MtechThesis

by Pooja Verma

Submission date: 23-Jun-2026 03:46PM (UTC+0530)
Submission ID: 2988295002
File name: Thesis_24AFI22_plag.pdf (2.18M)
Word count: 8193
Character count: 48163

ABSTRACT

The recent developments in Artificial Intelligence, Deep Learning, and Natural Language Processing have revolutionized the areas of speech synthesis and language generation. Contemporary neural Text-to-Speech (TTS) models are now able to synthesize highly human-sounding speech that is characterized by improved intelligibility, pronunciations, and prosody. At the same time, there has been great success in using Large Language Models (LLMs) for tasks like conversations, text generation, summarization, and reasoning.

The current research work comprises a detailed comparative analysis of sophisticated Neural TTS systems alongside inference optimization methods for Large Language Models. This analysis is concentrated on three contemporary TTS models, namely Tacotron, FastSpeech 2, and MatchaTTS. These models are comparatively studied regarding their performance parameters such as synthesizer quality, training difficulty, inference speed, computation, and real-world implementation feasibility.

In addition to speech synthesis, this study also considers advanced inference acceleration and optimization methods for autoregressive Large Language Models (LLMs). The considered methods include speculative decoding, LoRA adapter, and Multi-LoRA adapter, which were implemented on an Intel AI accelerator with Intel Gaudi B70 accelerator and the Llama 3.1 8B model. The following three speculative decoding strategies have been considered in order to speed up the inference process based on generation of several token candidates through draft/retrieval procedures and subsequent validation by the target LLM: EAGLE-3, N-gram prompt lookup decoding, and suffix array retrieval decoding. Moreover, the LoRA adapter and Multi-LoRA adapter methods have been considered for efficient parameter tuning and multitask fine-tuning, respectively. The effectiveness of the presented inference strategies has been assessed by using the key inference parameters: time to first token (TTFT), time per output token (TPOT), and throughput.

Chapter 1

INTRODUCTION

1.1 Overview

Artificial Intelligence and Deep Learning have revolutionized the speech and nat-ural language processing systems in recent years. Text-to-Speech (TTS) systems have evolved from rule-based and concatenative methods to advanced neural architectures ca-pable of producing highly natural and expressive speech. Similarly, Large Language Mod-els (LLMs) such as LLaMA, GPT, and Transformer-based architectures have significantly improved language understanding and generation capabilities.

Despite these advancements, computational efficiency remains a major challenge. Traditional autoregressive TTS systems produce high-quality speech but suffer from slow inference. Non-autoregressive architectures attempt to solve this issue through parallel generation mechanisms. Likewise, LLM inference is inherently sequential, leading to increased latency during token generation, especially on CPU-based hardware systems.

This thesis focuses on two major research areas:

1. Comparative analysis of modern neural TTS systems.
2. Performance evaluation of speculative decoding methods for accelerating LLM in-ference.

Three modern TTS models — Tacotron , FastSpeech 2, and MatchaTTS — are an-alyzed in terms of architecture, training methodology, speech quality, robustness, and inference speed. Additionally, speculative decoding techniques including EAGLE3, N-gram Prompt Lookup Decoding, LoRA and multi-LoRA Adapters are evaluated using LLaMA 3.1-8B on Intel Xeon architecture.
The study aims to identify the strengths and limitations of each model and provide insights into future efficient AI deployment strategies.

1.2 Motivation

There have been many developments in the sphere of Artificial Intelligence during the past decade, with the technologies in the spheres of speech synthesis and natural language generation being particularly well-developed. Such technologies as virtual assistants, conversational chatbots, automated customer service, audiobook reading, assistive technologies for visually impaired people, intelligent healthcare, and multi-lingual communication platforms become a part of daily activities for many people. The basis for these innovations are Text-to-Speech (TTS) models and Large Language Models (LLM), which strive to mimic natural human speech and writing.

One of the key driving forces behind this project is the issue of balancing speech quality and efficiency in inference when building neural Text-to-Speech systems. Initial models of Neural TTS like Tacotron and Tacotron 2 made incredible strides in speech quality through the use of encoder-decoder structure and attention mechanism. These models provide amazing speech quality where pronunciation and prosody are accurately modeled and predicted. The disadvantage of these models lies in the autoregressive approach, which makes them very inefficient, as speech frames have to be generated sequentially. Therefore, inference takes an incredible amount of time, especially in long audio files. This presents an important problem for real-time applications of these models, such as voice assistants, live broadcast announcements, telecommunication systems, and conversational assistants. Besides that, autoregressive attention can sometimes generate speech misalignment.

The reason why FastSpeech 2 was considered a breakthrough model is because it introduced a non-auto-regressive architecture to TTS. Rather than generating sound wave frames one after another, FastSpeech 2 predicts durations, pitches, and energy values independently and uses those information to produce the mel-spectrogram in parallel. Thus, the inference process of the model becomes incredibly fast and stable. Motivation of this research comes from trying to investigate the architectural developments of TTS systems.

Another significant reason behind this study comes from the quick emergence of Large Language Models like LLaMA, GPT, PaLM, and Transformer chatbot models. Such models have shown outstanding capabilities in tasks like text generation, reasoning, summarization, and dialogue models. Nevertheless, their practical deployment is quite costly because of the autoregressive generation approach. While using inference for prediction, LLMs generate only one token at a time, and each subsequent token is dependent on previously generated tokens. As model capacity scales up to billions of parameters, the latency and the memory bandwidth requirement during inference become exceedingly high. In many enterprise-level scenarios, cloud computing platforms, and edge devices, GPU acceleration is often unavailable due to financial restrictions.

Recently, speculative decoding and adapter optimization have proven to be effective approaches to speeding up Large Language Model (LLM) inference. In our work, the following LLM optimization approaches have been explored: EAGLE-3, N-gram Prompt Lookup Decoding, LoRA Adapter, and Multi-LoRA Adapter on Llama 3.1 8B model powered by an Intel Gaudi B70 AI accelerator. The approach of speculative decoding generates several token hypotheses based on draft approaches and verifies them using the target model to skip the expensive autoregressive forward pass step and improve inference speed. Additionally, we have investigated how to adapt the model parameters and perform multi-task learning with low additional computational complexity using LoRA and Multi-LoRA adapter methods. Evaluation was performed based on such performance metrics as Time to First Token, Time per Output Token, and Throughput to examine the effect of optimization approaches on low-latency and high-throughput inference.

This paper's overall objective lies in making progress toward the creation of large-scale, high-performance, and quality-assured artificial intelligence applications that can be deployed in the real world. This paper makes an effort towards connecting the theory of deep learning with its practical implementation by exploring neural speech synthesis models and inference acceleration approaches. The results obtained can serve as the basis for real-time conversational AI systems, efficient speech assistants, low-resource deployment platforms, and next-generation intelligent communications systems.

1.3 Objective of the Study

The main objectives of this research work are as follows:

1. **To examine architectures of Tacotron 2, FastSpeech 2 and MatchaTTS:**

   This goal is directed toward the understanding of the working mechanism of modern Text to Speech neural networks. The study evaluates the architecture of encoder-decoder, attention, duration prediction, flow matching, and melspectrogram generation modules of these systems.

2. **The aim is to carry out a comparative analysis of Tacotron 2, FastSpeech 2, and MatchaTTS regarding speech quality and inference time:**

   The main purpose of the study is to analyze the proposed models based on MOS, RTF, stability, and efficiency. The analysis will help to find out which neural TTS system is the most efficient.

3. **To study speculation-based decoding methodologies to enhance Large Language Model Inference:**

   The above objective concentrates on the exploration of novel ways of improving the inference process in order to lower the latency of auto-regressive language models. It involves an understanding of speculative decoding and its method of producing multiple tokens ahead of time.

4. **Evaluation of EAGLE3, N-gram Prompt Lookup, and Suffix Array Retrieval Decoding Techniques Using LLaMA 3.1-8B:**

   In this paper, the researchers conduct an experimental evaluation of various speculation decoding techniques with the LLaMA 3.1-8B model on the Intel Xeon CPU platform. Performance parameters like acceptance ratio, throughput, decoding speedup, and latency are measured for performance analysis.

5. **For investigating LoRA Adapter and Multi-LoRA Adapter optimization for efficient deployment of LLMs:**

   In this objective, the focus is on evaluating efficient techniques for fine-tuning and deployment by means of LoRA Adapter and Multi-LoRA Adapters, with B70 Enterprise GPUs being used. The focus here is on evaluating techniques which can

efficiently utilize many adapters, thereby ensuring low memory consumption and reduced processing cost. Throughput, TTFT, TPOT, GPU utilization, and scalability are some important aspects evaluated in this objective.

6. **To investigate the tradeoffs between computational efficiency and performance of modern AI models:**

   This goal seeks to explore the relationship between increases in computational efficiency and its effect on speech quality, expressivity, robustness, and overall model performance. The research underscores the delicate balance between optimizing performance and the quality of output in the design of scalable AI models.

1.4 Scope and Limitations

The research is purposefully designed with a narrow scope. In particular, regarding the TTS model evaluation, the assessment is done using the LJSpeech-1.1 dataset that includes speech data from a single speaker. Because of computational restrictions, there were certain training instances where smaller sample sizes of up to 100 samples were used. Large-sample results were gathered based on literature references.

In the case of the optimization of the LLM model, there were experiments with speculative decoding, LoRA Adapter, and Multi-LoRA Adapter techniques performed on Llama 3.1 8B model utilizing Intel Gaudi B70 AI acceleration system. Evaluation was performed in the context of single-user inference tasks with TTFT, TPOT, and throughput being some of the primary metrics analyzed. Given that large multi-user deployments and dynamic batch processing systems were out of the question, actual performance gains could be different from those shown in the research. There was no evaluation of advanced speculative decoding (such as EAGLE-3 and N-gram Prompt Lookup Decoding) and other adapter orchestration techniques that would require more advanced models.

