UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI
DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

BACHELOR THESIS

By
Phạm Ngọc Lâm

Title:
AI-powered Weather Reporter

External Supervisor: Mr. Đỗ Ngọc Huỳnh
Internal Supervisor: Dr. Đoàn Nhật Quang

Hanoi, September 2025

To whom it may concern,

I, …… (supervisor's name), certify that the thesis/ internship report of Mr/Ms. ……(student's name) is qualified to be presented in the Internship Jury 2024-2025.

Hanoi,

Supervisor's signature

Table of Contents

ACKNOWLEDGEMENTS

I would like to express my gratitude to all those who gave me the possibility to complete this thesis in specific as well as academic support in general that I received during the time of my studying at the University of Science and Technology of Hanoi.

Foremost, I would like to send my honest thanks to my internal and external supervisor for letting me expand my creativity and giving me suggestions during this thesis, to family and friends, and especially my mother for always giving me motivation and support.

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This report presents a digital weatherman application that integrates natural language processing (NLP) and text-to-speech (TTS) technologies into an application for text generation and vocalization. The proposed idea is designed to create a coherent and relevant weather report and turn it into a natural voice. The communication from Ollama NLP model to Coqui TTS model is facilitated through Nodejs server. A NestJS user interface is implemented to demonstrate the system's results, with WebSocket to ensure real-time data transfer and visualization. This allows the user to observe both the weather information and listen to the speech simultaneously at a customizable time.

Both models are fine-tuned on Google Colab using custom datasets. The NLP models are trained on scripts related to weather, while the TTS model is customized with data collected from real humans. The project is a simple yet fully-automated web application that combines AI models with web technologies. The design not only ensures the functionality but also prevents unnecessary distraction and encourages the end user to pay more attention to their mental health.

Key words: Natural Language Processing, Text-to-speech, Weather reporter, Web development, NextJS, Nodejs.

1. INTRODUCTION

1.1. Context and Motivation

Morning routines shape day-long performance, yet the first thing people touch is often a smartphone lock screen crowded with overnight email or app alerts. Weather information is often displayed on the screen and this design can exhaust attention energy at the time users have the least cognitive bandwidth, it is also associated with degraded task focus[1]. In Hanoi or similar motorbike-heavy cities[2], sudden rains can disrupt commutes, making a missing raincoat a crucial mental weight.

Television weather segments deliver high-quality information but require the user to be at a screen on a broadcaster's schedule. Mobile weather apps excel at convenience, however they still depend on manual checking and visual attention. General-purpose voice assistants (Siri, Google Assistant or Alexa)[3][4] can read weather reports on demand and scheduled announcements-however these typically require manual setup and not streamline across devices. As a result, many users either delay the checking (and get caught by weather surprises) or check their phones and become sidetracked with unrelated notifications.

Three elements making a different approach possible are: efficient and integratable language model fine-tuned for niche domain; wide range of TTS models delivering natural accents at low latency and progressive web app (PWA) capabilities enabling a cross device experience without the need for platform-native apps. Another factor contributing to this project is WebSockets, it is a practical solution to deliver AI models responses without latency.

This thesis proposes a "digital weatherman" that converts weather API data into spoken briefings at user-defined times (e.g., wake-up). Technically, it combines (a) fine-tuned text generation using forecast data, and (b) low-latency neural

TTS, deployed as a PWA for universal access. This fills the gap between passive, visual apps and general assistants by offering zero-interaction weather forecasts, measurable on latency, accuracy, and user outcomes.

1.2. Objectives

This project aims to deliver an interaction-free morning weather forecast that prepares users for the day's weather. The primary goal is to reduce morning screen time after wake-up while improving preparedness for weather impact. At the same time, this is also an opportunity for me to learn and apply knowledge about modern web technologies and Artificial Intelligence altogether in one project and get more insight for the improvement of this project in the future.

The model's performance will be evaluated using established metrics such as ROUGE-1, ROUGE-2, ROUGE-L, and BERTScore to assess its precision, recall, and semantic fidelity. Moreover, the project will explore effective preprocessing techniques —including data collator and masking.

From the user's perspective, the app offers both a quick visual and a scheduled spoken forecast. For more information, the app will display daily weather information. By simply changing the time, it ensures that the user can prepare for the weather during daytime or anytime of the day before they get out of their shelter. When the time comes, the app will send a push notification and ring the user's phone as well as read out the weather forecast at a hearable rate. For privacy purposes (e.g. public spaces), the user has the choice to mute the voice in the settings section.

The NLP component must transform the information taken from WeatherAPI into a 15- seconds script. The output must retain all necessary information like description temperature, rain probabilities, wind speed, humidity, cloud level and has above 0.5 for both ROUGE and BERTScore metrics. This script will then be delivered to the TTS models and synthesized into natural, intelligible speech in

English. The system will also be streaming data through WebSockets and is expected to have a 20-second latency from the time it fetches API to the speech. The system is considered successful if it meets or exceeds the latency and accuracy thresholds above. Further improvement for the program and testing for reductions in morning distraction will be implemented in the future.

1.3. Methods

Over the years, Natural Language Processing (NLP) has gained significant popularity, particularly in the area of text generation. Early methods for text generation relied on relatively straightforward statistical approaches, such as rule-based systems and Hidden Markov Models (HMMs)[5]. These systems depended on predefined grammatical rules, templates, and linguistic structures crafted by human experts. Words or phrases were selected according to syntactic or semantic rules to fill placeholders within templates. While such approaches were simple and easy to implement, their outputs often lacked contextual relevance and long-range coherence, resulting in repetitive or unnatural text.

With the rise of deep learning, text generation has undergone a fundamental shift toward neural-based methods, especially causal language models[5]. These models operate in an autoregressive manner, predicting the next token in a sequence based on all previously generated tokens, which enables them to capture contextual dependencies more effectively than traditional statistical methods. Among the most well-known causal language models is GPT-2, which demonstrated that large-scale pretraining on diverse text corpora can produce coherent, contextually relevant, and human-like text across a wide range of domains.

In this project, a 2-stage approach will be applied for building the Digital weatherman:

1. Fine-tuning a Causal Language Model with LoRA

In the first stage, a pre-trained GPT-2 model is fine-tuned using Low-Rank Adaptation (LoRA) to generate natural language weather reports from JSON inputs. This enables the model to efficiently adapt to the domain of weather data while maintaining coherence and contextual meaning.

2. Text-to-speech audio generation

In the second stage, the generated text is converted into speech using a pre-trained TTS model. This ensures that the final output is delivered in an intelligible and natural-sounding voice, which limits as much phone interaction as possible.

1.4. Report Structure

This report provides an overview of the project, going through the development process, methodology and the current achieved results. The structure of the report is as follows:

Section 1 Introduction provides the project background, objectives, expected outcomes, and summary of text generation.

Section 2 Background describes the design and deployment of the two models.

Section 3 Materials and Methodology describes the dataset preparation, model development, summarization techniques, and evaluation metrics.

Section 4 Results and Discussion presents model results, challenges and setbacks.

Section 5 Conclusion and Perspective summarizes the project and suggests improvement for the application.

