Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента *Вахрушина Єгора Валентиновича*
(ПІБ)

академічної групи *121м-21-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *«Інженерія програмного забезпечення»*
(назва освітньої програми)

на тему: *Дослідження ефективності синтезу Української мови на базі сімейства нейронних мереж FastSpeech*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний	*Проф. Алексєєв М.О.*			

Рецензент				

Нормоконтролер	*Проф. Лактіонов І.С.*			

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » _____ 20 22 року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

студенту *121м-21-1* *Вахрушину Єгору Валентиновичу*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Дослідження ефективності синтезу Української мови на базі сімейства нейронних мереж FastSpeech*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 24.10.2022р. №1169-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процес синтезу мовлення з використанням нейронної моделі FastSpeech.

Предмет досліджень – методи синтезу мовлення.

Мета НДР – створення програмного забезпечення на основі моделі FastSpeech з реалізацією синтезу української мови.

Вихідні данні для проведення роботи – теоретичні та практичні навички отримані в ході дослідження, аудіо та текстові дані з наборів даних українських аудіо книг.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна даної роботи полягає в модифікації моделі трансляції тексту до мовлення FastSpeech, що дозволяє синтезувати українське мовлення.

Практична цінність результатів полягає в створенні програмного забезпечення моделі трансляції на основі FastSpeech для української мови, та оцінка синтезу результівної моделі, що допомагає в подальшому дослідженні предметної галузі.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати досліджень мають бути подані у вигляді, що дозволяє побачити та оцінити якість удосконаленої моделі трансляції до українського мовлення.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець)**
Аналіз теми та постановка задачі	10.09.2022 – 30.09.2022
Аналіз методів трансляції тексту до мовлення та обробки сигналів	01.10.2022 – 31.10.2022
Навчання моделі трансляції тексту до українського мовлення та проведення аналізу результатів	01.11.2022 – 30.11.2022

Завдання видав _____ *Алексєєв М.О.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Вахрушин Є.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *10.09.2022 р.*
Термін подання кваліфікаційної роботи до ЕК: _____.

РЕФЕРАТ

Пояснювальна записка: 73 сторінки, 22 рисунків, 5 таблиць, 2 додатки, 22 джерел.

Об’єкт дослідження: процес синтезу мовлення з використанням нейронної моделі FastSpeech.

Мета магістерської роботи: створення програмного забезпечення на основі моделі FastSpeech з реалізацією синтезу української мови.

Методи дослідження: для розв'язання поставлених задач використані: нейронні мережі, хмарні технології, функціональне програмування, моделі трансляції тексту, методи обробки сигналів та операційна система Unix з використанням Wsl.

Наукова новизна даної роботи полягає в модифікації моделі трансляції тексту до мовлення FastSpeech, що дозволяє синтезувати українське мовлення.

Практична цінність результатів полягає в створенні програмного забезпечення моделі трансляції на основі FastSpeech для української мови, та оцінка синтезу результівної моделі, що допомагає в подальшому дослідженні предметної галузі.

Область застосування: результат кваліфікаційної роботи може використатися в сферах, що потребують синтез україньскої мови, а також для подальшого розвитку у дослідженнях створення моделей на базі сімейства нейронних мереж FastSpeech.

Значення роботи та висновки: досліджено різні моделі сімейства FastSpeech та створено програмне забезпечення на основі моделі FastSpeech, що може використовуватись для подальшого розвитку в дослідженні моделей синтезу мовлення.

Прогнози щодо розвитку досліджень: дослідити можливість поліпшення моделі FastSpeech за рахунок подальшої модифікації архітектури та покращення мовного набору даних.

Список ключових слів: FFT, FastSpeech, нейронна мережа, модель синтезу мовлення, TensorFlow.

ABSTRACT

Explanatory note: 73 pages, 2 figures, 5 tables, 2 applications, 21 sources.

The object of research: the process of speech synthesis using the FastSpeech neural model.

The purpose of the master's thesis: creation of software based on the FastSpeech model with the implementation of the Ukrainian language synthesis.

Research methods: neural networks, functional programming, text translation models and the Unix operating system using Wsl were used to solve the problems.

Originality of research lies in the modification of the FastSpeech text-to-speech translation model, which allows for the synthesis of Ukrainian speech.

The practical value of the results lies in the creation of FastSpeech-based broadcast model software for the Ukrainian language, and the evaluation of the synthesis of the resulting model, which helps in further research of the subject area.

Scope of application: the result of the qualification work can be used in areas requiring the synthesis of the Ukrainian language, as well as for further development in research on the creation of models based on the FastSpeech family of neural networks.

The value of this work and conclusions: various models of the FastSpeech family were investigated and software based on the FastSpeech model was created, which can be used for further development in the research of speech synthesis models.

The value of this work and conclusions: investigate the possibility of improving the FastSpeech model due to further modification of the architecture and improvement of the language data set.

List of keywords: FFT, FastSpeech, neural network, speech synthesis model, TensorFlow.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

FFT – Fast Fourier Transform

TTS – Text to speech

GAN – Generative Adversarial Network

AI – Artificial Intelligence

HMM - Hidden Markov Model

RNN – Recurrent neural network

ЗМІСТ

ВСТУП

Синтез мовлення є актуальною темою для досліджень, особливо в останні роки. Синтез людського мовлення використовується в багатьох сферах, починаючи від побутового призначення, такого як, голосовий помічник в телефоні, чи озвучування тексту, для людей с вадами зору до професійного призначення, таких як допомога в кол-центрах, чи в сервісах.

В наш час найбільш популярним є створення нейронних мереж різних типів для кодування текстової інформації у звукове представлення.

Зараз, в більшості випадків, для цього використовують різні типи трансляції з використанням глибокого навчання і вони майже витіснили раніше популярні статично параметричний та конкатенативний методи трансляції.

Глибоке навчання – це складне направлення, яке вимагає багато часу та великий обсяг мовних даних, для створення якісного синтезу. Не заважаючи на це, зараз результат є найбільш наближеним до натурального мовлення. Окрім цього, цей підхід є комплексним і зазвичай потребує більш ніж одну нейрону мережу для отримання гарного результату синтезу.

Об’єкт досліджень: процес синтезу мовлення з використанням нейронної моделі FastSpeech.

Предмет досліджень: методи синтезу мовлення.

Мета дослідження: дослідження синтезу української мови.

Методи дослідження: для розв'язання поставлених задач використані: нейронні мережі, хмарні технології, функціональне програмування, моделі трансляції тексту, методи обробки сигналів та операційна система Unix з використанням Wsl.

Наукова новизна даної роботи постає в розробці моделі трансляції тексту до мовлення, що підтримує українську мову і базується на моделі FastSpeech.

Практичне значення: Результат створення моделі трансляції на основі моделі FastSpeech дає можливість використання даної моделі для синтезу українського мовлення, можливості її покрашення в майбутньому та створення нових моделей використовуючи накопичені знання.

Особистий внесок автора полягає в розробці моделі на основі FastSpeech з модифікацією моделі та можливістю навчання моделі на наборах даних української мови з попереднім дослідженням теоретичної частини та систематизації отриманих знань.

Структура та обсяг дипломної роботи: Кваліфікаційна робота складається з трьох розділів та висновків. Містить 73 сторінки тексту, 22 рисунків, 21 використаних джерел та 2 додатки.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Стан синтезу мовлення

Голос — це найприродніший спосіб спілкування. Тому не дивно, що розмовні помічники розвиваються саме в напрямку голосу. Ці віртуальні голосові помічники можуть бути використаними в багатьох випадках. Наприклад:

- для допомоги кол-центрам шляхом попередньої кваліфікації запиту абонента, беручи до уваги найпростіші запити (наприклад, призначення зустрічі);
- для людей із вадами зору вони можуть озвучувати текст на екрані або описати сцену перед ними;
- для персональної роботи з особистими пристроями такими як смартфони та комп'ютери;
- оператор може втрутитися в машину, щоб відремонтувати її, отримавши голосову допомогу, і він може взаємодіяти з нею без рук, тобто без клавіатури чи миші;
- протягом кількох років GPS-навігація використовує голос для надання вказівок;

Розмовні голосові помічники досягли такого рівня розвитку, який ставить їх майже на той самий рівень, що й людину, щодо розпізнавання мовлення та створення «простого» (монотонного) мовлення. Генерація мови – це складний процес, який полягає в генерації кількох тисяч значень, що представляють звуковий сигнал із простого речення.

Нейронні мережі замінили традиційні технології конкатенативної генерації, забезпечивши кращу якість сигналу, спрощену підготовку навчальних даних і зменшивши час генерації, який був скорочений до такого рівня, що дозволяє генерувати речення в кілька сотень разів швидше, ніж це зробила б людина.

Генерація сигналу, як правило, виконується у 2 основні етапи: на першому етапі генерується частотне представлення речення (мел-спектрограма), а на другому етапі генерується форма сигналу з цього представлення. На першому кроці текст перетворюється на символи або фонеми. Вони векторизуються, потім архітектура типу кодер-декодер перетворює ці вхідні елементи в стисло приховане представлення (кодер) і перетворює ці дані в частотне представлення через декодер. Технології, які найчастіше використовуються на цьому етапі — це згорткові мережі, пов’язані з механізмом уваги, щоб покращити узгодження між входом і виходом. Це узгодження часто підкріплюється механізмами передбачення тривалості, рівня та тону. На другому етапі, обробленому так званим вокодером, тривимірне представлення частоти в часі (час, частота та потужність) перетворюється на звуковий сигнал. Серед найефективніших архітектур є архітектури GAN (Generative Adversarial Network), де генератор генеруватиме сигнали, які будуть обробленими дискримінатором.