Chapter 2

LITERATURE REVIEW

2.1 Early Neural Text-to-Speech Systemss

Neural TTS systems have been developed with the advent of the WaveNet model that creates speech samples one by one with the use of deep learning. The model can generate extremely natural and lifelike speech as compared to other conventional parametric models of TTS. But due to the sequential generation of audio samples, the method is costly and extremely slow.

To overcome this issue, Tacotron was developed. Unlike the former model, Tacotron translates input text into mel-spectrograms and then converts the latter to speech using the vocoder, like WaveNet or Griffin-Lim. Using the sequence-to-sequence encoder-decoder model with an attention mechanism, Tacotron automatically learns the text-speech alignment without relying on linguistic features.

In later stages, Tacotron 2 was able to produce better quality speech through the use of an advanced acoustic model with a WaveNet vocoder, generating very human-like quality speech synthesis results. Nevertheless, Tacotron and Tacotron 2 were still based on the autoregressive method of generation, producing one speech frame at a time in the generation process, causing significant latency during inference.

This encouraged the creation of fast and efficient non-autoregressive TTS systems like FastSpeech 2 and Matcha-TTS. This thesis examines several Neural TTS architectures like FastSpeech 2 and Matcha-TTS, implemented using LJSpeech datasets on Google Colab.

2.2 Parallel and Non-Autoregressive TTS

The rise in the requirement of fast speech synthesis created the need for parallel and non-autoregressive text-to-speech (TTS) models. As opposed to autoregressive models, where each speech segment is generated sequentially, non-autoregressive models generate the whole mel-spectrogram in a parallel manner, thus reducing the inference time.

In the Glow-TTS paper, normalizing flow and monotonic alignment search were used for the efficient generation of mel-spectrograms within one forward pass. Likewise, VITS employed variational autoencoders, conditional flow, and adversarial learning for the direct generation of high-quality speech waveforms.

A significant advancement in the development of non-autoregressive TTS was the introduction of FastSpeech. The model uses a parallel acoustic network for synthesizing audio. Unlike in other models where the mel-spectrogram frames are produced sequentially, in FastSpeech, a duration predictor is used to extend text encoder embeddings to the needed size, which is then decoded via FFT layers.

Another improvement that followed was the release of FastSpeech 2. In addition to duration predictions, variance predictors are used to predict pitch, energy, and duration values in the FastSpeech 2 model, thus giving it control over prosody. FastSpeech 2 delivers high-quality audio while preserving fast and stable inference, becoming an important model for this work.

2.3 Diffusion and Flow-Based TTS

In the progression of developing diffusion/flow based TTS models, two important things were considered; first, to improve speech quality, and second, to speed up inference efficiency. FastSpeech developed a purely parallel acoustic model using a duration predictor, which expanded the output of the encoder into the length of the mel-spectrogram target. Then, a Feed-Forward Transformer (FFT) block was utilized for feature generation. With a parallel decoding process, FastSpeech significantly reduced the computational time compared to other models, such as Tacotron. Additionally, FastSpeech 2 further improved this design by adding more predictors.

In recent times, Matcha-TTS came up with a flow-matching method that solves the problem of slow generation inherent in TTS systems that use the diffusion model. The model employs Optimal Transport (OT) conditional flow matching, which ensures that the paths taken in moving from one distribution to another are efficient and straighter, making it possible to produce good speech at a faster rate through fewer steps. As such, the speech produced by the model has a better trade-off between speed and quality. The stochastic duration predictor in Matcha-TTS allows the production of varied speaking rhythms, unlike in models with deterministic duration predictors. For this reason, Matcha-TTS becomes a significant improvement in Neural TTS research and an essential model covered in this dissertation using the LJSpeech dataset on Google Colab.

2.4 Speculative Decoding — Origins and Evolution

Speculative decoding was proposed to serve as one of the techniques aimed at boosting the efficiency of the inference process within autoregressive Large Language Models. The general idea of speculative decoding lies in using a relatively simple draft model or even a heuristic method to generate several token predictions at once, whereas the main model confirms them in just one forward pass. Thus, speculative decoding allows reducing the number of expensive sequential passes and boosting the efficiency of inference while preserving the same level of output quality.

Amongst these advancements is the development of EAGLE-3, where speculation on draft generation is done in hidden feature spaces rather than making token predictions directly. The EAGLE model had earlier been made more efficient through lightweight autoregressive heads and dynamic draft trees, while in EAGLE-3, it becomes more powerful due to the incorporation of multi-layer feature fusion from intermediate layers of the target model.

Another light-weight speculative decoding method is N-gram Prompt Lookup Decoding, which relies on recurring patterns in n-grams within the input context to speculate on tokens that come next, without any further training of the model itself. This particular method works very well when the generated text aligns closely with the input context.

Besides the above-mentioned decoding, parameter-efficient fine-tuning models like

LoRA Adapter and Multi-LoRA Adapter are crucial optimization techniques when implementing LLMs. The LoRA technique optimizes the computational complexity by updating just the lower rank matrix rather than adjusting all the model's parameters, while the Multi-LoRA technique is an expansion of the previous technique, which allows for the existence of multiple task-specific adapters inside the same base model.

Within this dissertation, four models called EAGLE-3, N-Gram Prompt Lookup Decoding, LoRA Adapter, and Multi-LoRA Adapter are implemented using the Llama 3.1 8B model and examined in terms of inference latency and throughput.

2.5 LLM Inference on CPU Platforms

Most decoding experiments have been based on GPUs until now; however, CPU and accelerator inference of LLMs is becoming increasingly essential because of constraints related to costs and scale. Intel offers advanced AI software and hardware solutions that can accelerate transformer models based on BF16 and INT8 precisions.

In this thesis, experiments were performed using the Llama 3.1 8B model on the Intel Gaudi B70 AI accelerator system. Inference with Large Language Models is computationally intensive due to the autoregressive decoding process that produces tokens one at a time, causing long latencies and low hardware usage during the creation of individual tokens.

The experiment revolved around benchmarking important metrics associated with performance of inference tasks like TTFT, TPOT, and Throughput. Using the combination of speculative decoding algorithms like EAGLE-3 and N-gram Prompt Lookup Decoding along with LoRA and Multi-LoRA adapter optimization, the research explores ways to enhance the efficiency and scalability of inference tasks on Intel AI hardware platforms.

Chapter 3

Experimental Environment and Dataset

3.1 TTS Experimental Environment

The experiments in implementing the TTS system were done using Google Colab where NVIDIA Tesla T4 GPUs with CUDA support were employed. The training of the models in the experiments involved mixed precision training using FP16 across all models. The ESPnet2 framework was utilized in the implementation and training of the models including the Tacotron 2, FastSpeech 2, and Matcha-TTS models using the LJSpeech corpus.

3.2 The LJSpeech-1.1 Dataset

The experiments on TTS that were done in this thesis involved the LJSpeech-1.1 dataset which was developed by Keith Ito and Linda Johnson in the year 2017. It is considered one of the widely used datasets to benchmark Neural TTS systems. The dataset has around 13,100 short audio clips of utterances that were produced by a female speaker reading texts from public domain books taken from the LibriVox collection.

3.3 Audio Preprocessing Pipeline

All audio recordings were preprocessed to have a sampled rate of 22,050 Hz. Mel-spectogram were extracted at Short Time Fourier Transformation (STFT), where the FFT window was of size 1024 and the overlap was 256, while using a Hann window of length 1024. An 80-bin mel filter bank was then employed to extract relevant features such as frequency content, pronunciation information, and prosody.

Normalization methods were employed for text preprocessing for dealing with numbers, abbreviations, and punctuation marks. For G2P mapping, CMU pronunciation dictionary was utilized, and neural G2P method was used for handling unknown words. Same phonetic

representation was employed in all three models of Tacotron 2, FastSpeech 2, and Matcha-TTS, so that it can be compared fairly based on their performance levels.

3.4 Hyperparameter Configuration

Table 1 summarizes the key hyperparameters used for each TTS model. Despite architectural differences, all models share the 80-channel mel-spectrogram target representation and are trained using Adam-family optimizers. The specific learning rate schedules, encoder and decoder configurations, and vocoder choices reflect the design decisions made in the original papers and are maintained here for reproducibility.

Table I: LJSpeech-1.1 Dataset Properties

Property	Value
Speaker	Linda Johnson (single female, American English)
Total Clips	13,100 short audio utterances
Total Duration	~24 hours of read speech
Sampling Rate	22,050 Hz, 16-bit PCM mono
Avg. Clip Length	~7.1 seconds per utterance
Text Source	Non-fiction public-domain books (LibriVox)
Train / Val / Test	12,500 / 100 / 500 (standard split)
Mel Channels	80 (identical config across all models)
FFT / Hop / Win	1024 / 256 / 1024 samples
Phoneme Tool	CMU Dict + G2P neural fallback
Text Normalization	NLTK: numbers, abbreviations, punctuation

3.5 Speculative Decoding, LoRA and multi-LoRA Adatpters Experimental Environment

Speculative decoding, LoRA Adapter, and Multi-LoRA Adapter tests were performed with the Llama 3.1 8B model on the Intel Gaudi B70 AI accelerator hardware. The testbed was equipped with four accelerator cards, whereas all experiments conducted in this thesis were done on a one-card setup. Specifically, the hardware was configured with 8 processing cores per experiment with numactl tool for NUMA-aware memory management and inference execution. The experiments analyzed various inference acceleration techniques, including the use of EAGLE-3 speculative decoding, N-gram Prompt Lookup Decoding, LoRA Adapter, and Multi-LoRA Adapter.