2. BACKGROUND

2.1 Language Models

2.1.1 Causal Language Model

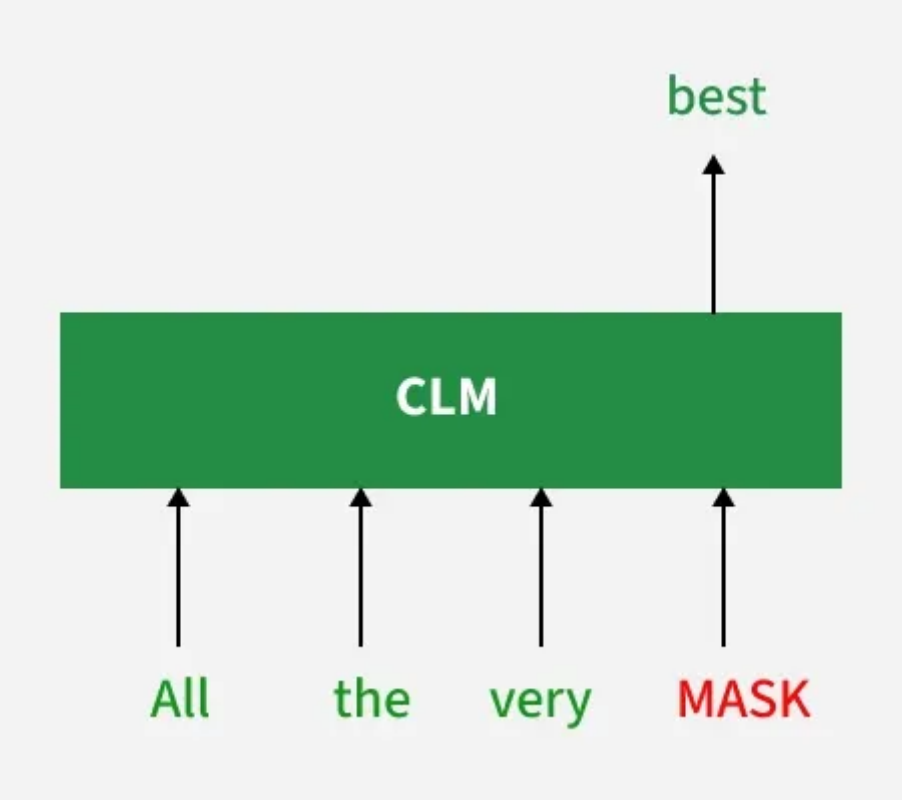

Figure 1: Example of how a causal language model works (GPT2)

(*Source: GeeksforGeeks)*

A Causal Language Model (CLM) is a type of neural network that predicts the next word in a sequence based on the preceding context. This autoregressive property makes it especially effective for tasks such as text generation and script writing. In this project, I used the **GPT2LMHeadModel**, a variant of the GPT-2 architecture provided by the Hugging Face Transformers library. The GPT2LMHeadModel extends the base GPT-2 by adding a language modeling head on top of the transformer decoder, enabling it to compute next-word probabilities and generate coherent text outputs and an efficient integration with a small set of data.

2.1.2 Sequence-to-sequence Language Model

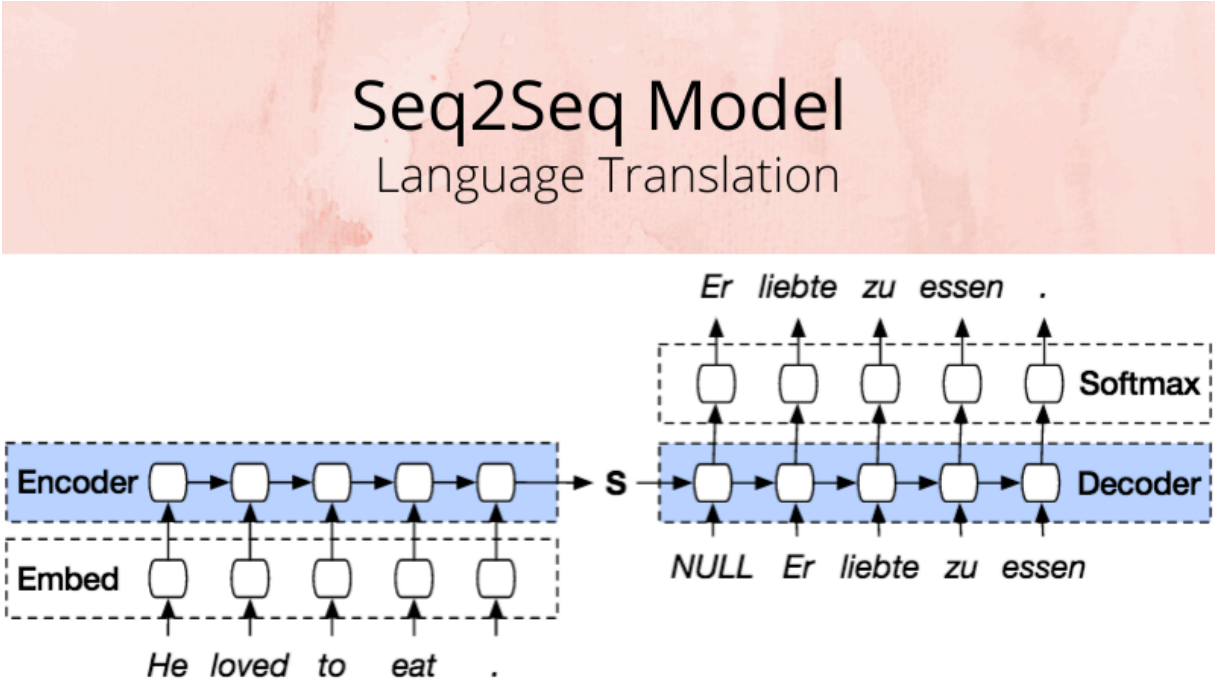

Figure 2: Example of how a Sequence-to-sequence model works

(*Source: Medium)*

A Sequence-to-sequence (or Seq2seq) is a language model architecture suitable for translation tasks and is bidirectional[13]. Aside from predicting the next token in a sequence, it also looks at the previous ones to ensure context consistency. Although not directly implemented into this project, theoretically, it excels in json-script processing thanks to it being a translational task, further testing will be implemented in the future to evaluate the performance of each model in practice with the entire application.

2.2 Dataset Preparation

OpenWeatherMap not only provides real-time weather information but also offers access to **historical data archives** and the possibility of making **bulk data calls**, which can be valuable for constructing larger and more diverse training datasets in future work.

A promising resource for future improvement is **Gemini 2.5 Flash Lite**, which has demonstrated strong performance in text generation tasks. Its capabilities make it suitable for supporting or complementing the fine-tuning of the ScriptGen model, particularly in generating diverse and stylistically rich training data. In addition, the Gemini API provides generous free access in terms of both request limits and token[7] allowances, making it a practical and cost-effective tool for experimentation during development.

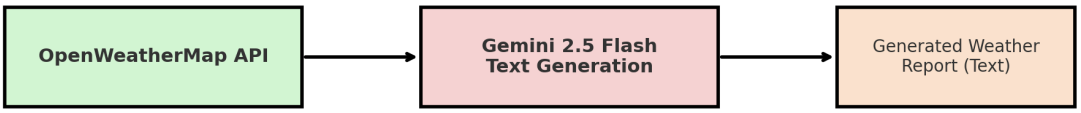

Figure 3: Data generation pipeline

For the fine-tuning of the GPT-2 model, a program was developed to automatically generate the dataset required for training. This program interacts with the **OpenWeatherMap API** to retrieve real-time weather data. The API responses provide structured JSON objects containing six essential attributes: *description, temperature, humidity, rainfall, wind speed,* and *cloud coverage*. These attributes capture the key features of a local weather condition in a machine-readable format. Once collected, each JSON input is then passed to **Gemini 2.5 Flash**, which generates a natural language weather report through a carefully designed prompt. This ensures that each weather condition is paired with a corresponding descriptive script that mimics how a human forecaster might present the information.

input	generated_output
{"description": "scattered clouds", "temp": 27.58, "humidity": 87, "rain": 0.0, "wind_speed": 3, "clouds": 40}	Good day, everyone! Today's forecast paints a picture of intermittent cloud cover, with temperatures hovering around a pleasant 27 degrees Celsius. It's quite humid out there, so keep that in mind. We're not expecting any precipitation, and the wind is a gentle breeze. Given these conditions, why not embrace the dappled sunlight and enjoy a leisurely afternoon exploring a botanical garden?

Table 1: Example dataset format

The final dataset is stored in a **TSV (tab-separated values) file** for formatting purpose with two columns: *(input, output)*. The *input* column holds the weather JSON, while the *output* column contains the generated weather script. This format is simple and consistent to be fed into machine learning pipelines. By creating data pairs in this way, the GPT-2 model can effectively learn to map structured numerical and categorical weather data into fluent, contextually appropriate text. This automated approach ensures scalability, diversity of samples, and high-quality training material for fine-tuning.