Коли ці архітектури оцінює людина, рівень якості майже досягає якості навчальних даних. Оскільки важко зробити краще, ніж вхідні дані, дослідження тепер спрямовані на внесок елементів, які зроблять згенерований сигнал ще ближчим до реальності з додаванням елементів просодії, ритму та індивідуальності, а також на можливість параметризації покоління точніше.

Через це текст у мовлення (TTS) привернула багато уваги останніми роками. Системи на основі глибоких нейронних мереж стають все більш популярними для TTS, наприклад: Tacotron [1], Tacotron 2 [2], Deep Voice 3

[3], ClariNet [4] тощо. Ці моделі зазвичай спочатку авторегресійно генерують мел-спектрограму з введеного тексту, а потім синтезують мову з мел-спектрограми за допомогою вокодера, такого як Griffin-Lim [5], WaveNet [6], Parallel WaveNet [7] або WaveGlow [8]. TTS на основі нейронної мережі перевершив звичайний конкатенативний та статистичний параметричний підходи [9] щодо якості мовлення.

1.2. Базування методу глибинного навчання

Глибинне навчання — це клас алгоритмів машинного навчання, який використовує кілька шарів для поступового вилучення високорівневих ознак з необроблених вхідних даних. Наприклад, під час обробки зображень нижні рівні можуть ідентифікувати краї, тоді як вищі рівні можуть ідентифікувати поняття, які надають сенсу для людини, наприклад цифри, літери чи обличчя.

Далі буде розглянуто основні поняття які потрібні для праці із DL та TTS загалом.

1.2.1. Сигнал

Сигнал – це зміна певної величини з часом. Для аудіо, величиною, яка змінюється, є тиск повітря. Щоб зафіксувати цю інформацію в цифровому вигляді, потрібно взяти зразки атмосферного тиску з часом. Частота вибірки даних може змінюватися, але найчастіше це 44,1 кГц або 44 100 вибірок на секунду. Результатом є хвиля сигналу (рис 1.1), яку можливо інтерпретувати, змінювати й аналізувати за допомогою комп’ютерного програмного забезпечення.

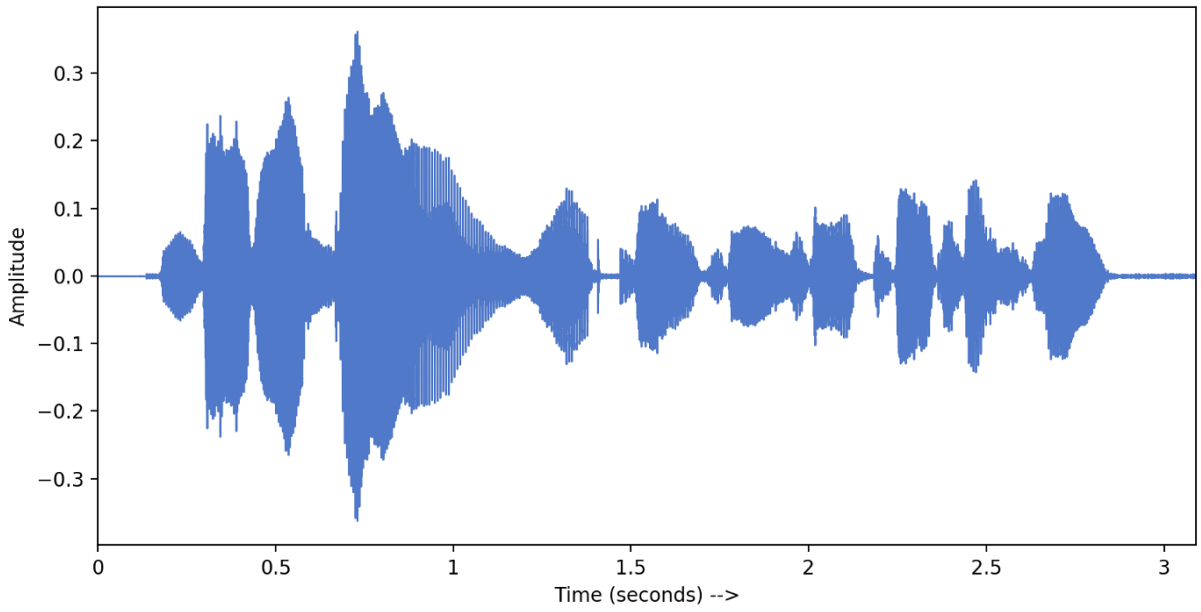

Рис. 1.1 Приклад відображення аудіо сигналу

1.2.2. Трансформація Фур'є та Швидке перетворення Фур’є

Щоб отримати з аудіо сигналу інформацію з якою можливо працювати використовується трансформація Фур'є.

Звуковий сигнал складається з кількох одночастотних звукових хвиль. Збираючи зразки сигналу в часі, ми фіксуємо лише отримані амплітуди. Перетворення Фур’є — це математична формула, яка дозволяє нам розкласти сигнал на його окремі частоти та амплітуду частоти. Іншими словами, він перетворює сигнал із часової області в частотну. Результат називається спектром (рис 1.2).

Це можливо, оскільки кожен сигнал можливо розкласти на набір синусоїдальних і косинусоїдальних хвиль, які в сумі утворюють вихідний сигнал. Ця теорема, відома як теорема Фур’є.

Швидке перетворення Фур’є (FFT) — це алгоритм, який обчислює дискретне перетворення Фур’є. Дискретне перетворення Фур’є (DFT) може бути записаним формулою 1.1.

$$x[k] = \sum_{n}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}} \quad (1.1)$$

Для обчислення DFT дискретного сигналу (де N – це область сигналу), ми множимо кожне його значення на e, зведене до деякої функції від n. Потім ми підсумовуємо результати, отримані для заданого n. Тобто для виконання алгоритму потрібно виконати N() на N(). Це означає, що складність цього алгоритму $O(N^2)$. Як випливає з назви, швидке перетворення Фур'є (FFT) — це алгоритм, який визначає дискретне перетворення Фур'є вхідних даних значно швидше, ніж безпосередньо обчислюючи їх. FFT зменшує складність алгоритму з $O(N^2)$. до $O(Nlog(N))$.

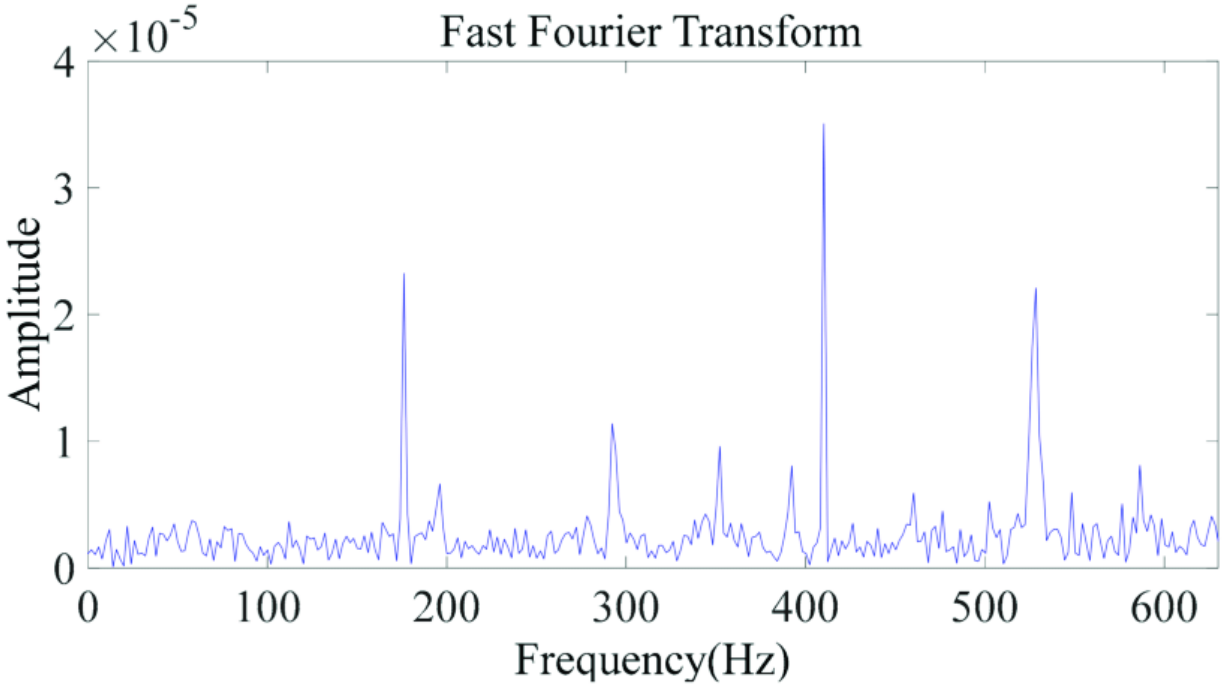

Рис. 1.2 Спектр створений завдяки перетворенню Фур'є

1.2.3. Спектрограма

Швидке перетворення Фур’є є потужним інструментом, який дозволяє нам аналізувати частотний вміст сигналу, але частотний вміст сигналу може змінюється з часом. Це стосується більшості звукових сигналів, таких як музика та мова. Ці сигнали відомі як неперіодичні сигнали. Оскільки вони змінюються з часом, потрібен спосіб представити спектр цих сигналів. Для цього використовується короткочасне перетворення Фур'є. FFT обчислюється на сегментах сигналу, що перекриваються, і ми отримуємо те, що називається спектрограмою (рис 1.3).