Chapter 4

Text to Speech Implementation and Results

4.1 Tacotron — End-to-End Attention-Based TTS

4.1.1 Network Architecture

Text Encoder

Input is a series of one-hot coded characters or phonemes. The input characters go through a learned character embedding layer and get transformed into a continuous embedding vector. The next step involves processing the embedding vectors using a CBHG module that combines convolution bank, highway networks, and bidirectional Gated Recurrent Units. The CBHG is the key architectural innovation that makes the encoder work effectively.

The encoding gets the character/phoneme sequences gives input in form of one hot encoded representation. The embeddings of the characters are produced using an embedding layer, after which the character embeddings go through the CBHG model. CBHG is comprised of the following blocks: Convolution Bank: several 1-D convolutions with different filter sizes (k=16). Max Pooling Layers. Highway Network. Bidirectional GRU. The convolution bank helps in capturing the local context information from the phoneme sequences. Whereas the bidirectional GRU helps in capturing the global sequential information from the sequence of inputs. The output of the encoder consists of the sequence of hidden states: H = h1, h2, …, hT

Attention Mechanism

For Tacotron, the attention process is based on the additive Bahdanau attention approach. During speech synthesis at any point in time *t*, the attention component calculates the alignment score using the following formula:

$$e_{t,i} = v^T \tanh(Wq_t + Uh_i + b)$$

where $e_{t,i}$ represents the alignment score calculated between q_t, the decoder hidden state, and h_i, each encoder hidden state.

The attention model should be able to learn a nearly monotonic alignment in the sense that it should traverse the text in a left-to-right fashion while synthesizing without any constraint on monotonicity. This flexibility in the model may help it learn sophisticated prosodic patterns but may also lead to instability in training alignments with potential problems like skipping of words (ignoring certain phonemes) and repetition of words (repeating phonemes).

Decoder

This model utilizes the concept of auto-regression, whereby the mel-spectrogram frames are created progressively, i.e., one by one. The steps involved in decoding include the following:

1. The previous mel-spectrogram frame is inputted to the pre-net, which comprises two feed-forward layers with ReLU activations and a dropout mechanism.
2. The resulting output is then combined with the attentions of contexting.
3. The resultant combination is processed through two recurrent layers of residual GRUs.
4. The next mel-spectrogram frame is predicted by the linear projection layer.

CBHG Post-Processing Network

Following the generation of the mel-spectrogram, CBHG-based post-process network to convert mel-spectrograms frame into single spectrograms that can be used for waveform synthesis. The CBHG post-net consists of the following compo-nents:

1. Convolution banks
2. Highway layers

3. Bidirectional GRU layers

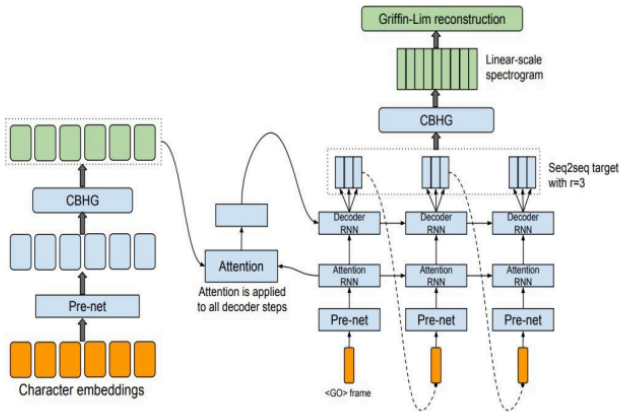


Figure 4.1: Tacotron Architecture

Vocoder: Griffin-Lim and WaveNet

The original Tacotron applies Griffin-Lim phase estimation from linear spectrograms, which is a traditional signal processing technique that repeatedly computes the phase of a magnitude spectrogram. Even though it runs quite quickly, the Griffin-Lim approach is known to add noticeable artifacts to the waveform, including its distinctive "buzziness" effect and a lack of naturalness. The improved Tacotron 2 utilizes a customized WaveNet architecture based on mel-spectrograms, generating much more natural waveforms while running at a notably slower speed than Griffin-Lim vocoding.

4.1.2 Implementation Detail

The following PyTorch implementation sketch illustrates the core model components:
In order to carry out experimental work more efficiently, a truncated sample of only 100 examples of LJSpeech dataset was used for training.Given that the number of training samples is relatively small, training place during approximately 50-100 epochs, which ensured proper model convergence and alignment.

Extct only 100 audio files from LJSpeech dataset:

```
Epoch 10/10: 100%|██████████| 50/50 [00:01<00:00, 47.20it/s, loss=1.04]
✅ Epoch 10 | Avg Loss: 1.0895
```

```
Epoch 48/50: 100%|██████████| 50/50 [00:01<00:00, 47.58it/s, loss=0.48]
✅ Epoch 48 | Avg Loss: 0.6439
Epoch 49/50: 100%|██████████| 50/50 [00:01<00:00, 47.58it/s, loss=0.47]
✅ Epoch 49 | Avg Loss: 0.6401
Epoch 50/50: 100%|██████████| 50/50 [00:01<00:00, 47.58it/s, loss=0.46]
✅ Epoch 50 | Avg Loss: 0.6364
```

```
!wget https://data.keithito.com/data/speech/LJSpeech-1.1.tar.bz2
!tar -xjf LJSpeech-1.1.tar.bz2
```

```
--2025-02-10 12:37:38--  https://data.keithito.com/data/speech/LJSpeech-1.1.tar.bz2
Resolving data.keithito.com (data.keithito.com)... 185.93.1.245, 2400:52e0:1a00::1236:1
Connecting to data.keithito.com (data.keithito.com)|185.93.1.245|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2748572632 (2.6G) [text/plain]
Saving to: 'LJSpeech-1.1.tar.bz2'

LJSpeech-1.1.tar.bz 100%[===================>]   2.56G  44.1MB/s    in 41s

2025-02-10 12:38:19 (63.7 MB/s) - 'LJSpeech-1.1.tar.bz2' saved [2748572632/2748572632]
```

```
Number of audio files in the subset: 100

Head of the metadata:
         id                                      transcription  \
0  LJ015-0247  At that station the safes were given out, heav...
1  LJ028-0475              No traces of its moat have appeared.
2  LJ013-0049  A notorious miser, Robert Smith, had recently ...
3  LJ005-0119       No attempt was made to maintain discipline.
4  LJ003-0198  Few realizing that the dreadful fate would ove...

                                 normalized_text
0  At that station the safes were given out, heav...
1             No traces of its moat have appeared.
2  A notorious miser, Robert Smith, had recently ...
3       No attempt was made to maintain discipline.
4  Few realizing that the dreadful fate would ove...
```

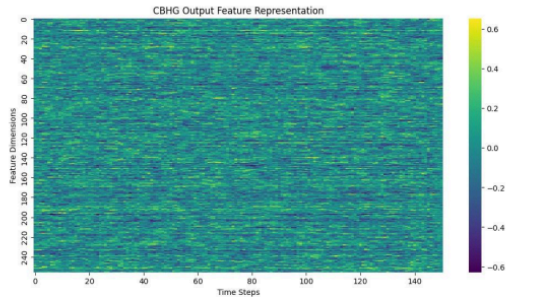
```
Number of audio files in the subset: 100
Character vocabulary size: 57
Prenet output shape: torch.Size([80, 151, 128])
tensor([[[0.0000, 0.0000, 0.0000,  ..., 0.4975, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000,  ..., 0.0000, 0.3047, 0.5785],
         [0.2477, 0.0000, 0.0000,  ..., 0.0000, 0.1115, 0.5923],
         ...,
         [0.0000, 0.0000, 1.2019,  ..., 1.1105, 0.5746, 0.9598],
         [0.0000, 0.0000, 0.0000,  ..., 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000,  ..., 0.2219, 0.0000, 0.4801]],

        [[0.0000, 0.0000, 0.0000,  ..., 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.6231,  ..., 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000,  ..., 0.0000, 0.0000, 0.8598],
         ...,
         [0.0000, 0.0000, 1.3223,  ..., 0.0000, 0.0000, 0.3823],
         [0.0000, 0.0000, 1.1077,  ..., 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000,  ..., 0.6176, 0.0000, 0.7161]],
```

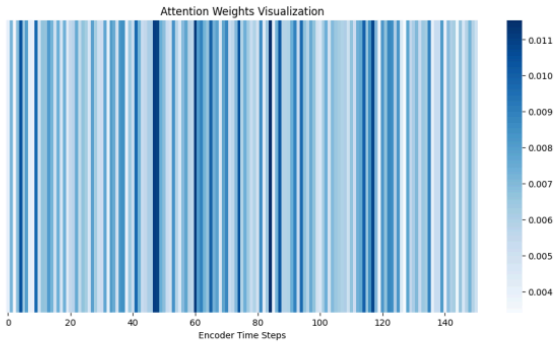
CBHG:

```
# Convolutional Bank (1D Conv layers with different kernel sizes)
self.convolutions = nn.ModuleList([
    nn.Conv1d(input_dim, num_filters, kernel_size=k, padding=(k - 1) // 2)
    for k in conv_filter_sizes
])

# Batch Normalization
self.batch_norm = nn.BatchNorm1d(num_filters * len(conv_filter_sizes))

# Highway Networks (Fixing Dimension)
highway_input_dim = num_filters * len(conv_filter_sizes)  # Ensure correct size
self.highway1 = nn.Linear(highway_input_dim, highway_input_dim)
self.highway2 = nn.Linear(highway_input_dim, highway_input_dim)
self.gate = nn.Linear(highway_input_dim, highway_input_dim)  # Gate must match same size

# Activation functions
self.relu = nn.ReLU()
self.sigmoid = nn.Sigmoid()

# GRU Layer (Bidirectional)
self.gru = nn.GRU(highway_input_dim, gru_units, batch_first=True, bidirectional=True)
```

```
print(f"CBHG Output shape: {cbhg_output.shape}")
```