2.3 Low-rank Adapter (LoRA)

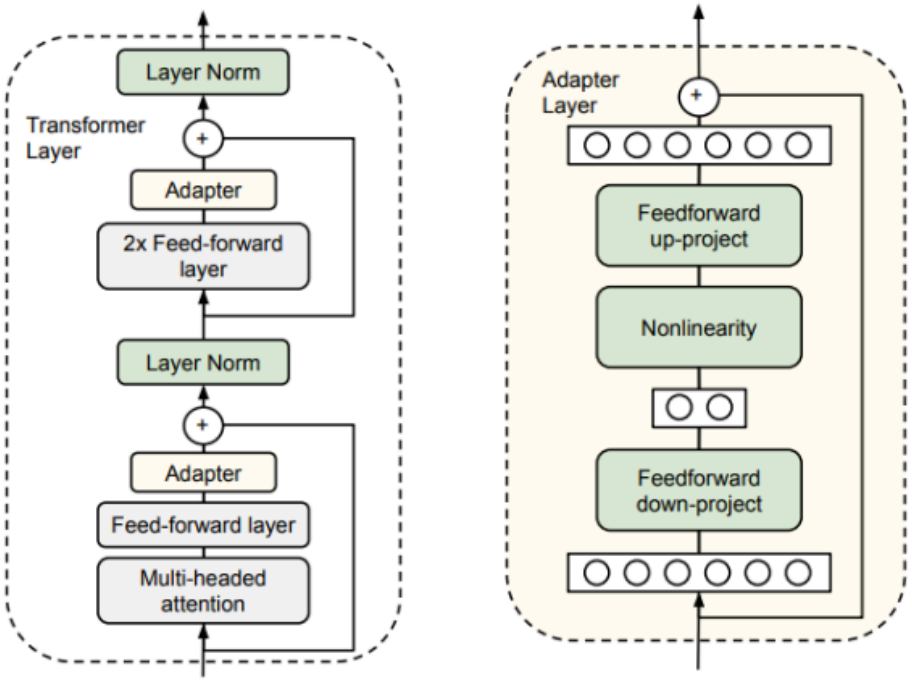

Figure 4: Example of LoRA

(Source: viblo.asia)

Low-Rank Adaptation (LoRA)[6] is an efficient fine-tuning technique that reduces the number of trainable parameters required when adapting large language models. Instead of updating all the weights in the original GPT-2 model, LoRA introduces small, low-rank trainable matrices into certain layers (usually attention projections such as query/key/value or feedforward layers). During training, only these injected matrices are updated, while the pre-trained weights of GPT-2 remain frozen.

Mathematically, a large weight matrix **W** is decomposed into two smaller matrices **A** and **B** of rank r (with r much smaller than the full dimension). The update is expressed as:

$$W' = W + \Delta W \ where \ \Delta W = A \cdot B$$

This significantly reduces memory usage and training time while still allowing the model to learn domain-specific adaptations (e.g., weather information context).

When applied to **GPT2LMHeadModel**, LoRA modules are injected into the model's attention and projection layers. During fine-tuning, only these LoRA parameters are updated. The result is still a **GPT2LMHeadModel**, but now enhanced with LoRA adapters that specialize it for the target dataset.

2.4 Data preprocessing

Formatting data

prompt	target
{"description": "broken clouds", "temp": 11.47, "humidity": 88, "rain": 0.0, "wind_speed": 2.32, "clouds": 51} **<OUT>**	Good day everyone! Looks like we've got a bit of a mix out there today,...
{...}	...

Table 2: Formatted data

In this project, the model receives weather information json as input and generates the script as the output respectively. In order for the model to do so, the data must be formatted so that the GPT2 model can differentiate between input and output. By adding a token ***<OUT>***, it ensures that during the training process, the model can clearly learn and map the json input with the script.

Masking in Causal LM (GPT-2)

input_ids	labels	attention_masks
[4895, 11213, 1298, 366, 25826, 15114, 1600, 366, 29510, 1298, 2808, 13,..., **198, 50257, 198,** 10248, 1110, 2506, 0, 29403, 588, 356, 1053,...]	[-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100 ,..., **-100, -100, -100,** 10248, 1110, 2506, 0, 29403, 588, 356, 1053,...]	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,..., **1, 1, 1,** 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,..., 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

Table 3: Masked data

The next step of pre-processing involves turning all the words including json input and script output of the dataset into numbers in the *input_ids* column, since language models learn with numbers mapped to words, not actual words. Next, all of the input is labelled as *-100* so that the model only focuses on learning the output. GPT-2 is a causal language model, this means it learns by predicting the next token given all previous tokens. To ensure the model only attends to past tokens and not future ones, in the last step, a causal attention mask is applied. Instead of replacing tokens, this mask ensures that at position t, the model can only see tokens ≤ t. This maintains autoregressive next-word prediction.

Data Collator

[10,20,30,40,50]
[5 , 6 , 7 , 0 , 0]

Table 4: Example of a padded (below) and masked (above) data cell.

In this project, instead of using the default DataCollatorForLanguageModeling provided by Hugging Face, a custom collator named **SimpleCollator** was implemented. The purpose of a collator is to process raw samples from the

dataset and combine them into a properly padded batch for training. In the table above, the below cell is padded with *0s* so that it matches the length of the above cell. This ensures that all sequences within the same batch share the same length and prevents any delays or errors during runtime.

2.5 Text-to-speech with CoquiTTS

Text-to-Speech (TTS) technology aims to convert written text into natural-sounding speech. Modern TTS systems rely on deep learning architectures that model both the linguistic content and the acoustic features of speech, producing outputs that approach human-like naturalness and intelligibility. Earlier approaches, such as concatenative or parametric synthesis, often resulted in robotic or unnatural audio, but advances in sequence-to-sequence models and neural vocoders (e.g., Tacotron, WaveNet) have significantly improved voice quality.

CoquiTTS is an open-source deep learning toolkit specifically developed for TTS research and deployment. Originating as a fork of Mozilla's TTS project, it provides a wide range of pre-trained models, multilingual support, and flexible training pipelines. One of its key strengths is accessibility—it enables researchers and developers to fine-tune models on custom voices, experiment with prosody control, and integrate TTS into downstream applications without requiring extensive infrastructure.

For this project, CoquiTTS was chosen because it offers state-of-the-art pretrained models such as **XTTS v2**, which support multi-lingual voice cloning and synthesis. This made it possible not only to generate intelligible speech but also to adapt the system to mimic a specific reference voice thus forms the backbone of the TTSGen component, bridging the gap between the textual output of ScriptGen and the final audio experience delivered to the user.

3. MATERIALS AND METHODS

In order to create a powerful model tailored for script generation, we need to perform a sequence of data processing, fine-tuning and evaluation. This section will go through all of the process of fine-tuning a model- data preparation, experimentation, ... The final result is a fine-tuned GPT2-based model specialize in weather report script generation.

3.1. Required Materials

To replicate and extend the results of this project, it is essential to establish a suitable computational environment that balances accessibility with performance. The materials outlined in this section ensure that the development, training, and testing of the models can be carried out effectively without imposing excessive hardware or software demands. By specifying these requirements, the project provides clear guidance for researchers, students, or practitioners who wish to reproduce the experiments or adapt the system for their own use. The focus is on tools that are widely available, affordable, and compatible with modern deep learning workflows, making the project both practical and reproducible.

3.1.1 Hardware Requirement

To replicate the project presented in this thesis, a modest yet capable computing environment is required. The essential tool is a laptop equipped with a decent GPU that supports CUDA technology, as GPU acceleration plays a critical role in handling the computational demands of training and running deep learning models. CUDA compatibility ensures faster execution compared to CPU-only setups, which would otherwise make experiments slow and impractical. The laptop should also be able to run Visual Studio Code with Python, serving as the primary development environment. VSCode provides debugging, version

control, and seamless integration with Python libraries, making it suitable for coding, managing scripts, and testing models. For users without access to a CUDA-capable laptop, Google Colab offers a reliable alternative. Colab provides free cloud GPUs, a preconfigured Python environment, and direct integration with Google Drive, ensuring experiments can still be reproduced effectively and efficiently.