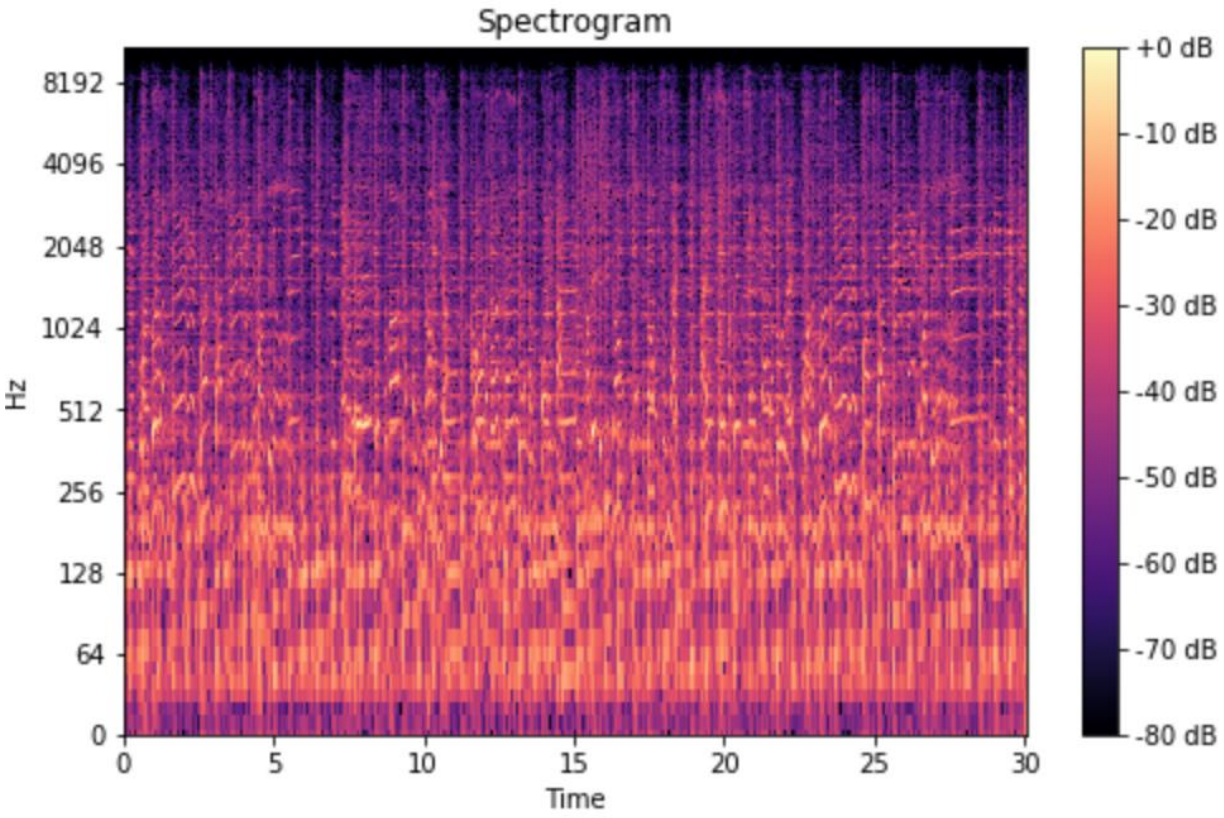

Рис. 1.3 Приклад спектрограми

Це виглядає як група FFT, розташованих одна на одній. Це спосіб візуального представлення гучності або амплітуди сигналу, оскільки вона змінюється з часом на різних частотах. Вісь *y* перетворюється на

логарифмічний масштаб, а розмірність кольору перетворюється на децибели (це можна вважати логарифмічним масштабом амплітуди). Це через те, що люди можуть сприймати лише дуже малий і концентрований діапазон частот і амплітуд.

1.2.4. Шкала Мел

Дослідження показали, що люди не сприймають частоти в лінійному масштабі. Ми краще вміємо виявляти відмінності на нижчих частотах, ніж на вищих. Наприклад, ми можемо легко визначити різницю між 500 і 1000 Гц, але навряд чи зможемо визначити різницю між 10 000 і 10 500 Гц, навіть якщо різниця між двома значеннями однакова.

У 1937 році Стівенс, Фолькман і Ньюман запропонували таку одиницю висоти, щоб рівні висоти звучали однаково далеко для слухача. Це називається шкалою Мела. Ми виконуємо математичну операцію над частотами, щоб перетворити їх у шкалу Мел (рис. 1.4).

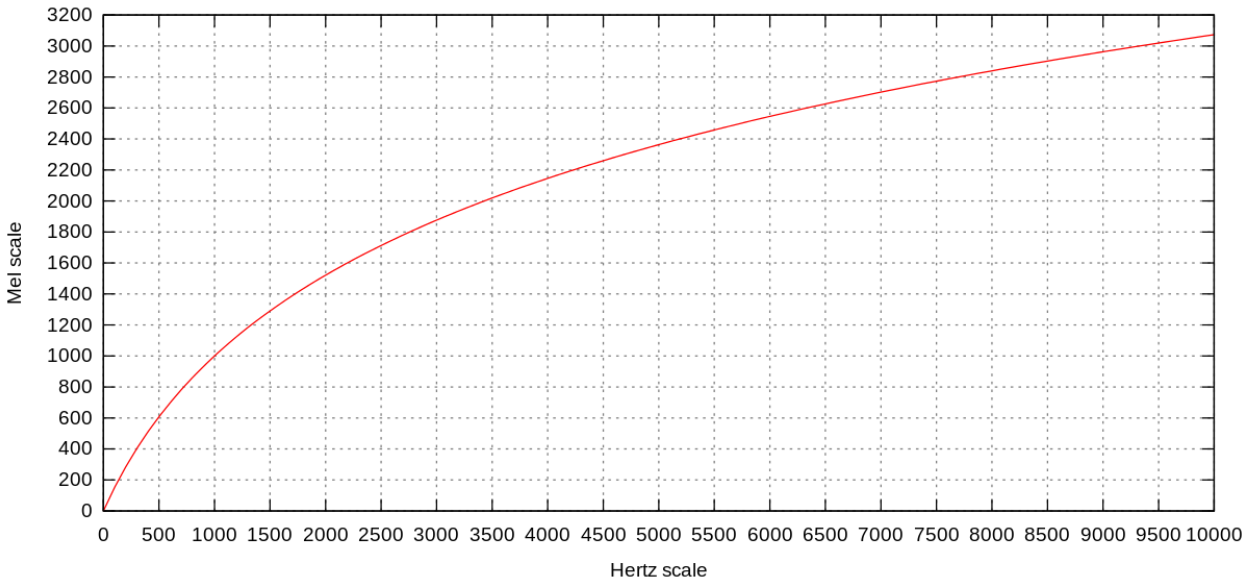

Рис. 1.4 Шкала Мел

1.2.5. Мел-спектрограма

Мел-спектрограма — це спектрограма, у якій частоти перетворені в мел-шкалу (рис 1.5).