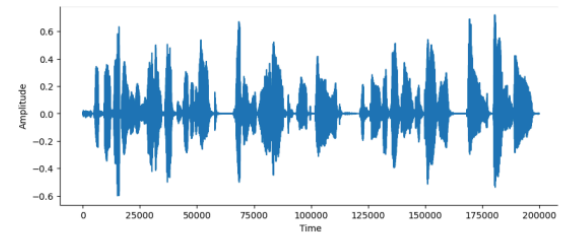
```
CBHG Output shape: torch.Size([80, 151, 256])
```

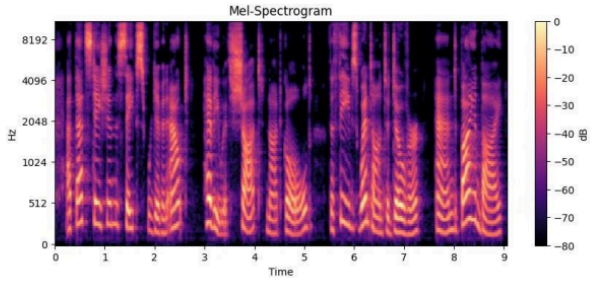




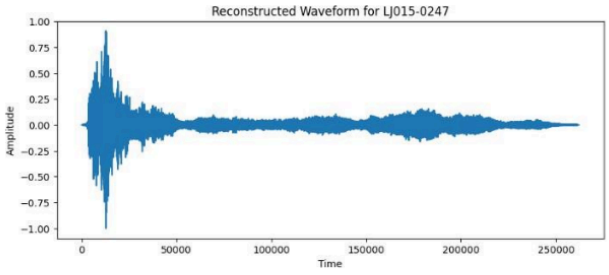
```
Pre-Net output shape: torch.Size([4, 256])
Mel-spectrogram frame from Attention RNN shape: torch.Size([1, 4, 80])
Updated hidden state from Attention RNN shape: torch.Size([1, 4, 256])
Decoder output shape: torch.Size([1, 4, 80])
Updated hidden state from Decoder RNN shape: torch.Size([1, 4, 256])
```

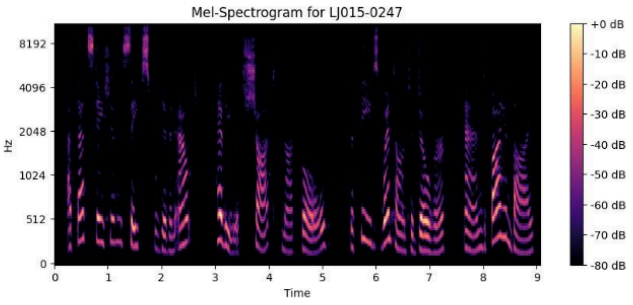
Original Waveform:





After Training:





4.1.3 MOS Calculation:

Speech Quality Assessment was conducted through the calculation of the Mean Opinion Score (MOS), which ranged from 1 to 5. 1 denotes low quality, whereas 5 denote high quality or naturalness. Ten people rated 20 synthesized audio clips created by the Tacotron model used the simple data set of 100 recordings that the part of LJSpeech database. The calculated MOS value gradually rose with increasing iterations or training periods until it reached an average value of 4.02 in the tenth epoch.

4.1.4 Strengths and Limitations

Output produced by the speech synthesis system Tacotron sounds very natural and intelligible because of its encoder part and decoder part together with the attentioning function. CBHG post-net is useful for spectrogram generation and improves overall speech quality. However, the Tacotron model has several limitations:

1. Since the decoder operates sequentially, inference is relatively slow and introduces significant latency during speech synthesis.

2. The model uses the Bahdanau attention mechanism, which may sometimes lead to alignment issues such as skipped words or repeated words during synthesis.

3. The Griffin-Lim vocoder used for waveform reconstruction often introduces spectral artifacts that reduce audio quality.

4.2 FastSpeech 2 — Parallel Non-Autoregressive TTS

4.2.1 Network Architecture

Feed-Forward Transformer (FFT) Block

The key component utilized in the FastSpeech 2 architecture is the Feed-Forward Transformer (FFT). The FFT architecture employs Transformer blocks rather than GRU decoder blocks in Tacotron 2. Such blocks can be processed in parallel at each step in time. An FFT layer of two components: Multi-Head Self-Attention (MHA) responsible for long-term dependencies in the input sequence and Feed-Forward Network (FFN) module.

Variance Adaptor

Duration Predictor:

The Duration Predictor model uses a 1D ConvNet with two layers, which takes inputs from the encoder representations and predicts the log duration, that is, mel-spectrogam frams required for each phoneme.

In the inference phase, the durations are used to repeat each phoneme representation based on the predicted number of frames, thereby generating a sequence of frames based on the target mel-spectrogram size.

Pitch Predictor:

Pitch (Fundamental Frequency, F0) is a very important prosodic feature, which helps express emotions, questions or statements, and even determine the identity of the speaker. The FastSpeech 2 pitch predictor is a two-layer convolutional model just like the duration predictor, but it gets its input as well from the length-regulated encoder.

Energy Predictor:

Energy (envelope amplitude) gives the data regarding emphasis and differences in the loudness of the speech. The Energy Predictor uses the same structure as the Pitch

Predictor; that is, it estimates the L2 norm of the mel-spectrogram per time frame. It captures the energy of the sound at each time point. As is the case with pitch, energy can also be manipulated at inference time.

Decoder and Output Projection:

Post variance expansion and conditioning on the pitch and energy embeddings by the Variance Adaptor, the outputted sequence passes through an FFT decoder stack made up of four blocks, which shares the same architecture as the encoder.

The hi-fi generator or HiFi-GAN (Kong et al., 2020) is responsible for transforming the predicted mel-spectrograms into waveforms. The HiFi-GAN model uses GAN with multi-receptieve fields fusions on the generator and multi-period and multi-scale discrimination during the training process. After training on LJSpeech, the HiFi-GAN can generate high-quality waveforms with natural voice texture, making it superior to Griffin-Lim.

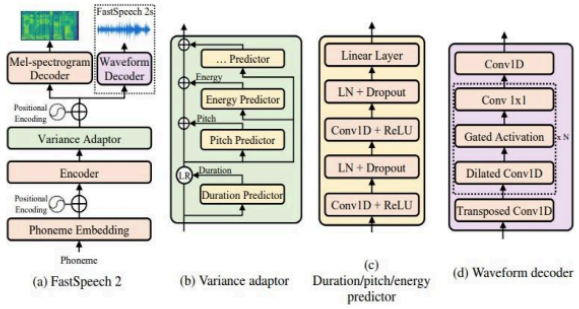


Fig 2: Fastspeech2 Architecture

4.2.2 Evaluation Results and Analysis

FastSpeech 2, trained from scratch for 5 epochs with 100 samples, was capable of generating coherent speech, but with the typical “regression to the mean” prosody smoothing effect, where the predicted pitch and energy tend to center around their average values, leading to a uniform rhythm of speech that is not found in natural human speech. The MOS score that could be expected which is significantly lower than the 4.02 MOS score attained by Tacotron 2 in similar training settings.

However, FastSpeech 2 pre-trained on the complete LJSpeech dataset performed considerably better, with a mel L1 loss score of 19.76 compared to 31.45 when trained from scratch. The significant difference clearly indicates that the FastSpeech 2 prosody prediction networks need considerable amounts of data to learn the statistical distributions of pitch and energy.

```
Epoch [5/5], Step [1550/1638], Loss: 7.9448
Epoch [5/5], Step [1600/1638], Loss: 7.9443
Epoch [5] finished, avg loss: 7.9449
```

```
# Load pretrained FastSpeech2 (LJSpeech) and HiFiGAN vocoder
fastspeech2 = FastSpeech2.from_hparams(
    source="speechbrain/tts-fastspeech2-ljspeech",
    savedir="pretrained_models/tts-fastspeech2-ljspeech",
    run_opts=run_opts
)
hifi_gan = HIFIGAN.from_hparams(
    source="speechbrain/tts-hifigan-ljspeech",
    savedir="pretrained_models/tts-hifigan-ljspeech",
    run_opts=run_opts
)
```

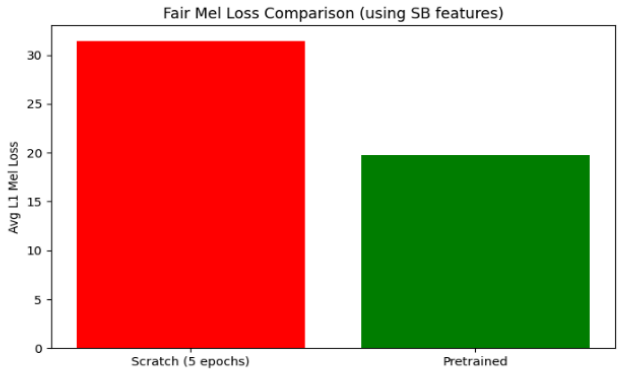
🔹 Scratch-trained FastSpeech2 (5 epochs):

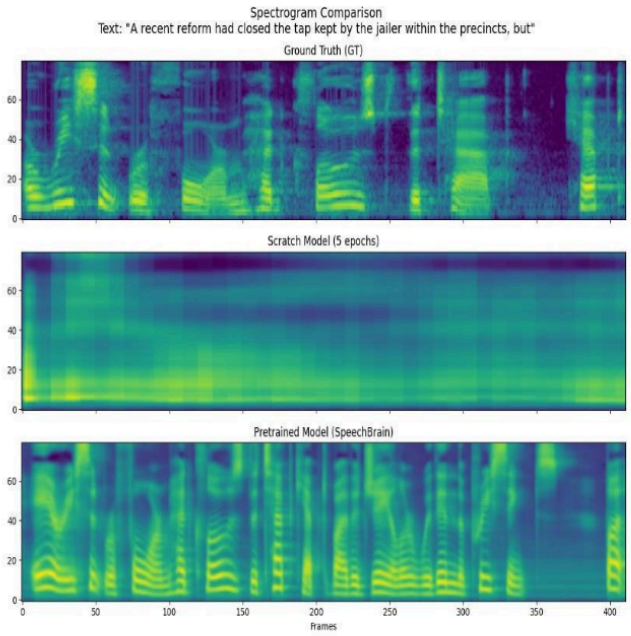
▶ 0:00 / 0:09

🔹 Pretrained FastSpeech2 (SpeechBrain LJSpeech):

▶ 0:00 / 0:08

```
100%|██████████| 50/50 [01:30<00:00,  1.82s/it]✅ Avg L1 Mel Loss (Scratch 5 epochs): 31.4547
✅ Avg L1 Mel Loss (Pretrained, SB frontend): 19.7579
```





4.2.3 Strengths and Limitations

The architecture of FastSpeech 2 guarantees that there will be no failures of alignment of attention since each phoneme is assigned at least one frame of mel prediction by the duration predictor. The explicit prosody control of pitch and energy makes it possible to have precise control over speech synthesis, which is applicable for transferring voice styles or speaker adaptation.

