semi-Supervised Learning and Interpretation*. 2021. Available at: https://arxiv.org/abs/2101.00390.

[126] Warner, B.; Chaffin, A.; Clavié, B.; Weller, O.; Hallström, O. et al. *Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference*. 2024.

[127] Watanabe, S.; Hori, T.; Karita, S.; Hayashi, T.; Nishitoba, J. et al. ESPnet: End-to-End Speech Processing Toolkit. In: *Proceedings of Interspeech*. 2018, p. 2207–2211. Available at: http://dx.doi.org/10.21437/Interspeech.2018-1456.

[128] Wu, Y.; Schuster, M.; Chen, Z.; Le, Q. V.; Norouzi, M. et al. *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. 2016. Available at: https://arxiv.org/abs/1609.08144.

[129] Xiong, R.; Yang, Y.; He, D.; Zheng, K.; Zheng, S. et al. *On Layer Normalization in the Transformer Architecture*. 2020. Available at: https://arxiv.org/abs/2002.04745.

[130] Xu, H.; Jia, F.; Majumdar, S.; Huang, H.; Watanabe, S. et al. *Efficient Sequence Transduction by Jointly Predicting Tokens and Durations*. 2023.

[131] Xu, K.; Ba, J.; Kiros, R.; Cho, K.; Courville, A. et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In: Bach, F. and Blei, D., ed. *Proceedings of the 32nd International Conference on Machine Learning*. Lille, France: PMLR, 07–09 Jul 2015, vol. 37, p. 2048–2057. Proceedings of Machine Learning Research. Available at: https://proceedings.mlr.press/v37/xuc15.html.

[132] Xu, L.; Xie, H.; Qin, S.-Z. J.; Tao, X. and Wang, F. L. *Parameter-Efficient Fine-Tuning Methods for Pretrained Language Models: A Critical Review and Assessment*. 2023.

[133] Xu, Y.; Li, X.; Yuan, H.; Yang, Y. and Zhang, L. Multi-Task Learning With Multi-Query Transformer for Dense Prediction. *IEEE Transactions on Circuits and Systems for Video Technology*, 2024, vol. 34, no. 2, p. 1228–1240.

[134] Xu, Y.; Xie, L.; Gu, X.; Chen, X.; Chang, H. et al. *QA-LoRA: Quantization-Aware Low-Rank Adaptation of Large Language Models*. 2023.

[135] Yang, A. X.; Robeyns, M.; Wang, X. and Aitchison, L. *Bayesian Low-rank Adaptation for Large Language Models*. 2023.

[136] Yang, F.; Balakrishnan, S. and Wainwright, M. J. Statistical and computational guarantees for the Baum-Welch algorithm. In: *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, September 2015, p. 658–665. Available at: http://dx.doi.org/10.1109/ALLERTON.2015.7447067.

[137] Yu, D. and Deng, L. *Automatic Speech Recognition: A Deep Learning Approach*. Springer London, 2014. Signals and Communication Technology. ISBN 9781447157793. Available at: https://books.google.cz/books?id=rUBTBQAAQBAJ.

[138] Zaheer, M.; Guruganesh, G.; Dubey, K. A.; Ainslie, J.; Alberti, C. et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 2020, vol. 33, p. 17283–17297.

[139] Zaken, E. B.; Ravfogel, S. and Goldberg, Y. *BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models*. 2021.

[140] Zeyer, A.; Irie, K.; Schlüter, R. and Ney, H. Improved Training of End-to-End Attention Models for Speech Recognition. In: *Proceedings of the Annual Conference of the International Speech Communication Association (Interspeech 2018)*. ISCA, September 2018. Available at: https://doi.org/10.21437/Interspeech.2018-1616.

[141] Zhang, B. and Sennrich, R. *Root Mean Square Layer Normalization*. 2019. Available at: https://arxiv.org/abs/1910.07467.

[142] Zhang, L.; Zhang, L.; Shi, S.; Chu, X. and Li, B. *LoRA-FA: Memory-efficient Low-rank Adaptation for Large Language Models Fine-tuning*. 2023.