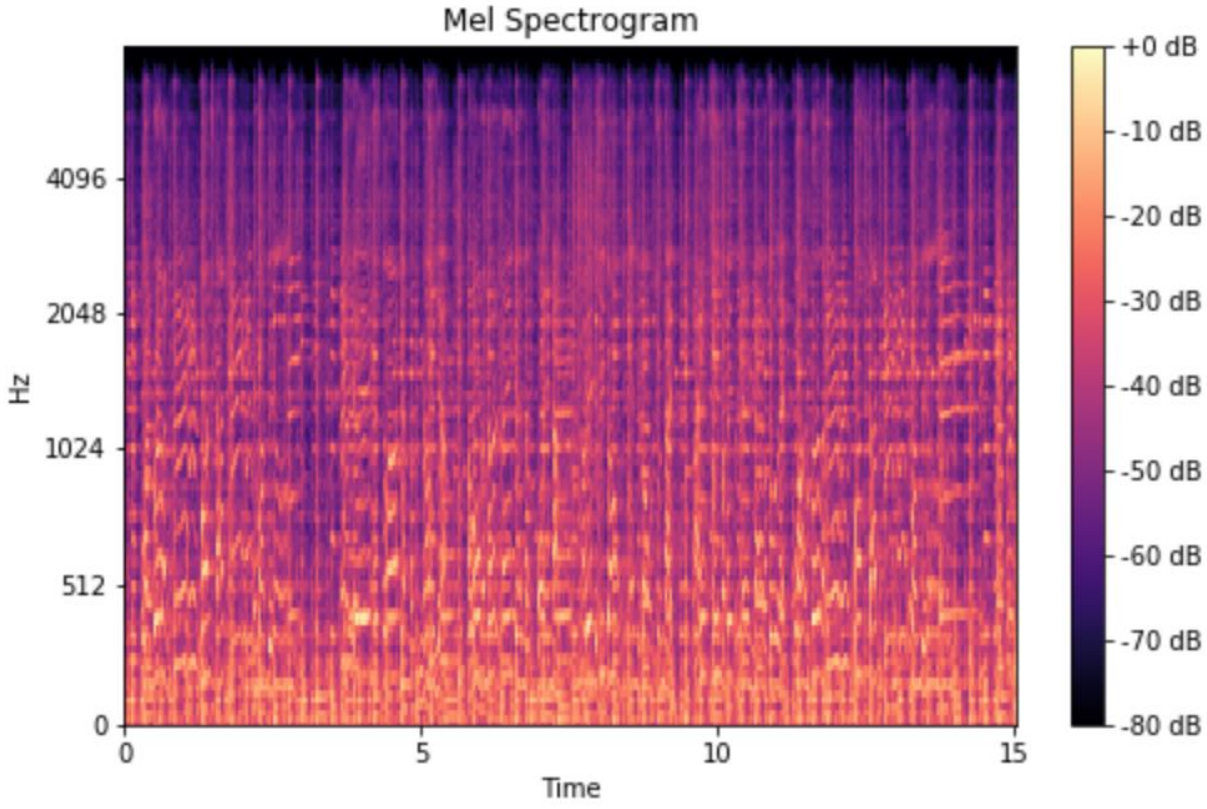

Рис. 1.5 Мел-спектрограма

1.3. Процес праці генерації голосу

Перші системи генерації голосу використовували безпосередньо повітря для створення звуків, потім інформатика принесла системи, які могли використовувати правила генерації за параметрами, щоб швидко адаптувати генерацію речень шляхом конкатенації дифонів із більш-менш послідовної бази даних звуків (є більше ніж 1700 дифонів англійською та 1200

французькою мовами, які дублюються мовцем, початком/кінцем речення, просодією…)

Традиційні системи синтезу мовлення часто класифікують на дві категорії: конкатенативні системи та генеративні параметричні системи. Збірка дифонів належить до категорії конкатенативного синтезу мовлення.

Існують дві різні схеми для конкатенативного синтезу: одна базується на коефіцієнтах лінійного прогнозування (LPC), інша — на синхронному додаванні перекриттів (PSOLA). Результат часто плоский, монотонний і роботизований, тобто не має справжньої просодії. Під просодією маються на увазі над сегментні особливості, такі як інтонація, мелодія, паузи, ритм, потік, акцент… Цей метод було вдосконалено завдяки створенню генеративних акустичних моделей на основі «Прихована модель Маркова» (HMM) і реалізації контекстних дерев рішень. Глибинні генеративні системи тепер стали стандартом, скинувши старі системи.

Згідно з принципом «один до багатьох», система повинна перетворити текст в проміжний стан, а потім перетворити цей проміжний стан в аудіо сигнал. Більшість систем статистичного параметричного синтезу мови (SPSS) не генерують сигнал безпосередньо, а радше його частотне представлення. Потім другий компонент, який називається вокодером, завершує генерацію на основі цього представлення. Принципи генеративних мереж стали нормою в останні роки для згорткових мереж, потім для рекурентних, VAE (2013), механізмів уваги (2014), GAN (2014) та інших мереж.

На рисунку 1.6 нижче описано різні компоненти архітектури процесу генерації мовлення на основі машинного навчання.

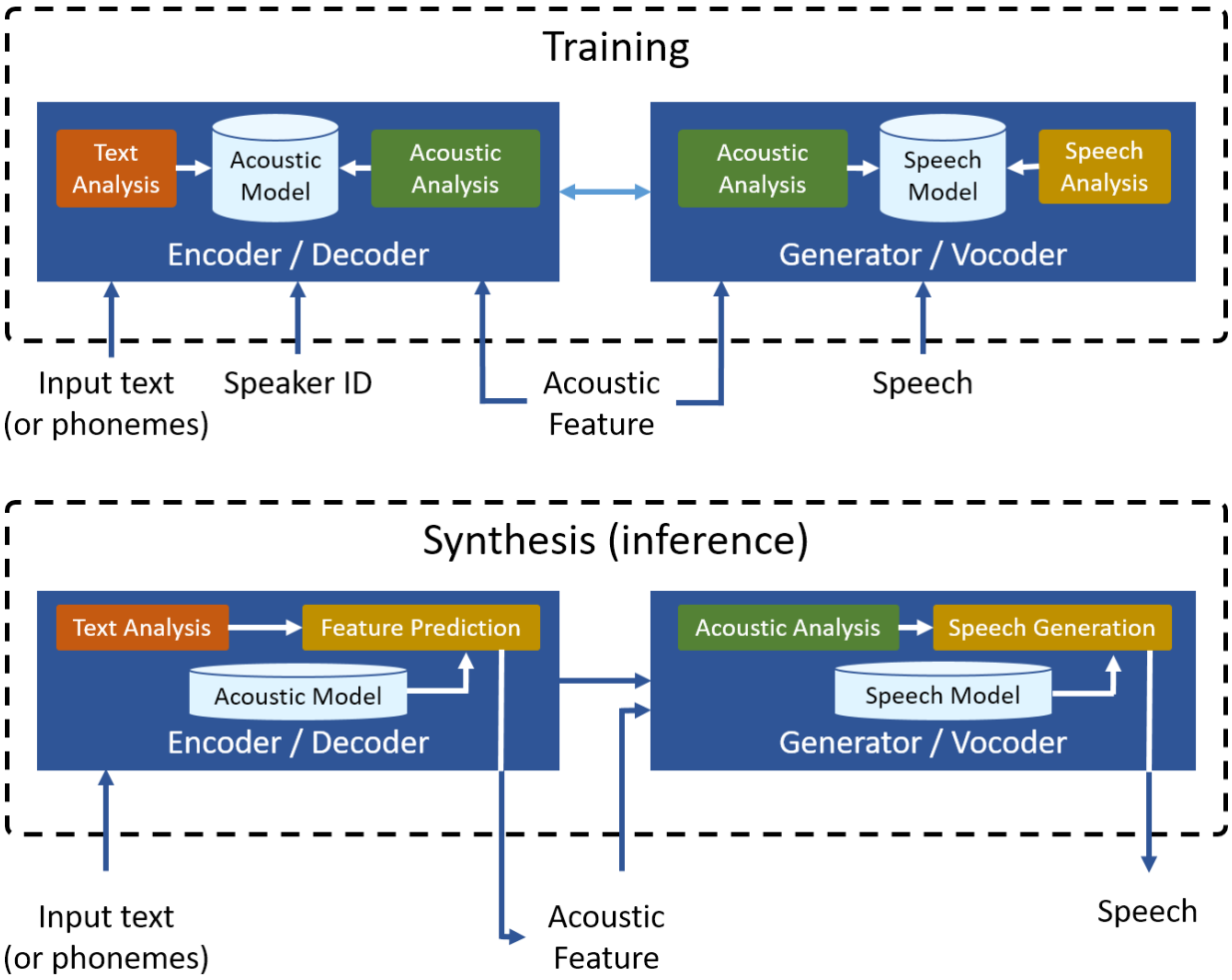

Рис. 1.6 Стандартна діаграма системи генерації мовлення

Як і будь-яка система, заснована на навчанні, генерація в основному складається з 2 фаз: фази навчання та фази генерації (або фази висновку). Іноді вставляється «проміжна» фаза, щоб виконати тонке налаштування акустичної моделі з іншими даними.

Зовнішній вигляд процесу змінюється в залежності від фази:

- Під час фази навчання конвеєр дозволяє генерувати моделі. Речення є вхідними даними кодера/декодера та голосовими файлами, пов'язаними з реченнями. Іноді до цього додається ідентифікатор спікера. У багатьох системах генерується мел-спектрограма, а вокодер перетворює це представлення на хвилю.

Вхідними даними для вокодера є акустичні параметри (зазвичай мел-спектрограма) і голос, пов’язаний із параметрами. Набір інформації, отриманої з двох модулів аналізу синтезу, називається «лінгвістичною особливістю» (Acoustic Feature).

- Під час фази генерації конвеєр відповідає за створення висновку (або синтезу чи генерації). Вхідними даними є речення, яке потрібно трансформувати, іноді ідентифікатор мовця для вибору функцій мовлення, які відповідатимуть згенерованому голосу. Результатом є мел-спектрограма. Роль вокодера полягає в тому, щоб створити остаточну форму хвилі з компактного представлення аудіо, з якого буде згенеровано мовлення.

Процес, який використовується для навчання, включає:

- Модуль аналізу тексту, який виконує операції нормалізації тексту, перетворює числа на текст, ділить речення на частини (частини мови), перетворює графеми (написані склади) на фонеми (G2P), додає елементи просодії тощо. Деякі системи обробляють символи тексту безпосередньо, інші використовують лише фонеми. Цей модуль часто використовується «як є» під час навчання та синтезу.
- Модуль акустичного аналізу отримує як вхідні дані акустичні характеристики, що асоціюються з текстом. Цей модуль також може отримати ідентифікатор спікера під час навчання з кількома ораторами. Цей модуль аналізуватиме відмінності між теоретичними характеристиками та даними, отриманими під час фази навчання. Акустичні характеристики можна генерувати зі зразків голосу за допомогою «класичних» алгоритмів обробки сигналу, таких як швидке перетворення Фур’є. Цей модуль також може генерувати моделі для прогнозування тривалості сигналу (зв’язок між фонемою та кількістю зразків мел-спектрограми) та його узгодження з текстом. Найновіші системи, як правило,

покращують мережі прогнозування та додають прогнозування висоти. Наприкінці 2020 року Єва Секелі зі школи EECS у Стокгольмі додає обробку дихання на етапах навчання, що зменшує різницю між людиною та машиною.