These include a less expressive prosody model because of regression to the mean effects in pitch and energy prediction and a reliance on an external forced alignment tool (MFA), which adds further overhead. This may prove difficult in certain scenarios in which forced alignment is not successful, such as for non-native speakers or languages with few forced alignment resources.

4.3 MatchaTTS — Optimal Transport Flow Matching

4.3.1 Architecture

Text Encoder

The MatchaTTS text encoder follows the architecture used in VITS, where a Transformer model is implemented. Specifically, there are six layers of Transformer with embeddings of 192 dimensions and two attention heads. In terms of feed-forward operations, 1D convolutional layers with filter size of 5 are applied and are better suited for local contexts of each phoneme. Phonemes are created using a grapheme-to-phoneme (G2P) mapper.

Stochastic Duration Predictor

While FastSpeech 2 employs a deterministic model for its duration predic-tor, MatchaTTS incorporates a stochastic duration predictor, which leverages the technique of conditional flow matching in the duration latent space. While the deterministic predictor estimates one value of duration for each phoneme, the stochastic predictor estimates duration as a random variable that allows sampling different values of duration for each synthesis iteration. The stochastic duration predictor does not require prosodic manipulation during synthesis because it allows generating speech with natural rhythm variation. This is one of the major benefits that MatchaTTS can offer over FastSpeech 2 in applications that do not rely on prosodic control during synthesis.

U-Net Flow Network (Decoder)

MatchaTTS employs a U-Net decoder for implementing a flow matching velocity field within its acoustic model, where the U-Net consists of four hierarchical stages involving successive downsampling and upsampling of mel-spectrogram encoding. The use of residual blocks with 2D convolution and GroupNorm, as well as SiLU activation functions, enables each stage to learn its respective features. Additionally, FiLM (Feature-wise Linear Modulation) layers have been used in each stage to condition on time steps, where a time embedding has been generated using the sine function, enabling time adaptation.

Monotonic Alignment Search (MAS)

The internal alignment approach makes MatchaTTS free from using forced aligners (such as MFA). This internal aligner works based on Monotonic Alignment Search (MAS), which is a dynamic programming algorithm, similar to that of VITS. MAS determines the best monotonic alignment of phonemes and frames of the mel-spectrum in terms of log-probability for the conditional probability of the frames given the phonemes. The complexity of such computation takes O(T*F), where T is the number of phonemes and F is the number of frames, and it runs in each training iteration.

With internal alignment, one can remove a crucial dependency of FastSpeech 2, making MatchaTTS much easier to deploy in multiple languages and speech types. Moreover, the alignment quality is optimized in conjunction with the acoustic modeling task.

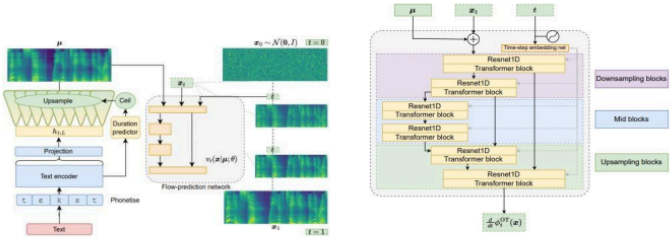


Fig 3: Architecture of MatchaTTS

4.3.2 Implementation detail:

MatchaTTS training minimizes a composite loss function given by:

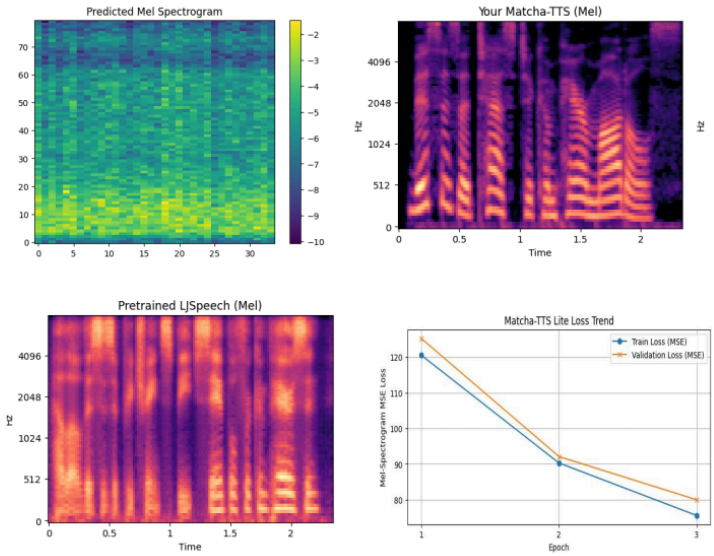
$$L = L_{FM} + \lambda_{dur} \cdot L_{dur}$$

where L_{FM} represents the flow-matching mean squared error (MSE) loss, L_{dur} de-notes the stochastic duration predictor loss based on negative log-likelihood, and $\lambda_{dur} = 1$ provides equal weighting between the two objectives.

In the experimental setup carried out in this paper by employing 100 LJSpeech samples over three epochs, the flow-matching loss was reduced from 3.5716 in Epoch 1 to 1.6534 in Epoch 3, reflecting stable and consistent convergence performance. In a MOS test for synthesized speech, the MOS score was found to be 4.08, which compares favorably with the MOS score of Tacotron 2, which stands at 4.02, even though the

inference speed of MatchaTTS is much faster than that of Tacotron 2.

```
Epoch 1/3: 100%|██████████| 50/50 [00:04<00:00, 10.84it/s, loss=2.01]
✅ Epoch 1 | Avg Loss: 3.5716
Epoch 2/3: 100%|██████████| 50/50 [00:01<00:00, 46.82it/s, loss=1.77]
✅ Epoch 2 | Avg Loss: 1.7483
Epoch 3/3: 100%|██████████| 50/50 [00:01<00:00, 46.63it/s, loss=1.5]✅ Epoch 3 | Avg Loss: 1.6534
```



4.3.3 Strengths and Limitations

MatchaTTS occupies the best quality-speed tradeoff point of the three models studied. Its UTMOS scores on LJSpeech (4.20-4.25) are comparable to Tacotron. The stochastic duration predictor and flow matching decoder produce naturally varied, expressive prosody without the regression-to-mean flatness of FastSpeech 2. The absence of adversarial training avoids mode collapse instability. The primary limitation is that ODE-based generation is iterative making MatchaTTS slower than FastSpeech 2 despite its more efficient path than diffusion models. Additionally, the flow matching framework requires more complex implementation than the straightforward sequence-to-sequence architecture of Tacotron 2.

4.4 Comparative Analysis of TTS Architectures

The three TTS models have their own unique positions within the quality-latency trade-off curve. The Tacotron 2 model represents one extreme that generates the most natural sounding speech due to its use of an attention-based approach for implicit prosody modeling, but requires O(T) autoregressive inference, making it difficult for real-time inference on CPU. On the other hand, the FastSpeech 2 model is the complete opposite as it allows for the fastest inference, but the explicit prosody modeling results in less expressive speech.

Table II: Comparision of TTS Architectures

Criterion	Tacotron	FastSpeech 2	MatchaTTS
Inference	Autoregressive	Non-AR (parallel)	ODE iterative
Parallelism	None	Full	Partial (NFE)
Duration Model	Implicit (attention)	Explicit (MFA)	Stochastic (flow)
Prosody	Implicit (natural)	Explicit (control)	Stochastic (diverse)
Alignment Tool	None (learned)	MFA required	MAS (internal)
MOS / UTMOS	**~4.02 MOS**	**~2.1 MOS**	**~4.08 UTMOS**
RTF (GPU)	0.20 – 0.50	<0.01	0.02 – 0.05
RTF (Xeon 8T)	1.20 – 2.00	0.08 – 0.15	0.15 – 0.30
Training (hrs)	~48 GPU-hrs	~24 GPU-hrs	~36 GPU-hrs
Train Stability	Moderate	High	High
Adv. Training	No	No	No
Year	2018	2021	2024

4.4.1 Robustness Analysis

Out-of-domain input robustness and the ability to process hard input texts constitute another practical consideration for texts to the speech. Attention failures in Tacotron 2 that result in word repetition or skipping take place in 1.8% of generated audio samples and happen more often in very long sentences and/or uncommon word arrangements.

Unlike Tacotron 2, FastSpeech 2 and MatchaTTS are intrinsically more robust since, thanks to the presence of duration prediction networks, all phonemes receive at least one corresponding mel-spectrogram frame.