For this project, the Language Model was trained using **Google Colab Pro**, which provided access to an **NVIDIA L4 GPU**..

3.1.2 Hosting Services

Hugging Face provides a versatile hosting service called **"Hugging Face Spaces"** that enables researchers, developers, and organizations to easily share and deploy machine learning models. Through the Hugging Face Hub, models can be uploaded, versioned, and accessed by the community or kept private for internal use. This allows users to showcase models through web applications without the need for complex server management. Additionally, Hugging Face hosting can be integrated with popular machine learning libraries, including PyTorch, TensorFlow, and Transformers, ensuring straightforward deployment and inference. A special requirement to make this possible is some understanding with Docker to set up the environment (*Appendix* 1).

Vercel and **Render** are two popular choices to host the Front-end and Back-end of the application thanks to their automated CI/CD pipeline and intuitive interface.

3.2. Fine-tune GPT2 with LoRA Adapter

To improve the model's performance in generating coherent and domain-specific weather reports, a fine-tuning process was applied to the **GPT2LMHeadModel** with the integration of a **Low-Rank Adaptation (LoRA)** adapter[6]. From this

point onward, the fine-tuned system is referred to as the **ScriptGen Model**. The fine-tuning pipeline consisted of several essential stages. First, the LoRA adapter was configured to inject additional trainable parameters into the GPT-2 architecture, focusing on its attention and projection layers. Next, the dataset was reformatted so that each weather entry was represented as a structured JSON input followed by the special <OUT> token (in **2.3 Data Preprocessing**), which clearly marked the transition from input data to the expected natural language output. The corresponding target script served as the ground-truth label for the model. Once this structure was established, the sequences were tokenized and padded, and masking[9] was applied to ensure that padding tokens did not influence the loss calculation. A custom data collator was then used to assemble batches dynamically, aligning inputs, labels, and attention masks for training.

Parameter	**Value**
Task Type	Causal LM
Rank (r)	8
LoRA Alpha	32
LoRA Dropout	0.05
Target Modules	["c_attn", "c_proj"]
Bias	none

Table 5: LoRA Configuration

The LoRA adapter configuration is summarized in *Table 5*. This setup specified causal language modeling as the task type, with a rank of 8, an alpha value of 32, and a dropout probability of 0.05. The adapters were injected into the *c_attn*

and *c_proj* modules, while bias terms were left unchanged. These settings enabled the model to adapt effectively to the weather domain effectively without re-training the whole GPT2 model.

Parameter	**Values (Previous)**	**Values (Recent)**
Number of datasets (rows)	1000	2000
Max new tokens	220	100
Early stopping	No	Yes
Number of Training Epochs	5	12
Train Batch Size per Device	2	4
Eval Batch Size per Device	2	8
Gradient Accumulation Steps	4	4
Learning Rate	2e-4	2e-4
Warmup Ratio	0.03	0.05
Logging Steps	20	20

Table 6: Parameters for fine-tuning GPT2

In addition to configuring LoRA, training parameters were carefully selected to balance convergence speed and resource efficiency. *Table 6* indicates that this

model was trained with 2 sets of parameters to test how the amount of dataset will affect the output.

In previous training, the model was trained with **1000 rows** of dataset and for **5 epochs**, with both training and evaluation batch sizes set to **2**. Gradient accumulation was set to **4 steps**, effectively simulating a larger batch size while remaining within the memory limits of available hardware. A **learning rate of 2e-4** was applied, together with a **warmup ratio of 0.03**, to stabilize early updates during training. Logging occurred every **20 steps** to provide regular feedback on the model's progress. The output for this training is set to 220 new tokens with no early stopping.

On later runs, the model was trained with higher amounts of parameters, notably at the amount of dataset train and evaluation batch size and warm up ratio, other parameters such as gradient accumulation, learning rate and logging steps were retained. Moreover, early stopping is implemented to generate less filler words.

This combination of LoRA configuration and training hyperparameters allowed the ScriptGen Model to specialize in the language and structure of weather forecasting. By introducing lightweight adapters instead of retraining the entire model, the approach achieved strong adaptability while maintaining computational efficiency, the result for each set of hyperparameters will be discussed in *Section 4: Result and Discussion*.

3.2.1 Model Evaluation

The ScriptGen Model has been evaluated with standard, widely used natural language processing evaluation metrics: ROUGE-1. ROUGE-2, ROUGE-L, ROUGE-Lsum, BERTScore F1

BERTScore (Bidirectional Encoder Representations from Transformers Score)

BERTScore[10] is a semantic similarity metric that evaluates the quality of generated text by comparing it to reference text using contextual embeddings from a pre-trained BERT model. Unlike surface-level n-gram overlap metrics, BERTScore captures semantic meaning by computing pairwise cosine similarity between token embeddings. For each token in the candidate sentence, the most similar token in the reference sentence rrr is identified. Based on this, BERTScore calculates precision, recall, and F1:

$$P = \frac{1}{|c|} \sum_{x \in c} \max_{y \in r} \cos(e(x), e(y)) \qquad R = \frac{1}{|r|} \sum_{y \in r} \max_{x \in c} \cos(e(y), e(x))$$

$$F_1 = \frac{2PR}{P+R}$$

where $e(x)$ and $e(y)$ represent contextual embeddings of tokens, and $cos(\cdot)$ is cosine similarity. BERTScore-F1 is typically reported as the final evaluation metric.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

ROUGE[11] is a family of metrics that measure the overlap between generated text (candidate) and reference text in terms of n-grams, word sequences, or longest common subsequences. ROUGE focuses on surface-level textual similarity, which is especially useful in measuring lexical fidelity.

$$ROUGE - N = \frac{\sum_{gram_n \in Ref} \min(Count_{cand}(gram_n), Count_{ref}(gram_n))}{\sum_{gram_n \in Ref} Count_{ref}(gram_n)}$$

where $gram_n$ denotes an n-gram, and counts are computed in the candidate and reference text.

ROUGE-1 and ROUGE-2

ROUGE-1 measures the overlap of unigrams (individual words). It is sensitive to vocabulary usage and lexical choice. ROUGE-2 measures the overlap of bigrams (pairs of consecutive words). This adds a measure of fluency and phrase-level coherence beyond word-level similarity.

Both are calculated using the ROUGE-N formula above, with $n = 1$ for ROUGE-1 and $n = 2$ for ROUGE-2.

ROUGE-L and ROUGE-Lsum

ROUGE-L evaluates similarity based on the **Longest Common Subsequence (LCS)** between candidate and reference text. Unlike fixed n-grams, LCS allows for flexible matching that respects word order without requiring strict consecutiveness. It computes precision, recall, and F1 as:

$$R_{LCS} = \frac{LCS(c, r)}{|r|}$$

$$F_{LCS} = \frac{(1 + \beta^2) \cdot R_{LCS} \cdot P_{LCS}}{R_{LCS} + \beta^2 \cdot P_{LCS}}$$

$$P_{LCS} = \frac{LCS(c, r)}{|c|}$$

where $LCS(c, r)$ is the length of the longest common subsequence between candidate c and reference r, and β is typically set to give higher weight to recall.

ROUGE-Lsum extends ROUGE-L to multi-sentence evaluation by aggregating LCS-based scores across all reference sentences. It is particularly useful for evaluating longer texts or summaries where sentence segmentation plays a role.

3.3. Pre-trained Coqui XTTSV2

Coqui XTTSV2[12] is a pre-trained **text-to-speech (TTS) model** designed to convert written text into human-like speech. In this project, it was employed to generate audio files from scripts produced by the fine-tuned GPT-2 model, thereby forming a complete pipeline that transforms structured weather data into natural-sounding spoken forecasts. To further personalize the generated audio, the model was adapted to use my own voice as a reference, creating a more authentic and user-specific listening experience. For clarity within this thesis, the system built on Coqui XTTSV2 will be referred to as the **TTSGen Model**.