- Акустичні моделі з фази навчання представляють латентні стани, виділені з вектора вбудовування речення, вектора мовця та акустичних характеристик. Крім того, існують моделі прогнозування для вирівнювання та інших функцій.
- Модуль аналізу мовлення використовується для вилучення різних параметрів з оригінальних голосових файлів (Ground-Truth). У деяких системах, особливо «наскрізних» системах, мовчання спереду та ззаду видаляються. Виділення, яке відрізняється від системи до системи, може складатися з виділення висоти, енергії, наголосу, тривалості фонеми, основної частоти (частота 1-ї гармоніки або F0) тощо з вхідних мовних сигналів. Ці вхідні голосові файли можуть мати одного чи багатьох ораторів. У випадку систем із кількома ораторами вектор ораторів додається до вхідних даних.

Процес, який використовується для синтезу, включає:

- Базуючи на основі результату модуля аналізу тексту, модуль Feature Prediction генерує компактне представлення мовлення, необхідне для завершення генерації. Результівні дані можуть бути одним або декількома з наступних представлень: мел-спектрограма сигналу (MelS), кепстральні коефіцієнти шкали Барка (Cep), спектрограми лінійної шкали логарифмічної величини (MagS), основна частота (F0), тривалість фонем, висота звуку тощо
- Вхідним сигналом вокодера може бути одне або більше з наведених вище представлень. Існує багато версій цього модуля, і він, як правило, використовується як окремий блок користуючись

наскрізними системами Серед найпопулярніших вокодерів Griffin-Lim, WORLD, WaveNet, SampleRNN, GAN-TTS, MelGAN, WaveGlow і HiFi-GAN, які забезпечують сигнал, близький до сигналу людини.

Ранні архітектури на основі нейронних мереж спиралися на використання традиційних параметричних процесів TTS, таких як; DeepVoice 1 і DeepVoice 2. DeepVoice 3, Tacotron, Tacotron 2, Char2wav і ParaNet використовують архітектури seq2seq на основі уваги. Системи синтезу мовлення, засновані на глибоких нейронних мережах (DNN), зараз перевершують так звані класичні системи синтезу мовлення, такі як синтез конкатенативного вибору одиниць і HMM, які (майже) більше не зустрічаються в дослідженнях.

На діаграмі нижче представлено різні архітектури, класифіковані за роками, публікації наукової статті. Він також показує посилання, коли система використовує функції попередньої системи.

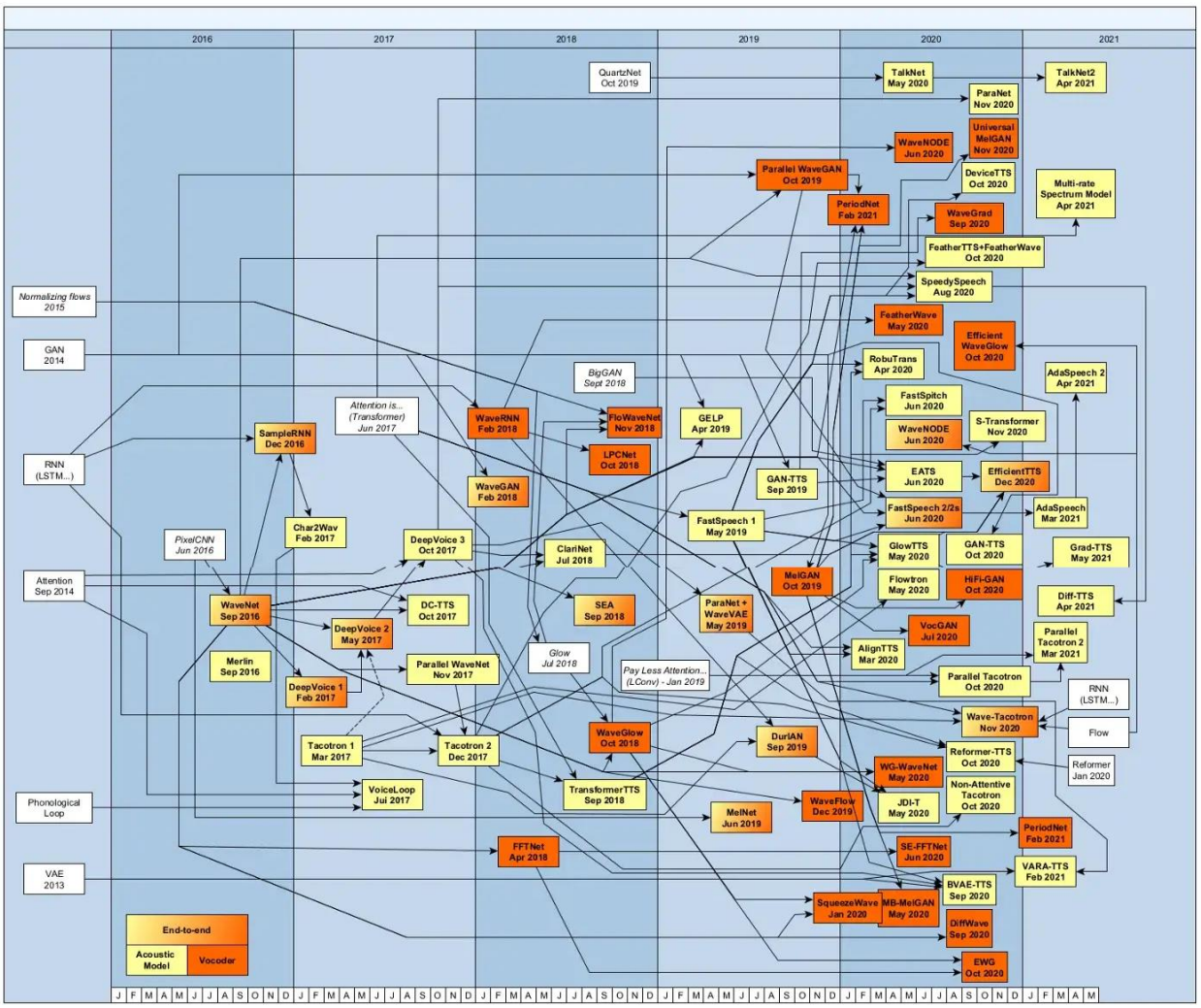

Рис. 1.7. Архітектури класифіковані за роками

1.4. Проблема «один до багатьох»

Щоб згенерувати аудіо сигнал, система синтезу виконуватиме набір більш-менш складних кроків (рис. 1.6). Однією з головних проблем, які має вирішувати синтез голосу, є моделювання «один-до-багатьох», яке полягає в здатності перетворювати вхідну інформацію (речення, яке буде озвучено) у дані (форму сигналу), які приймають кілька тисяч значень. Крім того, цей сигнал може мати багато різних характеристик: гучність, акцентування окремого слова, швидкість дикції, керування кінцем речення, додавання почуттів, висоту... Проблема для архітектора системи полягає в тому, щоб

вирішити цю складну обробку шляхом поділу цього процесу на етапи, які можна навчати глобально чи індивідуально.

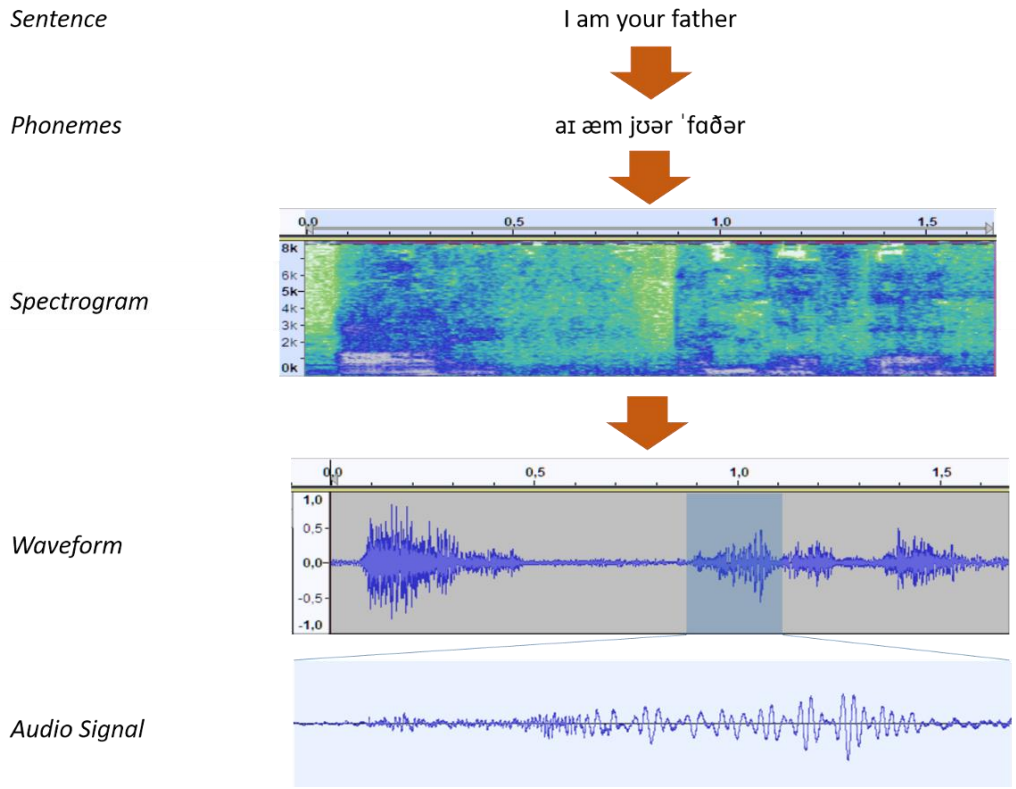

Рис. 1.8 Приклад кроків потрібних для генерації сигналу

1.5. Проблематика авторегресійної послідовності

У сучасних системах TTS на основі нейронної мережі мел-спектрограма генерується авторегресійно. Через довгу послідовність мел-спектрограми та авторегресійну природу, ці системи стикаються з декількома проблемами:

- Повільна швидкість генерації мел-спектрограми. Хоча CNN і TTS на основі трансформатора [10] можуть прискорити навчання моделей на основі RNN [11], усі моделі генерують мел-спектрограму на основі попередньо згенерованих мел-спектрограм

і страждають від низької швидкості знаходження висновку, враховуючи, що в послідовності мел-спектрограм зазвичай сотні або тисячі елементів

- Синтезоване мовлення зазвичай не надійне. Через поширення помилок [12] і неправильне вирівнювання уваги між текстом і мовленням в авторегресійній генерації. Згенерована мел-спектрограма зазвичай неповноцінна з проблемами пропуску та повторення слів [13].
- Синтезоване мовлення некероване. Попередні авторегресійні моделі генерують мел-спектрограми одну за одною автоматично, без явного використання вирівнювання між текстом і мовою. Як наслідок, зазвичай важко безпосередньо контролювати голос швидкість і просодія в авторегресійній генерації.

1.6. Не авторегресійна модель

В останні роки не авторегресійні моделі TTS використовуються для розв'язання проблем які виникають при використанні авторегресійної моделі. Не авторегресійні моделі створюють мел-спектрограми з надзвичайно високою швидкістю та уникнути проблем із надійністю, досягнувши порівнянної якості голосу з попередньою авторегресійною моделлю.

На відміну від генерації авторегресійної послідовності, не авторегресійні моделі генерують послідовність паралельно, без явної залежності від попередніх елементів, це може значно прискорити процес знаходження висновку.

1.7. Сімейство алгоритмів FastSpeech

Серед цих не авторегресійних методів TTS, FastSpeech є однією з найбільш успішних моделей. FastSpeech має два способи полегшення проблеми «один до багатьох»:

- Зменшення дисперсії даних на цільовій стороні шляхом використання згенерованої мел-спектрограми з авторегресійної моделі вчителя як навчальної цілі (тобто дистиляції знань).
- Знайомство з інформацією про тривалість (витягнута з карти уваги моделі викладача), щоб розширити текстову послідовність відповідно до довжини послідовності мел-спектрограми.

Поки ці особливості FastSpeech полегшують роботу із проблемою «один до багатьох» у TTS, вони також приносять кілька недоліків:

- Двоступенева підготовка вчителя-учня робить навчальний процес біль складним.
- Цільові мел-спектрограми, створені з моделі викладача, мають певну втрату інформації порівняно з базовими, оскільки якість звуку, синтезованого з генеровані мел-спектрограми зазвичай гірші ніж цільові.
- Тривалість витягнута з карти уваги моделі вчителя недостатньо точна.

1.8. Швидкість синтезу FastSpeech

На сьогодні FastSpeech є однією із найбільш швидких моделей синтезу мовлення. В науковій статті наведено порівняння швидкості синтезу FastSpeech зі швидкістю синтезу авторегресивної моделі Transformer TTS, яка має аналогічну кількість параметрів з FastSpeech. Результати порівняння наведено в таблиці 1.1. Можна побачити, що FastSpeech прискорює генерацію мел-спектрограми у 269 разів у порівнянні з моделлю Transformer

TTS. Також наведено швидкість синтезу моделей з WaveGlow у якості вокодеру. Можна помітити, що FastSpeech все ще досягає 38-кратного прискорення генерації звуку.

Таблиця 1.1

Порівняння швидкості FastSpeech

Метод	Затримка	Прискорення
Transformer TTS (Mel)	6.735±3.969	$\frac{1}{269.4}$
FastSpeech (Mel)	0.025±0.005	
Transformer TTS (Mel + WaveGlow)	6.895±3.969	$\frac{1}{38.3}$
FastSpeech (Mel + WaveGlow)	0.180±0.078	

1.9. Постановка задачі

Отже, в результаті розгляду різних підходів до синтезу мовлення та проблем з якими вони стикаються, стає зрозумілим, що сімейство алгоритмів FastSpeech, що використовує не авторегресійну модель, є актуальним напрямком у сфері. У даній кваліфікаційній роботі необхідно розв'язувати наступні задачі:

1. Розглянути моделі сімейства FastSpeech;
2. Обрати модель із сімейства FastSpeech для подальшої модифікації;
3. Запропонувати удосконалення цієї моделі задля навчання та синтезу українського мовлення;

4. Створити програмне забезпечення оновленої моделі.
5. Провести навчання нової моделі, для подальшого порівняння у ході експериментів.
6. Провести оцінювання якості синтезованого мовлення.
7. Обробити отримані дані, візуалізувати їх за допомогою таблиць та графіків та зробити висновки.

РОЗДІЛ 2

ВИЗНАЧЕННЯ МОДЕЛІ СИНТЕЗУ МОВЛЕННЯ

2.1. Вибір алгоритму сімейства FastSpeech

На сьогодні існує декілька версій алгоритму FastSpeech: FastSpeech, FastSpeech 2 та FastSpeech 2s. Засновуючись на результаті порівняння в науковій статті [20], можливо констатувати наступне:

- FastSpeech 2 має кращу якість результівного звуку ніж FastSpeech
- FastSpeech 2 має простіший процес обробки та має в 3 рази більшу швидкість ніж FastSpeech
- FastSpeech 2s має якість звуку та швидкість навчання наближенні до FastSpeech 2

Далі детально описано архітектуру та особливості кожної з перерахованих моделей.

2.1.1. FastSpeech

У цьому розділі представлений дизайн-архітектури FastSpeech. Щоб генерувати цільову послідовність мел-спектрограми паралельно, використовується структура прямого зв’язку. Замість використання архітектури на основі «кодер-увага-декодер», що використовується більшістю авторегресійних [14] і не авторегресійних [15] генераціях послідовності. Компоненти детально описуються в наступних підрозділах.

2.1.1.1. Трансформатор прямого поширення

FastSpeech використовує нову структуру – трансформатор прямого поширення, відкидаючи звичайну структуру кодер-увага-декодер, як показано на рисунку 2.1. Основним компонентом трансформатора прямого поширення [16] є «блок трансформатор прямого поширення» (блок FFT, як показано на рисунку 2.2), який складається з механізму власної уваги та одновимірної згортки [17]. Блоки FFT використовуються для перетворення послідовності фонем у послідовність мел-спектрограми, з кількістю блоків N зі сторін фонеми та мел-спектрограми відповідно. Між ними є регулятор довжини, який використовується для вирішення невідповідності довжини між фонемою та послідовністю мел-спектрограми.

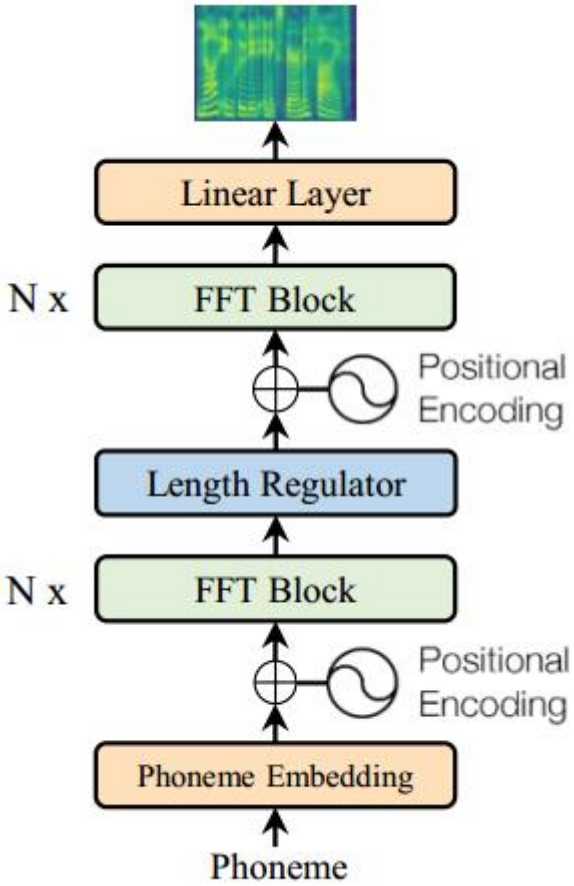

Рис. 2.1 Трансформатор прямого поширення

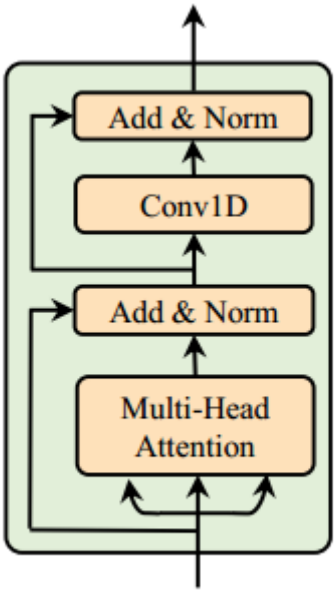

Рис. 2.2 Блок FFT

2.1.1.2. Регулятор довжини

Регулятор довжини моделі показаний на рисунку 2.3. Оскільки довжина послідовності фонем менша, ніж довжина послідовності мел-спектрограми, одна фонема відповідає кільком мел-спектрограмам. Кількість мел-спектрограм, яка відповідає фонемі, називається тривалістю фонеми. Регулятор довжини розширює приховану послідовність фонем відповідно до тривалості, щоб відповідати довжині послідовності мел-спектрограми. Також регулятор довжини може пропорційно збільшувати або зменшувати тривалість фонеми, щоб регулювати швидкість голосу, а також змінювати тривалість порожніх токенів для регулювання розривів між словами, щоб контролювати просодії.