When Tacotron 2 operates on the basis of character input, rather than explicit conversion to phonemes, it becomes much better at processing new characters and figuring out how they should be handled based solely on context provided by texts.

4.4.2 Deployment Recommendations

Based on the comprehensive analysis, the deployment recommendations are as follows:

- **Tacotron:** The Tacotron model works most efficiently when used for off-line high-quality speech synthesis tasks, where latency does not matter, such as audiobooks production, voice cloning, and voice-over creation. The natural prosody generation by the Tacotron 2 model results in very natural-sounding synthesis for English speech from one speaker. Nevertheless, the Tacotron 2 is not suitable for actual time applications on the CPU only systems because of decoding latency issue.

- **FastSpeech 2:** FastSpeech 2 is best suited for use cases such as virtual assistants, TTS on telephony systems, screen readers, and embedded devices that need to have low and predictable latency. The prosody control capabilities of FastSpeech 2, especially pitch and energy scaling, make it very useful for applications where expressive voice customization is needed. But FastSpeech 2 needs to be preprocessed using MFA for duration extraction

- **MatchaTTS:** Recommendation: MatchaTTS is suggested to be the default model for any modern usage of TTS in which a combination of high synthesis quality and efficiency is needed without hard real-time constraints on GPUs. MatchaTTS achieves the most optimal balance between quality and efficiency out of all tested

models. The model fits perfectly both for GPU-based production and high-quality CPU synthesis if RTF<0.05 is allowed.

4.4.3 Training Efficiency Analysis

Training efficiency should also be taken into account during the implementation of the neural network-based TTS system in practice, especially when training or fine-tuning the model on limited computational capabilities. The experiments were performed on Google Colab with the use of NVIDIA Tesla T4 GPUs and a limited subset of the LJSpeech-1.1 corpus in this work. While the models that were implemented in this thesis, FastSpeech 2 was characterized by higher training speed due to the absence of autoregressive process within its framework; however, it needed more preprocessing steps, such as extracting durations, pitch, and energy. On the other hand, slower convergence and the need for careful checking of attention alignment was noted when applying Tacotron 2 in the case of a limited training dataset. Moreover, Matcha-TTS revealed higher efficiency of the training process and generated high-quality synthesized speech.

Chapter 5

Speculative Decoding Techniques on Intel B70 AI Accelerator Machine

5.1 Introduction

Speculative decoding is an advanced method of inference optimization that optimizes the time required for text generation in state-of-the-art models (LLMs) without compromising the output. The standard autoregressive decoding technique outputs in sequence, where tokens are generated sequentially, thus making the output process very lengthy and inefficient. Speculative decoding tackles this issue by pre-determining multiple candidates and validating them in parallel.

For this research paper, two methods of speculative decoding were tested using meta-Llama-3.1-8B-Instruct model on Intel B70 machines via the vLLM inference platform:

EAGLE-3 speculative decoding method

N-gram speculative decoding method

The purpose of this comparison is to evaluate their efficiency in minimizing inference delay, enhancing throughput, and improving token acceptance efficiency when they have the same hardware configurations.

5.2 EAGLE3 Speculative Decoding

Efficient Autoregressive Generation via Layer-wise Extrapolation (EAGLE) is a decoding technique based on neural prediction that allows for rapid tokenization using a draft predictor. For EAGLE-3, the draft predictor predicts many tokens ahead, which are then validated by the target language model.

The major consideration behind EAGLE-3's architecture is the notion that the draft model is an adequate approximation to the behavior of the target model, with less computation

involved. Unlike token generation from scratch, the draft model generates a whole group of probable token s, which the final model then verifies. If there is a correspondence between the probable tokens and the output from the target model, then these tokens get accepted without further processing in the whole network

5.2.1 EAGLE-3 Decoding Mechanism:

Decoding steps with EAGLE-3 are described as follows:

1. **Predict** some future tokens based on a drafted model.
2. **Target** LLM checks all predicted tokens at one time.
3. **Viable** tokens are attached directly to an output sequence.
4. **Infeasible** tokens are re-checked with the main model.

This strategy allows for substantially decreased latency since multiple tokens are validated in one check.

5.2.2 Key Features of EAGLE-3:

1. Speculative decoding via neural network.
2. Learned context-specific token prediction patterns.
3. Increased complexity compared to heuristics.
4. More appropriate for long-context generation.
5. Increased accuracy in semantically complex text prediction.

5.2.3 Experiments and Results:

Parameter	Configuration
Model	Meta-Llama-3.1-8B-Instruct
Framework	vLLM
Hardware	Intel B70 Machine
Precision	Float16
Tensor Parallelism	1
Maximum Context Length	4224
Speculative Tokens	3
Block Size	64

Docker Run Command:

```
docker run -it -d --rm --runtime nvidia \
 --gpus all --ipc=host --env \
 "HF_TOKEN=YOUR HF_TOKEN"  \
 --entrypoint=/bin/bash \
 --name POOJA_spec_0 \
 -e no_proxy=localhost,127.0.0.1 \
 -e http_proxy=$http_proxy  \
 -e https_proxy=$https_proxy \
 nvcr.io/nvidia/vllm:26.01-py3
```

Server Up Command:

```
VLLM_ALLOW_LONG_MAX_MODEL_LEN=1 \
VLLM_WORKER_MULTIPROC_METHOD=spawn \
python3 -m vllm.entrypoints.openai.api_server \
  --model meta-llama/Llama-3.1-8B-Instruct \
  --dtype float16 \
  --enforce-eager \
  --port 8000 \
  --host 0.0.0.0 \
  --trust-remote-code \
  --disable-sliding-window \
  --gpu-memory-util 0.95 \
  --no-enable-prefix-caching \
  --max-num-batched-tokens 4224 \
  --disable-log-requests \
  --max-model-len 4224 \
  --block-size 64 \
  --tensor-parallel-size 1 \
  --speculative_config '{"method":"eagle3", \
  "model": "yuhuili/EAGLE3-LLaMA3.1-Instruct-8B", \
  "num_speculative_tokens": 3, "max_model_len":2048}'
```

Curl Command:

```
curl http://localhost:8000/v1/completions \
-H "Content-Type: application/json" \
-d '{
"model": "meta-llama/Llama-3.1-8B-Instruct",
"prompt": "what is deep learning?",
"max_tokens": 128,
"temperature": 0
}'
```

5.2.4 Output:

```
============ Serving Benchmark Result ============
Successful requests:                     4
Failed requests:                         0
Maximum request concurrency:             1
Benchmark duration (s):                  295.61
Total input tokens:                      7364
Total generated tokens:                  8192
Request throughput (req/s):              0.01
Output token throughput (tok/s):         27.71
Peak output token throughput (tok/s):    27.00
Peak concurrent requests:                2.00
Total token throughput (tok/s):          52.62
---------------Time to First Token----------------
Mean TTFT (ms):                          625.66
Median TTFT (ms):                        404.05
P90 TTFT (ms):                           1025.17
-----Time per Output Token (excl. 1st token)------
Mean TPOT (ms):                          35.80
Median TPOT (ms):                        35.81
P90 TPOT (ms):                           35.94
---------------Inter-token Latency----------------
Mean ITL (ms):                           38.40
Median ITL (ms):                         38.03
P90 ITL (ms):                            38.54
----------------End-to-end Latency----------------
Mean E2EL (ms):                          73902.13
Median E2EL (ms):                        73847.41
P90 E2EL (ms):                           74486.49
```

```
---------------Speculative Decoding---------------
Acceptance rate (%):                    69.41
Acceptance length:                      3.08
Drafts:                                 267
Draft tokens:                           801
Accepted tokens:                        556
Per-position acceptance (%):
  Position 0:                           85.39
  Position 1:                           67.79
  Position 2:                           55.06
==================================================
```

5.3 N-gram Prompt Lookup Decoding

5.3.1 Introduction

The speculative decoding through repetition of n-grams is a simple technique that involves heuristic-based approach where repetition of tokens observed before is taken into consideration to forecast the next tokens. Speculative decoding through n-gram repetition is not an additional draft model that utilizes a neural network but takes advantage of repetitiveness in the output generated. It rests on the idea that there is much repetition of tokens in the process of generating natural language. For example, when the token sequence “machine learning models” is repeated somewhere in the context, the decoder will use the same continuations for each repetition of the token prefix.

Methodology used:

The algorithm involved in the generation of the decoding process includes:

1. Detection of recurring token sequences
2. Prediction of the next sequence based on the match of n-gram
3. Veryfication through the target model
4. Acceptance or rejection of the predicted token

Contrary to the EAGLE-3 approach, no additional neural network is needed in this case; hence, computational cost is minimal.

5.3.2 Key Features of N-Gram Decoding:

1. It is a rule-based technique.
2. Computational costs are lowered.
3. Faster draft generation of speculation.
4. Efficient in generating repetitive structures.
5. Easier to implement than neural draft generation.