3.3.1 Model Evaluation

The TTSGen Model is capable of producing audio output that resembles a real, intelligible human voice. In most cases, the generated speech is natural and easy to understand, with appropriate pacing and pronunciation. Nevertheless, during evaluation, it was observed that the output occasionally exhibits a “robotic” tone. This effect, while not frequent, highlights the limitations of current pre-trained TTS models when fine-tuned for voice cloning. It suggests that while the model is effective for generating personalized audio, further optimization or the use of higher-quality voice datasets may be required to consistently achieve fully natural prosody and timbre.

4. RESULTS AND DISCUSSION

This section presents the outcomes of both the model evaluation and the implementation of the web application. First, we will discuss the experimental results of the fine-tuned GPT2 model, including relevant performance metrics for both previous and recent runs. After that, we will evaluate the behavior and functionality of the web interface. Finally, the section outlines key challenges encountered throughout the project as well as its current limitations.

4.1 Results

4.1.1 Fine-tune GPT2 model (ScriptGen)

To evaluate the performance of the summarization model, a variety of metrics, including ROUGEscores, BERTScore, and evaluation loss have been utilised. These metrics provide both lexical and semantic assessments of the model's output quality.

Metric	**Values (Previous)**	**Values (Recent)**
Validation loss	1.931273	1.5482
ROUGE-1	0.3497	0.3218
ROUGE-2	0.0922	0.0475
ROUGE-L	0.2193	0.1843
ROUGE-Lsum	0.2193	0.1843
BERTScore (Precision)	0.6048	0.5882

BERTScore (Recall)	0.5834	0.5758
BERTScore (F1)	0.5935	0.5815
Generated text	Good day to everyone! It's a rather damp atmosphere out there today with the air feeling quite moist and humid at 89% humidity; while it feels somewhat muggy in terms of moisture meticortunately you'll have no rain on your list as we're seeing just one cloud cover per hour." The sky is completely covered by scattered patches that are almost touching around 25%. Given this persistent drizzle they might be an ideal time for some indoor activity or perhaps even try something new indoors – maybe enjoying our cozy kitchen decorating those days?	Greetings! The sky is draped in a thick blanket of scattered clouds today with temperatures hovering around 21 degrees Celsius and humidity at 92%. There's no rain expected, and the wind is gentle at 0 kilometers per hour; perhaps it would be perfect for a leisurely stroll through a quiet forest or maybe even try out some indoor baking recipes indoors

Table 7: Evaluation for ScriptGen model

The evaluation results of the ScriptGen model are presented in *Table 7*.

Between the two training runs, the model's **validation loss decreased from 1.93 to 1.55**, indicating that the models learn the dataset better. However, evaluation metrics such as **ROUGE-1, ROUGE-2, ROUGE-L, and ROUGE-Lsum** showed slight declines, suggesting reduced lexical and structural overlap with the reference outputs. Similarly, **BERTScore precision, recall, and F1** experienced minor decreases, reflecting a negligible drop in semantic similarity.
The recent run was implemented with early stopping, and it only generates until it reaches **100 tokens** or **3 sentences limit**. Whereas for the previous one, it tried to fill up **220 tokens**, which ended up being gibberish.
Overall, while the recent model demonstrates improved convergence and lower loss, it produces outputs that are slightly less similar to the reference scripts, highlighting a trade-off between optimization performance and linguistic fidelity. These results suggest that the ScriptGen model is capable of generating intelligible and partially accurate weather reports if it is exposed to more data. It demonstrates stronger performance at the word level and semantic relevance than at the phrase or sentence level. These outcomes meet the expectations for fine-tuned GPT-2 models on relatively small or domain-specific datasets, and it highlights directions for improvement, such as expanding training data or applying other fine-tuning strategies..

4.1.2 Text-to-speech model (TTSGen)

To evaluate the performance of the TTSGen model, since this pretrained model is robust, subjective listening tests were employed rather than purely numerical metrics. Participants were asked to provide their opinions on the generated audio. In general, the model was able to produce speech that was clear and intelligible, resembling a real human voice (me). However, listeners also noted occasional artifacts, such as robotic tone, which reduced the perceived naturalness.

Although no standardized objective score such as MOS (Mean Opinion Score), STOI (Short-Time Objective Intelligibility), or FAD (Fréchet Audio Distance) was computed due to deployment limitations, the subjective feedback demonstrates that the TTSGen model is functional and capable of producing usable outputs. This confirms its feasibility for integration into the overall system, while highlighting the need for further optimization and more rigorous evaluation in future work.

4.2 Deployment

The app is named **Digital Weatherman** for simplicity.

4.2.1 System Design

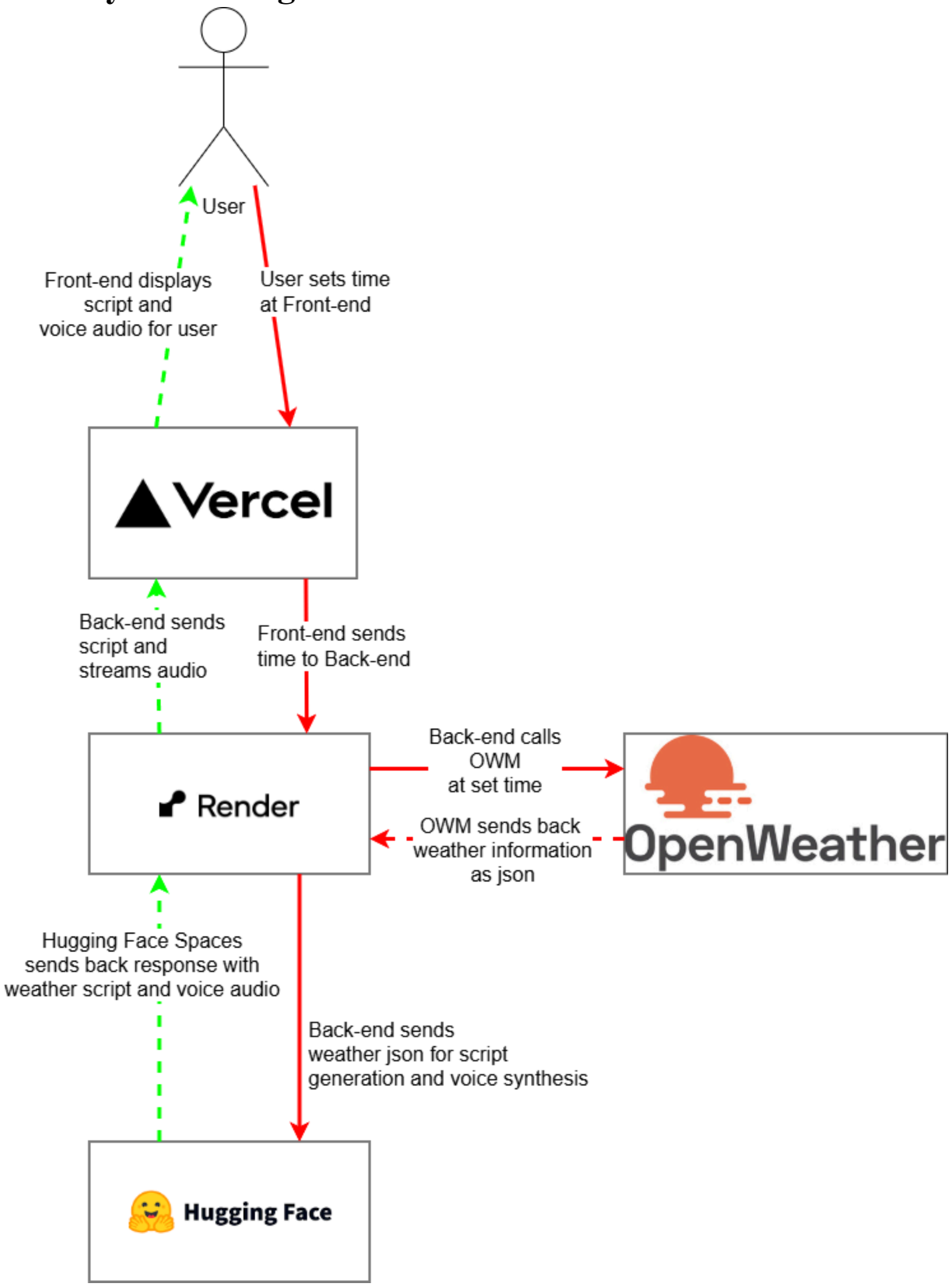

Figure 5: Application pipeline

This pipeline demonstrates the flow of this product, in general, the user will set a time and receive a script displayed on the app Front-end.