[143] Zhang, M.; Chen, H.; Shen, C.; Yang, Z.; Ou, L. et al. LoRAPrune: Structured Pruning Meets Low-Rank Parameter-Efficient Fine-Tuning. In: *Findings of the Association for Computational Linguistics ACL 2024*. Association for Computational Linguistics, 2024, p. 3013–3026. Available at: http://dx.doi.org/10.18653/v1/2024.findings-acl.178.

[144] Zhang, Q.; Chen, M.; Bukharin, A.; Karampatziakis, N.; He, P. et al. *AdaLoRA: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning*. 2023.

[145] Zhao, M.; Lin, T.; Mi, F.; Jaggi, M. and Schütze, H. Masking as an Efficient Alternative to Finetuning for Pretrained Language Models. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2020. Available at: http://dx.doi.org/10.18653/v1/2020.emnlp-main.174.

[146] Zhou, H.; Wan, X.; Vulić, I. and Korhonen, A. <scp>AutoPEFT</scp>: Automatic Configuration Search for Parameter-Efficient Fine-Tuning. *Transactions of the Association for Computational Linguistics*. MIT Press, 2024, vol. 12, p. 525–542. ISSN 2307-387X. Available at: http://dx.doi.org/10.1162/tacl_a_00662.

[147] Zusag, M.; Wagner, L. and Thallinger, B. CrisperWhisper: Accurate Timestamps on Verbatim Speech Transcriptions. In: *Proceedings of Interspeech 2024*. ISCA, September 2024, p. 1265–1269. Available at: https://doi.org/10.21437/interspeech.2024-731.

Appendix A

Timing Measurement Methodology

The timing measurements presented in Table 5.9 were obtained using the following procedure, implemented in PyTorch (version 2.5.1) and the model implemented with the Hugging Face Transformers (version 4.46.3) library. The goal was to isolate the forward and backward pass time for a single 10-second audio sample, excluding any data loading or other overhead.

```
# Model in training mode
model.train()

# Warm-up (discarded iterations)
for _ in range(NUM_WARMUP):
    outputs = model(input)
    loss = outputs.loss
    loss.backward()
    model.zero_grad()

# Measurement iterations
for _ in range(NUM_ITERATIONS):
    start_event = record_start_time() #e.g., torch.cuda.Event

    outputs = model(input)
    loss = outputs.loss
    loss.backward()

    end_event = record_end_time() #e.g., torch.cuda.Event
    synchronize() # Ensure GPU operations are complete, e.g. torch.cuda.synchronize
    time_taken = calculate_elapsed_time(start_event, end_event)
    store(time_taken)
    model.zero_grad()

# Calculate average and standard deviation of stored times.
average_time = average(stored_times)
std_deviation = std_dev(stored_times)
```

Listing A.1: Code used for Time Measurements

Appendix B

Contents of the External Attachment

The external attachment accompanying this thesis contains the source code, scripts, experimental configurations, and other relevant files used for the research. The main structure is as follows:

- `bart/`: Contains scripts related to the pre-training or continuous pre-training experiments for the BART decoder component.
- `config.py`: A configuration file.
- `ctc/`: Includes files and scripts for experiments involving Connectionist Temporal Classification (CTC) based ASR models.
- `graphs/`: Stores generated plots, figures, CSV data used for creating visualizations, and Jupyter notebooks for analysis and playground experiments.
- `poetry.lock`: The dependency lock file generated by the Poetry packaging tool.
- `pyproject.toml`: The main configuration file for the Python project, defining dependencies, project metadata, and build system settings managed by Poetry.
- `README.md`: The primary markdown file the project.
- `run_libri.sh`: A shell script for launching experiments specifically on the LibriSpeech dataset.
- `run_scripts/`: A directory containing a collection of shell scripts used to execute various training and evaluation runs for different model configurations and datasets.
- `run_speech_recognition_seq2seq.py`: The core Python script used for fine-tuning the sequence-to-sequence ASR models with the Hugging Face Transformers library.
- `run_voxpopuli.sh`: A shell script for launching experiments specifically on the VoxPopuli dataset.
- `scripts/`: Contains various utility and helper shell scripts.

- `src/`: The main source code directory for the project, housing custom Python modules such as specialized trainers, data collators, argument dataclasses, and model creation utilities.
- `thesis/`: Contains all Latex source files.
- `transducer/`: Includes files, scripts, and notebooks related to experiments or explorations with RNN Transducer ASR models.
- `.env.example`: Example environment variable file.