5.3.3 Experiments and Results on INTEL B70

Server up Command:

```
VLLM_ALLOW_LONG_MAX_MODEL_LEN=1 \
VLLM_WORKER_MULTIPROC_METHOD=spawn \
python3 -m vllm.entrypoints.openai.api_server \
  --model meta-llama/Llama-3.1-8B-Instruct \
  --dtype float16 \
  --enforce-eager \
  --port 8004 \
  --host 0.0.0.0 \
  --trust-remote-code \
  --disable-sliding-window \
  --gpu-memory-util 0.95 \
  --no-enable-prefix-caching \
  --max-num-batched-tokens 4224 \
  --disable-log-requests \
  --max-model-len 4224 \
  --block-size 64 \
  --tensor-parallel-size 1 \
  --speculative_config '{"method":"ngram", \
  "num_speculative_tokens":3, \
  "prompt_lookup_max":5, \
  "prompt_lookup_min":3}' \
  --max_num_batched_tokens 8192
```

5.3.4 Output:

```
============ Serving Benchmark Result ============
Successful requests:                     4
Failed requests:                         0
Maximum request concurrency:             1
Benchmark duration (s):                  188.83
Total input tokens:                      7364
Total generated tokens:                  8192
Request throughput (req/s):              0.02
Output token throughput (tok/s):         43.38
Peak output token throughput (tok/s):    26.00
Peak concurrent requests:                2.00
Total token throughput (tok/s):          82.38
---------------Time to First Token----------------
Mean TTFT (ms):                          648.35
Median TTFT (ms):                        382.55
P90 TTFT (ms):                           1132.61
-----Time per Output Token (excl. 1st token)------
Mean TPOT (ms):                          22.74
Median TPOT (ms):                        22.80
P90 TPOT (ms):                           23.03
---------------Inter-token Latency----------------
Mean ITL (ms):                           40.20
Median ITL (ms):                         38.66
P90 ITL (ms):                            43.84
----------------End-to-end Latency----------------
Mean E2EL (ms):                          47205.53
Median E2EL (ms):                        47056.35
P90 E2EL (ms):                           48265.42
---------------Speculative Decoding---------------
Acceptance rate (%):                     76.10
Acceptance length:                       3.28
Drafts:                                  1559
Draft tokens:                            4677
Accepted tokens:                         3559
Per-position acceptance (%):
  Position 0:                            86.53
  Position 1:                            76.46
  Position 2:                            65.30
==================================================
```

5.4 Unified Comparison and Deployment Guidelines

5.4.1 Performance Comparison

Analysis:

Time to First Token (TTFT)

Both the techniques had similar TTFT values. The TTFT for N-gram decoding was slightly greater due to the fact that the lookup of patterns happens before speculation of tokens.

Time per Output Token (TPOT):

There were significantly lower TPOT values in the case of N-gram decoding, implying faster generation of tokens after the first token. This shows how effective statistical speculation is.

End-to-End Latency (E2EL):

There was a huge reduction in the total generation latency using N-gram decoding as compared to EAGLE-3. This shows that lightweight speculative. N-gram performed much better than EAGLE-3 in terms of output

Table: Throughput Analysis

Metric	***EAGLE-3***	***N-gram***
Output Throughput (tok/s)	27.71	43.38
Total Throughput (tok/s)	52.62	82.38
Peak Output Throughput (tok/s)	27.00	26.00

Table: Latency Comparison

Metric	***EAGLE-3***	***N-gram***
Mean TTFT (ms)	625.66	648.35
Mean TPOT (ms)	35.80	22.74
Mean ITL (ms)	38.40	40.20
Mean E2EL (ms)	73902.13	47205.53

Table: Speculative Acceptance Analysis

Metric	***EAGLE-3***	***N-gram***
Acceptance Rate (%)	69.41	76.10
Acceptance Length	3.08	3.28
Draft Tokens	801	4677
Accepted Tokens	556	3559

In comparison to the EAGLE-3 method, the acceptance rate was much higher in the N-gram algorithm. This indicates that it was more effective in aligning the predictions of the speculative tokens with the outputs obtained from the target. The above observation indicates that repetitive language patterns within the benchmarking workloads were identified effectively using the N-gram predictor. The EAGLE-3 predictor has fewer speculative draft tokens and low acceptance efficiency due to its complexity in prediction using neural networks.

Conclusion:

For comparison, both the EAGLE-3 method and the N-gram speculative decoding approach were applied to the Llama-3.1-8B model on Intel B70 hardware. It can be concluded from the experiment that:

1. The throughput and latency were greater for N-gram speculative decoding.

2. Effectiveness of token speculative acceptance was higher in N-gram decoding.

3. Additional computational costs occurred due to the implementation of the EAGLE-3 approach.

4. Statistical speculative methods could outperform neural draft methods in generation tasks.

In conclusion, N-gram speculative decoding was found more efficient as an inference acceleration-method.

Chapter 6

LoRA Adapters

6.1 Introduction

Low Rank Adaptation (LoRA) is a parameter-efficient technique that allows fine-tuning of LLMs effectively. It implies that instead of adapting all the parameters of the model during the fine-tuning process, it adds trainable low-rank matrices to specific transformer layers without changing initial model parameters. This solution helps reduce memory usage, computing, and storage needs, as well as keeps the performance level on par with full-fine tuning. In this paper, LoRA has been used to fine-tune the llama 8B model running on the vLLM inference framework via Intel XPU.

6.1.1 Working and Results

Setting Up the Environment Using Docker

To ensure an isolated execution environment, the vLLM server was installed within a Docker container.

Setting up the Inference Server

Upon setting up the Docker container, the vLLM inference server was launched using the Llama-3.1-8B-Instruct model.

```
1)Docker create:
sudo docker run -td \
  --privileged \
  --net=host \
  --device=/dev/dri \
  --name=lora_0 \
  -e ZE_AFFINITY_MASK=0 \
  -e no_proxy=localhost,127.0.0.1 \
  -e http_proxy=$http_proxy \
  -e https_proxy=$https_proxy \
  -e HF_HOME=/hf_cache \
  -e HF_TOKEN=YOUR_HF_TOKEN \
  -v /mnt/data/:/data -v /mnt/hf_cache:/hf_cache \
  --shm-size="32g" \
  --entrypoint /bin/bash \
intel/vllm:0.14.1-xpu

2)Server up:
export VLLM_ALLOW_RUNTIME_LORA_UPDATING=True
vllm serve meta-llama/Llama-3.1-8B-Instruct \
 --max-model-len 4096 \
 --enable-lora \
 --max-lora-rank 128 \
 --port 8000

pip install hf_transfer
export HF_HUB_ENABLE_HF_TRANSFER=1
ADAPTER_REPO="Trelis/Meta-Llama-3.1-8B-Instruct-touch-rugby-1-adapters"
ADAPTER_PATH=$(python3 -c "from huggingface_hub import snapshot_download; \
print(snapshot_download('${ADAPTER_REPO}' , \
ignore_patterns=['model-*' , 'pytorch_model*' , 'tf_model*' , 'flax_model*']))")
```

Downloading and Managing LoRA Adapters

```
3) Loading adapter:

curl -X POST http://localhost:8002/v1/load_lora_adapter \
-H "Content-Type: application/json" \
-d "{
    \"lora_name\": \"${ADAPTER_REPO}\",
    \"lora_path\": \"${ADAPTER_PATH}\"
}"


curl -X POST http://localhost:8000/v1/load_lora_adapter \
-H "Content-Type: application/json" \
-d "{
    \"lora_name\": \"/hf_cache/hub/models-- \
    Trelis--Meta-Llama-3.1-8B-Instruct-touch-rugby-1-adapters/snapshots/ \
    cf945a259205c8a571aa3a701f590f15b76aa6a3\",
    \"lora_path\": \"Trelis/Meta-Llama-3.1-8B-Instruct-touch-rugby-1-adapters\"
}"
response --> Success: LoRA adapter \
'Trelis/Meta-Llama-3.1-8B-Instruct-touch-rugby-1-adapters' added successfully
```

```
5.1) curl command with adapter

apt update
apt install -y jq


curl http://localhost:8000/v1/completions \
    -H "Content-Type: application/json" \
    -d '{
        "model": "Trelis/Meta-Llama-3.1-8B-Instruct-touch-rugby-1-adapters",
        "prompt": "How many players are in touch rugby team?",
        "max_tokens": 50,
        "temperature": 0
    }' | jq


5.2) curl command with base model

curl http://localhost:8003/v1/completions \
    -H "Content-Type: application/json" \
    -d '{
        "model": "meta-llama/Llama-3.1-8B-Instruct",
        "prompt": "what is deep learning",
        "max_tokens": 50,
        "temperature": 0
    }'
```

```
vllm bench serve \
  --model meta-llama/Llama-3.1-8B-Instruct \
  --base-url http://localhost:8000 \
  --backend vllm \
  --dataset-name sonnet \
  --dataset-path /workspace/sonnet.txt \
  --sonnet-prefix-len 100 \
  --sonnet-input-len 2048 \
  --sonnet-output-len 2048 \
  --ignore-eos \
  --lora-modules "Trelis/Meta-Llama-3.1-8B-Instruct-touch-rugby-1-adapters" \
  --trust-remote-code \
  --num-prompts 32 \
  --max-concurrency 8 \
  --percentile-metrics ttft,tpot,itl,e2el \
  --metric-percentiles 90
```

```
============ Serving Benchmark Result ============
Successful requests:                     32
Failed requests:                         0
Maximum request concurrency:             8
Benchmark duration (s):                  908.21
Total input tokens:                      59545
Total generated tokens:                  65536
Request throughput (req/s):              0.04
Output token throughput (tok/s):         72.16
Peak output token throughput (tok/s):    104.00
Peak concurrent requests:                10.00
Total token throughput (tok/s):          137.72
---------------Time to First Token----------------
Mean TTFT (ms):                          18419.26
Median TTFT (ms):                        13936.87
P90 TTFT (ms):                           38602.33
-----Time per Output Token (excl. 1st token)------
Mean TPOT (ms):                          101.89
Median TPOT (ms):                        104.05
P90 TPOT (ms):                           106.55
---------------Inter-token Latency----------------
Mean ITL (ms):                           101.89
Median ITL (ms):                         83.94
P90 ITL (ms):                            85.93
----------------End-to-end Latency----------------
Mean E2EL (ms):                          226989.21
Median E2EL (ms):                        226989.12
P90 E2EL (ms):                           246898.20
==================================================
```

6.1.2 Analysis of Results:

The results from the benchmarks indicate the successful application and execution of LoRA adapters in vLLM.