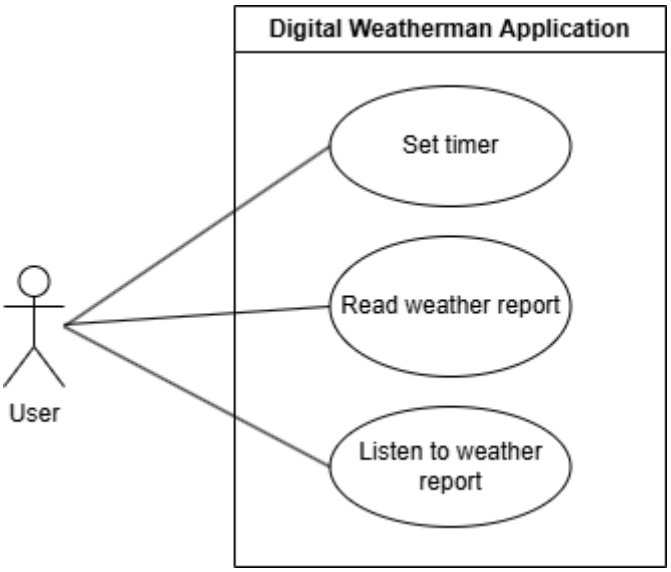

Figure 6: Use Case diagram

This Use Case diagram displays what the user can do in the Digital Weatherman Application, overall, there are 3 interactions with this app including: setting timer for the application to send script, as well as reading and listening to the weather report.

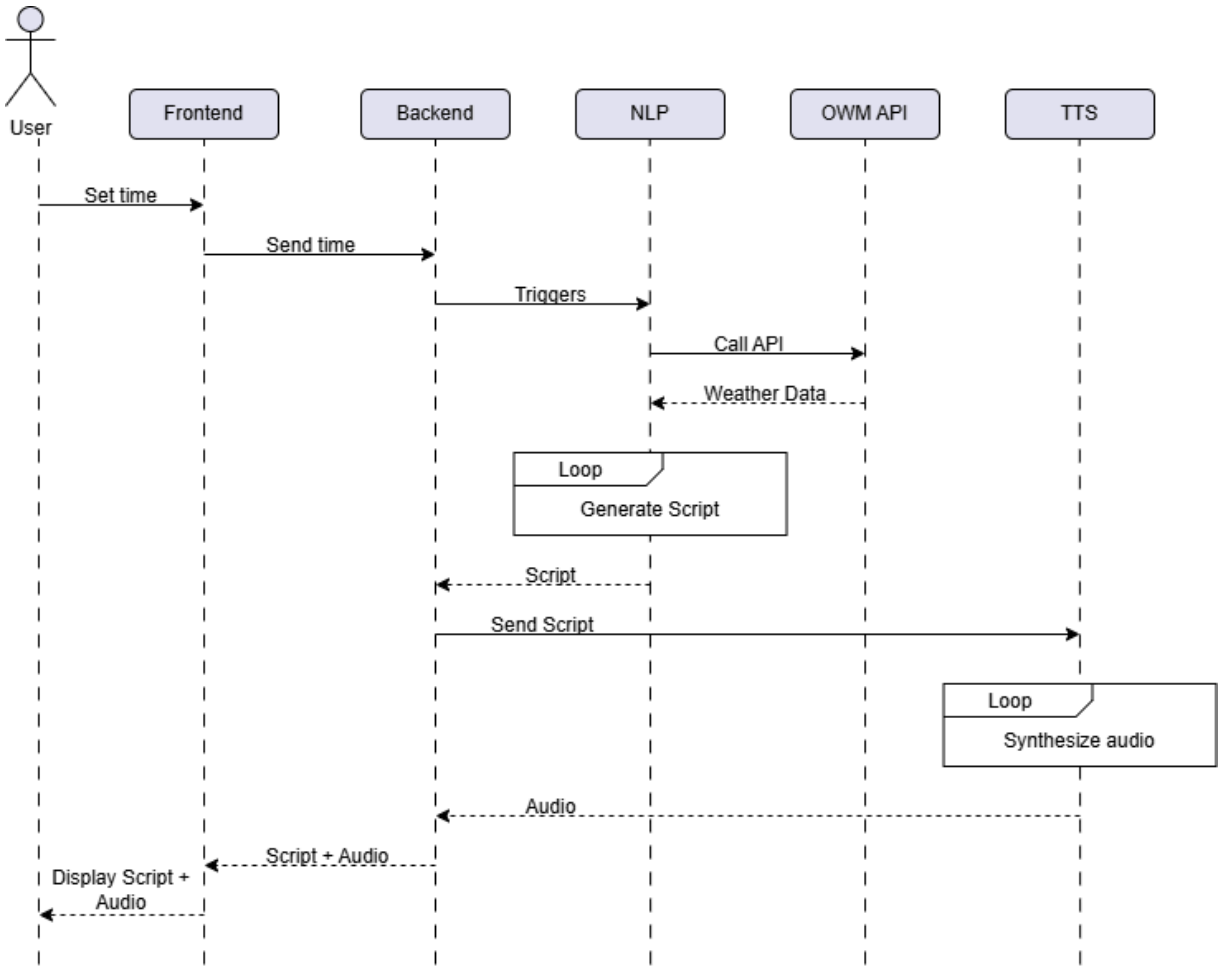

Figure 7: Sequence Diagram

The sequence diagram above visualizes the processes of each component from the time the user set. First, the user sets a time on the frontend, which triggers the backend to request weather data from the NLP service and OpenWeatherMap API. The NLP service generates a script, which is sent to the TTS service for audio synthesis. The backend returns both script and audio to the frontend, where the browser displays the text and plays the audio directly.

4.2.2 Hosting Services Deployment

All the machine learning models including the ScriptGen (NLP) Model and TTSGen (TTS) Model are hosted using **Hugging Face Spaces (HFS)**. In short, both models are pushed on their Hugging Face repositories and called through a Python file called *app.py* that uses Hugging Face native libraries and FastAPI-a

library to create endpoints and test environments with a website interface. HFS also requires a Dockerfile to install dependencies and set up a host environment, below is a demonstration for the NLP model *(Appendix 1)*.

For the Front-end and Back-end their code are pushed on separate **Github** repositories and connected to the account of the respective service (**Vercel and Render**). Along with that, all the environment variables must be declared. A side note is that on the free tier of these services, the application might be winded down after some inactivity to save their cloud resources and will take around 1 minute to be booted up, this boot time will not be taken into account for the expected product.

During the deployment stage, difficulties arose when attempting to expose the TTS model as a production endpoint, which prevented the integration of both models into a fully functional application within the project expectation. However, both the ScriptGen (NLP) and TTSGen (TTS) models were successfully trained and evaluated independently on Google Colab, demonstrating that the pipeline is technically possible. This indicates that the deployment challenge can be resolved and a complete product can be realized in the future using other hosting services such as AWS or Google Cloud.