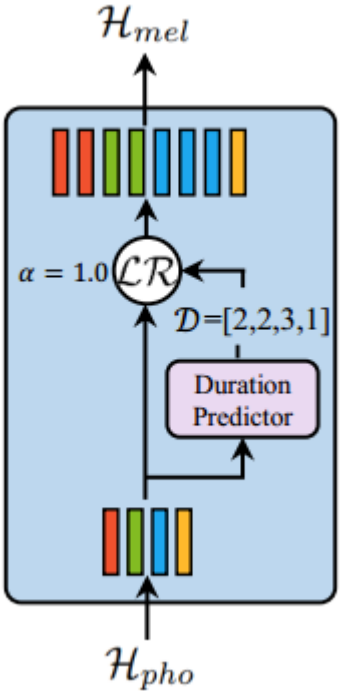

Рис 2.3 Регулятор довжини

2.1.1.3. Предиктор тривалості

Предиктор тривалості критично потрібен регулятору довжини для визначення тривалості кожної фонеми. Як показано на рисунку 2.4, предиктор тривалості складається з двошарової одновимірної згортки та лінійного шару для прогнозування тривалості. Прогноз тривалості накопичується в блоці FFT на стороні фонеми та навчається одночасно з FastSpeech за допомогою функції втрати середньоквадратичної помилки (MSE). Значення тривалості фонеми витягується з вирівнювання уваги між кодером і декодером в авторегресійній моделі викладача.

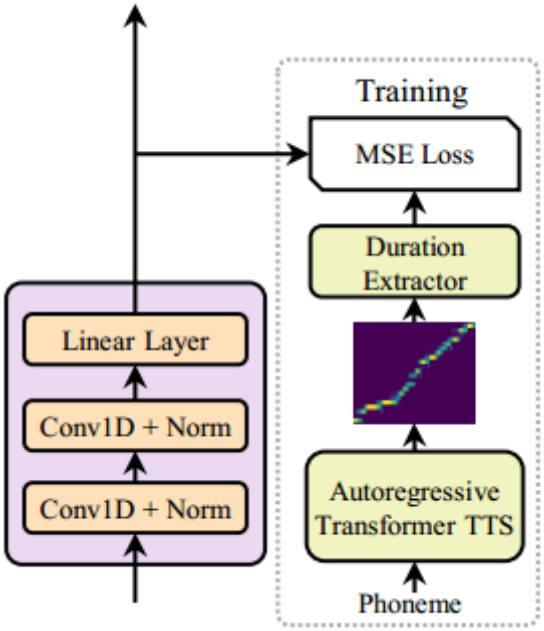

Рис 2.4 Предиктор тривалості

2.1.2. FastSpeech 2

Загальна архітектура моделі FastSpeech 2 показана на рисунку 2.5. Кодер перетворює послідовність вбудовування фонем у приховану послідовність фонем, а потім адаптер дисперсії додає різну інформацію про дисперсію, таку як тривалість, висота тону та енергія, до прихованої послідовності. Нарешті декодер мел-спектрограми перетворює адаптовану приховану послідовність у мел-спектрограму послідовності паралельно.

В структурі кодера й декодера мел-спектрограми використовується блок FFT і одномірна система згортки, як у FastSpeech. На відміну від FastSpeech, який покладається на дистиляційний процес «учитель-учень» і на тривалість фонеми що береться з моделі вчителя, у FastSpeech 2 є кілька покращень.

В моделі відсутній дистиляційний процес «викладач-учень». Також мел-спектрограми використовуються як ціль для навчання моделі, що допомагає уникнути втрати інформації в дистильованих мел-спектрограмах і підвищити

верхньої межі якості голосу. Адаптер дисперсії складається не лише з предиктора тривалості, але також з предиктора висоти та предиктора енергії. Предиктор тривалості використовує тривалість фонеми, отриману примусовим вирівнюванням [17], як ціль навчання, яка є точнішою, ніж отримана з карти уваги авторегресійної моделі вчителя.

Додаткові предиктори висоти та енергії можуть надати більше інформації про дисперсію, що є важливим для полегшення проблеми відношення «один до багатьох» у TTS. Нові компоненти детально описуються в наступних підрозділах.

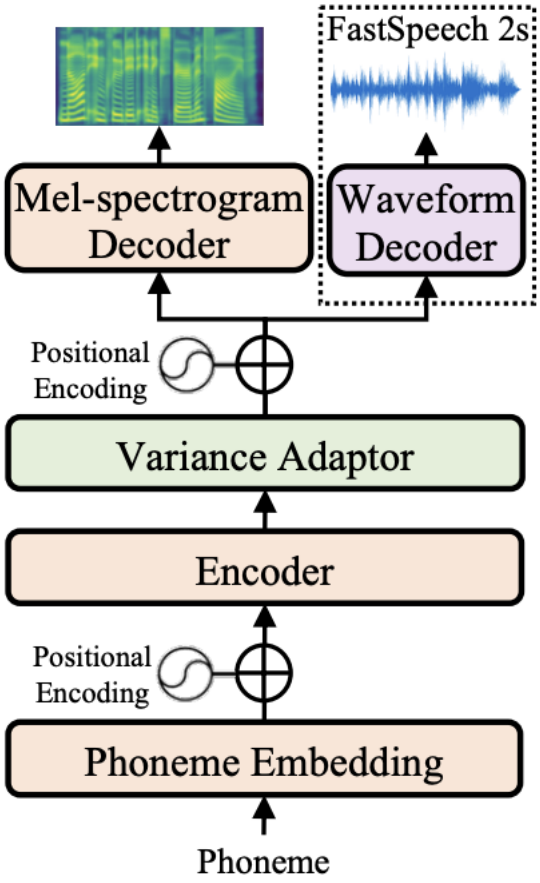

Рис 2.5 Архітектура FastSpeech 2

2.1.2.1. Адаптер дисперсії

Адаптер дисперсії додає інформацію про дисперсію (наприклад, тривалість, висоту тону, енергію тощо) до прихованої послідовності фонем. Як показано на рисунку 2.6, адаптер дисперсії складається з наступних компонентів:

- Предиктор тривалості (тривалість фонеми, яка показує, як довго звучить голос);
- Предиктор висоти (висота, яка є ключовою особливістю, що допомагає відображати емоції та впливає на просодію);
- Предиктор енергії (енергія безпосередньо впливає на гучність і просодію мови).

Під час навчання береться базове значення тривалості, висоти та енергії, отриманих із записів і використовуються як вхідні дані для прихованої послідовності. Це робиться для виявлення цільової мови. Одночасно з цим використовуються окремі предиктори дисперсії для прогнозів тривалості, висоти та енергії, які використовувалися під час знаходження висновку для синтезу цільової мови. Як показано на рисунку 2.7 прогнози тривалості, висоти та енергії мають схожу структуру моделі (але різні параметри моделі), яка складається з 2-шарової одновимірної згорткової мережі з активацією ReLU.