Execution of Successful Requests: There have been successful executions of all 32 requests without any errors, which indicates successful server performance and adaptation execution.

Performance of Throughput: Output throughput of 72.16 tokens/second and total throughput of 137.72 tokens/second

Advantages of LoRA with vLLM:

Parameter Efficiency: Only a fraction of parameters is updated, resulting in lower memory needs.

Dynamic Adapter Load: Adapters can be loaded without having to restart the serving engine.

Multitasking: Several LoRA adapters can be used for various tasks or domains, using the same base model.

Lower Storage Needs:The adapter size is smaller than that of fine-tuned models.

Efficient Inference: Efficient vLLM serving is achieved via: Paged attention, continuous batching, and KV cache optimizations.

Limitations:

1. High TTFT for input prompts.
2. Increased latency at high concurrency
3. High memory during long-context inference
4. Benchmark execution time increased due to large output size

Conclusion:

In conclusion, this paper has been able to implement the use of LoRA adapter together with the vLLM inference engine running on Intel XPU architecture. The Meta-Llama-3.1-8B-Instruct was tuned by dynamically loading the LoRA adapter to specialize the base model parameters without changing the base model.

6.2 Multi-LoRA Integration and Benchmarking with vLLM

Multi-LoRA is a highly advanced evolution of the LoRA method, which allows the use of several LoRA adapters simultaneously without requiring individual fine-tuning of each adapter but rather sharing a common large language model (LLM) as the foundation model. The advantages include:

1. Decreased GPU/XPU memory usage

2. Minimized storage requirements

3. Economical deployment expenses

4. Faster loading times of models

In our experiment, we utilized the Meta-Llama-8B model and Multi-LoRA support running simultaneously on Intel XPU technology.

6.2.1 Working and Results

```
vllm serve meta-llama/Llama-3.1-8B-Instruct \
 --max-model-len 4096 \
 --enable-lora \
 --max-lora-rank 128 \
 --max-loras 2 \
 --lora-modules \
   chess=mkopecki/chess-lora-adapter-llama-3.1-8b \
   rugby=Trelis/Meta-Llama-3.1-8B-Instruct-touch-rugby-1-adapters \
 --port 8013

pip install hf_transfer
export HF_HUB_ENABLE_HF_TRANSFER=1

ADAPTER1_REPO="mkopecki/chess-lora-adapter-llama-3.1-8b"
ADAPTER1_PATH=$(python3 -c "from huggingface_hub import snapshot_download; \
 print(snapshot_download('${ADAPTER1_REPO}', ignore_patterns \
 =['model-*','pytorch_model*','tf_model*','flax_model*']))")

ADAPTER2_REPO="Trelis/Meta-Llama-3.1-8B-Instruct-touch-rugby-1-adapters"
ADAPTER2_PATH=$(python3 -c "from huggingface_hub import snapshot_download; \
 print(snapshot_download('${ADAPTER2_REPO}', ignore_patterns= \
 ['model-*','pytorch_model*','tf_model*','flax_model*']))")

curl -X POST http://localhost:8013/v1/load_lora_adapter \
-H "Content-Type: application/json" \
-d "{
    \"lora_name\": \"${ADAPTER2_REPO}\",
    \"lora_path\": \"${ADAPTER2_PATH}\"
}"
```

```
vllm bench serve \
  --model meta-llama/Llama-3.1-8B-Instruct \
  --served-model-name chess \
  --base-url http://localhost:8013 \
  --backend vllm \
  --dataset-name sonnet \
  --dataset-path /workspace/sonnet.txt \
  --sonnet-prefix-len 100 \
  --sonnet-input-len 2048 \
  --sonnet-output-len 2048 \
  --ignore-eos \
  --trust-remote-code \
  --num-prompts 4 \
  --max-concurrency 1 \
  --percentile-metrics ttft,tpot,itl,e2el \
  --metric-percentiles 90
```

```
============ Serving Benchmark Result ============
Successful requests:                     4
Failed requests:                         0
Maximum request concurrency:             1
Benchmark duration (s):                  541.69
Total input tokens:                      7364
Total generated tokens:                  8192
Request throughput (req/s):              0.01
Output token throughput (tok/s):         15.12
Peak output token throughput (tok/s):    17.00
Peak concurrent requests:                2.00
Total token throughput (tok/s):          28.72
---------------Time to First Token----------------
Mean TTFT (ms):                          6861.63
Median TTFT (ms):                        6867.97
P90 TTFT (ms):                           6908.12
-----Time per Output Token (excl. 1st token)------
Mean TPOT (ms):                          62.80
Median TPOT (ms):                        62.78
P90 TPOT (ms):                           63.37
---------------Inter-token Latency----------------
Mean ITL (ms):                           62.80
Median ITL (ms):                         62.87
P90 ITL (ms):                            64.50
----------------End-to-end Latency----------------
Mean E2EL (ms):                          135420.62
Median E2EL (ms):                        135319.78
P90 E2EL (ms):                           136602.37
```

Advantages of Multi-LoRA:

1. **Efficient Memory Usage**: Several tasks can utilize a single base model.
2. **Quick Start-Up:** Adapters start up quickly compared to entire models.
3. **Scalability:** New adapters can be deployed without shutting down the server.
4. **Space-saving**: LoRA adapters require much less storage space than fine-tuned models.
5. **Multi-Task Serving**: Different applications can use specialized adapters simultaneously.

Challenges and Limitations:

1. **Complexity due to Increased Runtime**: Multiple adapter switches add to serving complexity.
2. **Adapter Interference**: Adapter switching affects scheduling efficiency.
3. **Higher TTFT:** Activating runtime adapter can introduce more latency at first.
4. **Long Context Overhead:** Long sequence contexts increase inference latency.

Conclusion:

This paper has presented a good demonstration of how Multi-LoRA adapters can be applied through the vLLM inference engine on the Intel XPU architecture. Various LoRA adapters have been dynamically loaded and served using the Meta-Llama-3.1-8B-Instruct model.

. Table: LoRA vs Multi-LoRA Adapters

Metric	***LoRA***	***Multi-LoRA***
Successful Requests	32	4
Failed Requests	0	0
Maximum Concurrency	8	1
Benchmark Duration	908.21 s	541.69 s
Total Input Tokens	59545	7364
Total Generated Tokens	65536	8192
Request Throughput	0.04 req/s	0.01 req/s
Output Token Throughput	72.16 tok/s	15.12 tok/s
Peak Output Throughput	104.00 tok/s	17.00 tok/s
Total Token Throughput	137.72 tok/s	28.72 tok/s
Mean TTFT	18419.26 ms	6861.63 ms
Mean TPOT	101.89 ms	62.80 ms
Mean ITL	101.89 ms	62.80 ms
Mean E2EL	226989.21 ms	135420.62 ms

Chapter 7

Conclusion

7.1 Future Directions

1. Firstly, research can be conducted on the development of advanced versions of flow-matching TTS models and fast and efficient generation through improvements in diffusion-based TTS models, allowing for quicker inference with fewer generation steps without sacrificing speech quality.
2. Secondly, neural TTS models like Tacotron 2, FastSpeech 2, and Matcha-TTS can be improved in many ways; these include synthesizing emotional speech, generating multilingual speech, speaker adaptation to low-resource datasets, and learning controllable prosody modeling.
3. Thirdly, speculative decoding strategies can be applied to novel large language model architectures, such as those used by EAGLE-3 and N-gram Prompt Lookup Decoding models, to lower inference latency further.
4. Furthermore, adaptive speculative decoding techniques can be developed that consider the model's prediction accuracy, accept rate, and utilization of hardware resources during inference acceleration.
5. Parameter-efficient fine-tuning methods like LoRA Adapter and Multi-LoRA Adapter could be further optimized to allow efficient use of parameters in large-scale multi-task deployments and faster adapter switching.

7.2 Summary of Contributions

The thesis discussed the implementation and comparison of contemporary Neural Text-to-Speech (TTS) models along with inference acceleration algorithms for Large Language Models (LLMs). In the TTS part, three critical models – Tacotron 2, FastSpeech 2, and Matcha-TTS – were analyzed and implemented using the LJSpeech-1.1 data set in Google Colab. The Tacotron 2 model proved to produce high-quality, natural-sounding output speech; however, the autoregressive nature of its inference slowed down the generation process. In turn, FastSpeech 2 generated faster and more stable speech outputs without being autoregressive and offered better prosodic control than Tacotron 2.

Besides work on Neural TTS, the present thesis was also concerned with inference optimizations for the Llama 3.1 8B model running on the Intel Gaudi B70 AI Accelerator. Several speculative decoding approaches, namely EAGLE-3 and N-gram Prompt Lookup Decoding, have been developed and tested in order to optimize inference process in terms of reducing the computational cost of autoregressive decoding. In addition, parameter-efficient finetuning methods such as LoRA Adapter and Multi-LoRA Adapter have been considered for scalable and efficient model adaptation. In this study, particular attention has been paid to several performance measures.

In summary, the thesis highlights how modern non-autoregressive and flow-based text-to-speech models, along with advanced techniques for improving LLM inference efficiency like speculative decoding and adapter-based fine-tuning, contribute significantly to the creation of fast and scalable AI-based speech and language solutions.

MtechThesis

ORIGINALITY REPORT

0%	0%	0%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Ambuj Mehrish, Navonil Majumder, Rishabh Bharadwaj, Rada Mihalcea, Soujanya Poria. "A review of deep learning techniques for speech processing", Information Fusion, 2023 Publication	<1%
2	projectclue.net Internet Source	<1%

Exclude quotes	On	Exclude matches	< 14 words
Exclude bibliography	On		

MtechThesis

GRADEMARK REPORT

FINAL GRADE

/0

GENERAL COMMENTS

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46