4.3 Discussion

4.3.1 Challenges

Insufficient data

One of the main limitations of this project lies in the insufficient amount of training data available for fine-tuning the ScriptGen model. As a result, while the model is able to generate partially coherent weather reports, the outputs often contain factual inaccuracies. Moreover, despite having been fixed by limiting the number of sentences the model can generate, occasionally, the model still generates nonsensical text. This behavior is consistent with the known

challenges of training large language models on small, domain-specific datasets, where the model tends to overfit to limited patterns and fails to generalize.

Despite these shortcomings, the results still demonstrate that the proposed pipeline is technically feasible. With a larger and more diverse dataset, it is expected that the ScriptGen model would improve in both factual accuracy and linguistic coherence, thereby producing weather reports that are both reliable and natural-sounding.

Hugging Face hosting service

During deployment, it was discovered that the Hugging Face Spaces hosting service could not be used to reliably host the TTSGen model. This limitation arose because the XTTSV2 model requires large dependencies, GPU resources, and low-level audio processing libraries that are not fully supported in the default Hugging Face container environment. As a result, while the ScriptGen model could be deployed successfully, the TTS component could not be exposed as an endpoint within the same infrastructure.

This constraint prevented the demonstration of a fully integrated application on Hugging Face. However, since both the ScriptGen and TTSGen models are functional when run locally or in development environments, the overall pipeline remains feasible. With access to more flexible hosting environments—such as dedicated cloud GPUs, Dockerized deployment on a service like AWS or GCP, or optimized model compression—the full system can be deployed in practice.

4.3.2 Setbacks

This project encountered two main setbacks that constrained the final outcome.

The first relates to the **availability of training data**. Weather report scripts are not publicly released in large, structured datasets, which require manual data crawling from online sources. As a result, the ScriptGen model was trained on a

relatively small and domain-specific dataset. This limitation reduced the model's ability to generalize, leading to partially coherent reports but also factual inaccuracies or nonsensical outputs toward the end of longer generations. Such behavior is consistent with the known challenges of fine-tuning large language models on insufficient data. Nonetheless, the results confirm that the pipeline is technically feasible, and with a larger, more diverse dataset, ScriptGen is expected to achieve greater factual accuracy and linguistic fluency.

The second setback occurred during the **deployment phase**. While the ScriptGen model could be hosted successfully on Hugging Face Spaces, the TTSGen model (based on XTTSV2) could not be deployed due to its heavy GPU requirements, large dependencies, and reliance on low-level audio processing libraries not supported in the Hugging Face environment. This limitation prevented the demonstration of a fully integrated web application. However, since both models run independently in local and development settings, the pipeline remains viable. With access to more flexible hosting options, such as dedicated cloud GPUs, Dockerized deployment on AWS or GCP, or optimized model compression, the complete system can be deployed in future work.

Importantly, these setbacks do not undermine the research contribution of this thesis. Despite the deployment constraint and limited dataset, both models were able to run and meet expectations, thereby demonstrating the feasibility of the proposed system while also highlighting practical challenges that future work can address.

5. CONCLUSION AND PERSPECTIVE

5.1 Conclusion

This thesis explored the design and implementation of a pipeline for generating spoken weather reports by combining natural language generation and text-to-speech techniques. The ScriptGen model, a fine-tuned GPT-2 language model, demonstrated the ability to produce coherent weather report scripts, while the TTSGen model, based on Coqui XTTSV2, successfully converted these scripts into intelligible speech. Despite setbacks related to limited training data and deployment challenges, both models were able to run and meet expectations, thereby confirming the technical feasibility of the proposed system. The application interface and back-end also surpassed the expected performance of generating text only 5 seconds after the timer goes off.

5.2 Perspective

Looking ahead, several directions can extend and strengthen this work. The most immediate perspective is to **deploy the system as a mobile application**, further leveraging the capability of the app. The quality of speech synthesis can be enhanced through **more natural voices**, either by adopting advanced TTS architectures or by fine-tuning with higher-quality voice datasets. Expanding to **other languages** would broaden accessibility and make the system relevant for a more diverse audience.

From a user-experience standpoint, future versions of the application could feature a **customizable interface** to adapt to individual preferences, along with **user authentication (login and registration)** to enable personalization and history tracking. Furthermore, extending the service to **more locations** would improve its utility by providing forecasts not only for a single area but also for multiple regions of interest. Together, these perspectives outline a pathway for

evolving the prototype into a robust, scalable, and user-friendly product that integrates artificial intelligence, weather forecasting, and personalized digital experiences.

REFERENCES

[1] Bâce, M., Staal, S., & Bulling, A. (2020). *Quantification of users' visual attention during everyday mobile device interactions*. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (pp. 1–14). Association for Computing Machinery. https://doi.org/10.1145/3313831.3376449

[2] Le, T., Pojani, D., Nguyen, T., Ha, T. T., & Nguyen, M. H. (2024). *Why is Vietnam a motorcycle nation? A transport psychology study*. *European Transport/Trasporti Europei*, 99, 1–17. https://doi.org/10.48295/ET.2024.99.6

[3] https://www.britannica.com/technology/Siri

[4]https://www.amazon.com/gp/help/customer/display.html?nodeId=GJWYQVSUF3W9V7N9

[5] Pandey, R., Waghela, H., Rakshit, S., Rangari, A., Singh, A., Kumar, R., Ghosal, R., & Sen, J. (2024). *Generative AI-Based Text Generation Methods Using Pre-Trained GPT-2 Model*. arXiv. https://arxiv.org/abs/2404.01786

[6] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv. https://arxiv.org/abs/2106.09685

[7]
Comanici, G., Bieber, E., Schaekermann, M., Pasupat, I., Sachdeva, N., **et al.** (2025). *Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities*. arXiv preprint arXiv:2507.06261. https://arxiv.org/abs/2507.06261

[9] Micheletti, N., Belkadi, S., Han, L., & Nenadic, G. (2024). *Exploration of masked and causal language modelling for text generation*. arXiv preprint arXiv:2405.12630. https://arxiv.org/abs/2405.12630

[10] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, “BERTScore: Evaluating Text Generation with BERT,” *arXiv preprint* arXiv:1904.09675, 2020. [Online]. Available: https://arxiv.org/abs/1904.09675

[11] C.-Y. Lin, “ROUGE: A package for automatic evaluation of summaries,” in *Proc. Workshop on Text Summarization Branches Out (ACL 2004)*, Barcelona, Spain, Jul. 2004, pp. 25–26.

[12] J. Meyer, D. I. Adelani, E. Casanova, A. Öktem, D. Whitenack, J. Weber, S. Kabongo, E. Salesky, I. Orife, C. Leong, P. Ogayo, C. Emezue, J. Mukiibi, S. Osei, A. Agbolo, V. Akinode, B. Opoku, S. Olanrewaju, J. Alabi, and S. Muhammad, “BibleTTS: A large, high-fidelity, multilingual, and uniquely African speech corpus,” *arXiv preprint* arXiv:2207.03546, 2022. [Online]. Available: https://arxiv.org/abs/2207.03546

[13] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2019). *BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension*. arXiv. https://arxiv.org/abs/1910.13461

APPENDICES

APPENDIX 1

All the codes and tests for the models are contained in Google Colab.

Dataset generation for NLP Model using OpenWeatherMapAPI and Gemini 2.5 Flash Lite:

https://colab.research.google.com/drive/1Ug3do51mJyn9NiAGKHz5mwujRQZtBmOM#scrollTo=OQV9EPTsGQZH

Fine-tuned GPT2 NLP Model containing preprocessing, LoRA configuration, training configuration:

https://colab.research.google.com/drive/1x_PAbox9IzM7Hlkw-FnNgpG__4p6mI84#scrollTo=XxxjOY7BALFZ

TTS Model using XTTS V2 and my own voice as reference:

https://colab.research.google.com/drive/1hlkKa0Lxg41oo0l9Smj0zmVxwyuCyBd6

In addition, the Hugging Face Space for the **ScriptGen** model is available with interactive documentation:

https://nwtt-gpt2peftweather.hf.space/docs

Dockerfile to set up ScriptGen Hugging Face Spaces:

```
FROM python:3.10-slim
# set working directory
WORKDIR /code
# install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
# copy app
COPY app.py .
# expose the Hugging Face Spaces default port
EXPOSE 7860
# run FastAPI with uvicorn
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "7860"]
```

APPENDIX 2

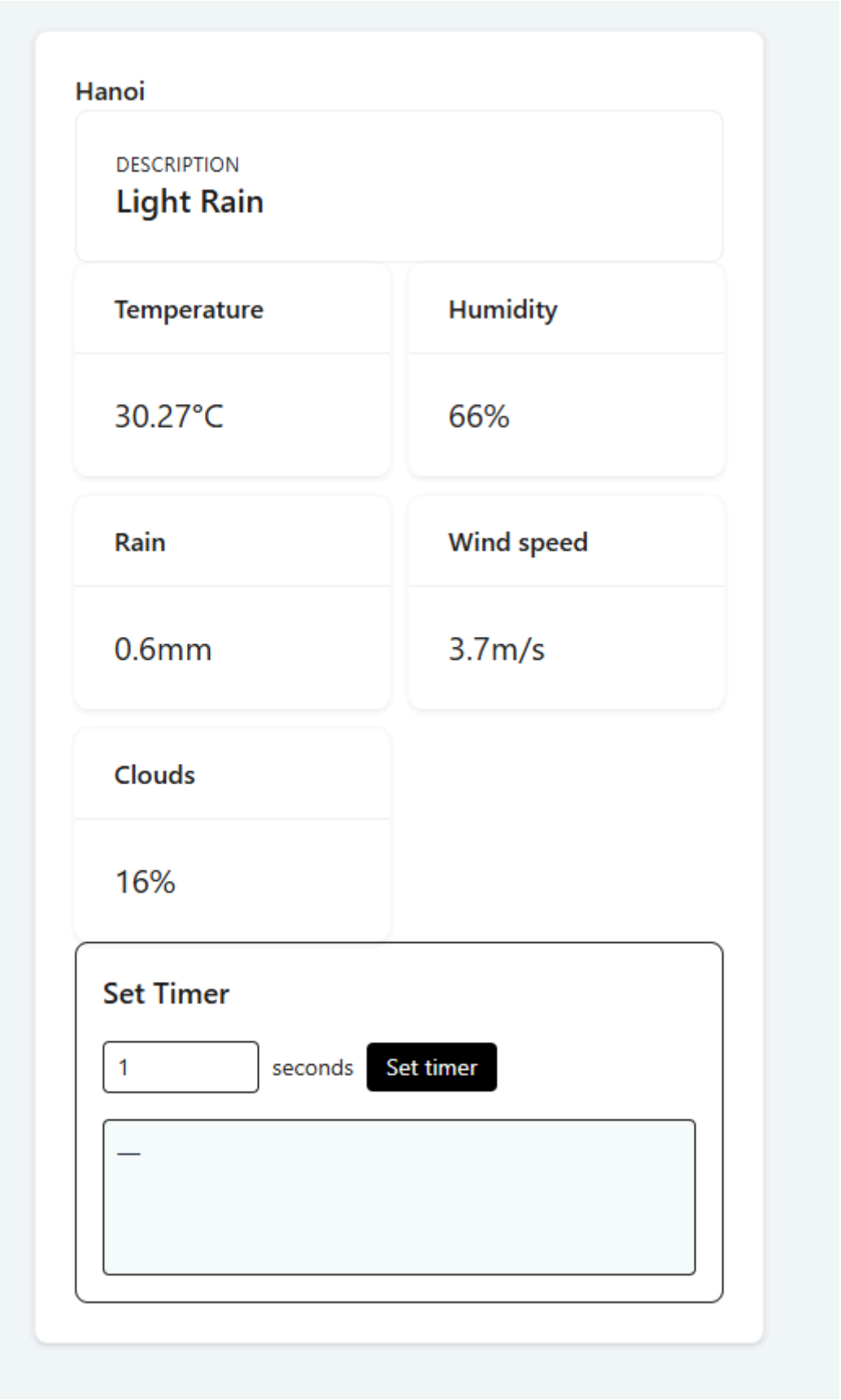

Figure 8: Application UI with basic set time function

and blank script box

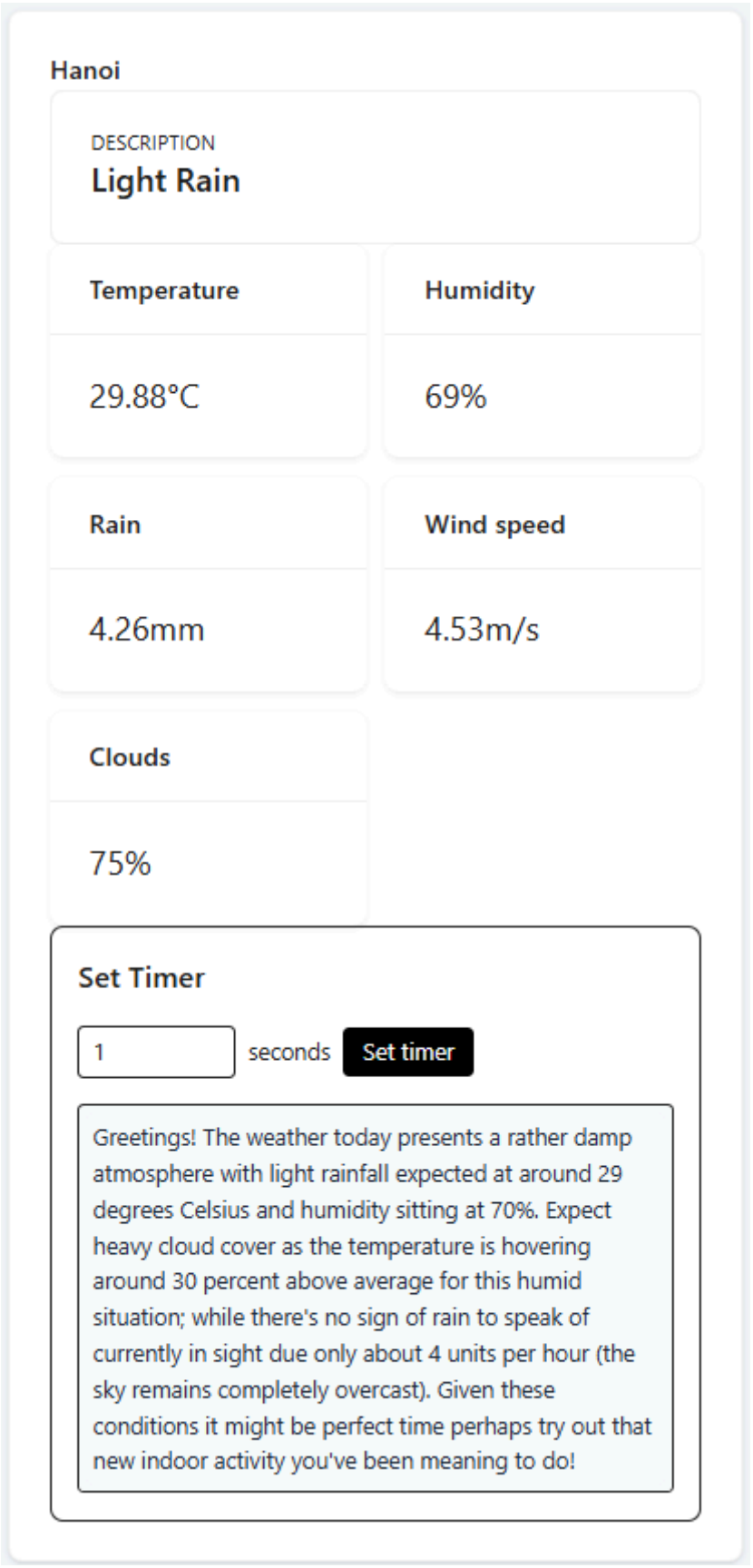

Figure 9: Application UI with generated script box