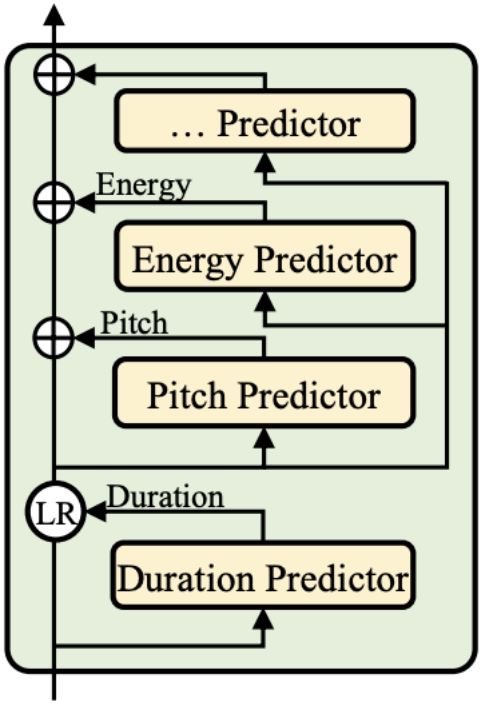

Рис. 2.6 Адаптер дисперсії

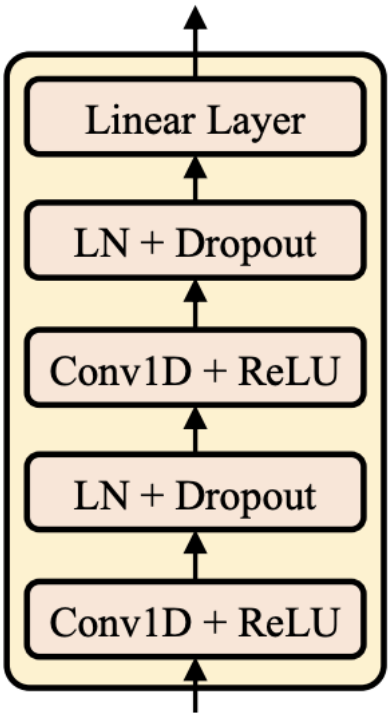

Рис 2.7 Предиктор тривалості/висоти/енергії

2.1.2.2. Предиктор тривалості

Предиктор тривалості приймає приховану послідовність фонеми як вхідні дані та передбачає тривалість кожної фонеми. Тривалість фонеми означає скільки мел-фреймів відповідає цій фонемі. Прогноз тривалості оптимізується за допомогою операції «Середньоквадратична похибка» (MSE), беручи виділену тривалість як ціль навчання. Для знаходження тривалості фонеми використовуються інструмент «Montreal forced alignment» (MFA). Він дозволяє підвищити точність вирівнювання та завдяки цьому зменшити інформаційний проміжок між входом і виходом моделі.

2.1.2.3. Предиктор висоти

Попередні системи TTS на основі нейронної мережі з прогнозуванням висоти [18] часто прямо прогнозують контур висоти. Однак через високі варіації основного тону, розподіл прогнозованих значень тону сильно відрізняється від очікуваного розподілу. Щоб краще передбачити варіації контуру висоти, використовується безперервне вейвлет-перетворення (CWT), щоб розкласти безперервний ряд висоти тону на спектрограму висоти тону [20] і взяти спектрограму висоти тону як ціль навчання для предиктора висоти тону, який оптимізується операцією MSE. У логічному висновку предиктор тону прогнозує спектрограму основного тону, яка далі перетворюється назад у контур основного тону за допомогою інверсного безперервного вейвлет-перетворення (iCWT).

2.1.3. FastSpeech 2s

FastSpeech 2s базується на FastSpeech 2. Головною ціллю FastSpeech 2s є повне забезпечення наскрізного навчання. Як показано на рисунку 2.8, FastSpeech 2s додає декодер сигналу, який приймає приховану послідовність адаптера дисперсії як вхідні дані та безпосередньо генерує сигнал. В архітектурі залишається декодер мел-спектрограм, щоб допомагати з виділенням ознак тексту під час навчання. У висновку видалено декодер мел-спектрограми, а використовується лише декодер форми сигналу для синтезу звуку.

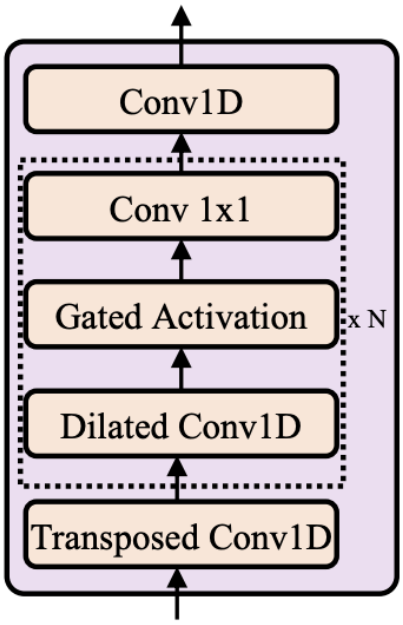

Рис 2.8 Декодер сигналу

2.2. Оцінка якості MOS

Для визначення якості та порівняння моделей FastSpeech буде використаний MOS.

Середня оцінка думки (MOS) — це числова міра оцінки людиною загальної якості події чи досвіду. В нашому випадку MOS – це рейтинг якості голосового сигналу.

MOS виражається як раціональне число, як правило, в діапазоні 1–5, де 1 означає найнижчу сприйману якість, а 5 – найвищу сприйману якість. Також можливі інші діапазони MOS, залежно від рейтингової шкали, яка використовувалася в основному тесті. Шкала оцінки абсолютної категорії дуже часто використовується, яка відображає оцінки від «Погано» до «Відмінно» на числа від 1 до 5, як показано в таблиці 2.1.

Таблиця 2.1

Оцінка та значення MOS

Оцінка	Значення
1	Відмінно
2	Добре
3	Середньо
4	Не достатньо
5	Погано

MOS рахується як середнє арифметичне значення усіх оцінок (Формула 2.1)

$$MOS = \frac{\sum_{n=1}^{N} R_n}{N}$$

Формула 2.1 значення MOS

2.3. Порівняння MOS між різними версіями алгоритму FastSpeech

В таблиці приведено порівняння між різними моделями FastSpeech за допомогою MOS. Данні для порівняння були взяті із наукової статті[20]. Як видно на таблиці 2.2, усі моделі мають відносно гарний результат, але найкращим є алгоритм FastSpeech 2

Таблиця 2.2

MOS різних моделей FastSpeech

Метод	MOS
FastSpeech	3.68 ± 0.09
FastSpeech 2	3.83 ± 0.08
FastSpeech 2s	3.71± 0.09

2.4. Висновки

Після дослідження та порівняння кожної з моделей сімейства FastSpeech, було вирішено розробляти модифіковану модель на основі базового алгоритму FastSpeech.

Не зважаючи на те, що FastSpeech 2 показує кращий результат в якості синтезу мовлення та швидкості навчання, він все ще базується на архітектурі моделі FastSpeech.

Через це, для отримання більш ґрунтових знань в напрямку, було обрано першу версію моделі. Це допоможе при подальшому дослідженні напрямку синтезу мовлення

РОЗДІЛ 3

МОДИФІКАЦІЯ ТА ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ З МОДЕЛЛЮ FASTSPEECH

3.1. Вибір мови програмування та середовища розробки

На цей час, найпопулярнішою мовою програмування нейронних мереж є Python. Є багато бібліотек та фреймворків для AI, таких як Numpy, Pandas, Matplotlib, SciPy, Scikit-learn, TensorFlow, Keras, PyTorch, Coqui-TTS тощо. Окрім цього є багато матеріалів, що полегшить праці із цією мовою програмування. Через це Python було обрано для розробки. Як середовище розробки було обрано Visual Code від Microsoft. Visual Code має багато встановлюваних модулів для полегшення праці з Python і його бібліотеками.

3.2. Алгоритм проведення практичних експериментів

Алгоритм проведення практичних експериментів складається з наступних етапів:

1. Вибір проєкт з відкритим кодом з імплементацією потрібної моделі синтезу, для подальшої модифікації.
2. Вибір машини для навчання: локальний, хмарний тощо.
3. Вибір конфігурації для комп’ютера, на якому будуть проводитись дослідження.
4. Модифікація моделей для їх навчання та запуску.
5. Пошук та вибір мовного набору даних для тренування моделей та проведення досліджень.

6. Обробка нового набору даних для використання в моделі.
7. Розподіл мовного корпусу для різних цілей: тренування, тестування, оцінювання.
8. Запуск навчання моделей.
9. Ручна перевірка тренованих моделей.
10. Вибір способу та шкали оцінювання результатів.
11. Проведення оцінювання.
12. Формування висновків проведеного дослідження.

3.3. Вибір проєкт з відкритим кодом

Серед проєкт з відкритим кодом, що мають реалізацію необхідної моделі є наступні варіанти:

- Coqui TTS;
- TensorFlowTTS.

Coqui TTS – це бібліотека для синтезу мовлення, що була створена як спроба зробити синтез мовлення якомога легшим і доступним та при цьому мати гарну якість синтезу.

TensorFlowTTS – це бібліотека для синтезу мовлення з відкритим кодом, що на відмінну від Coqui TTS має складніші API, і більш важча у використанні і запуску, але має більше можливостей для модифікації моделі.

В обох бібліотеках немає української мови для синтезу.

Для подальшої роботи було обрано TensorFlowTTS, через можливість легшої модифікації коду.

3.4. Структура проєкт

За основу було взято код з проєкт TensorFlowTTS. Цей проєкт знаходиться у відкритому доступі. В його основі використовується бібліотека TensorFlow.

TensorFlow – це одна з найбільш популярних бібліотек для штучного інтелекту. Вона має велику кількість інструментів для розв'язання різних задач із цієї області. Нижче приведені задачі, для рішення яких вона використається:

- числові обчислення на основі багатовимірних масивів;
- розподілені обчислювання завдяки GPU;
- диференціювання;
- побудова моделей, їх навчання та використання.

З проєкт TensorFlowTTS було взято код лише для моделі FastSpeech, для подальшої модифікації. Частково код було оновлено, для підтримки біль нової версії TensorFlow. Як результат, проєкт, що використовується в роботі, має наступні файли:

- uk_dataset.py – файл в якому проходить обробка мовного набору даних, задаються фонеми української мови. В цьому файлу також реалізовано видалення зайвих символів з тексту, таких як точка, кома, тире, тощо;
- base_config.py – файл в якому задається клас базової конфігурації, який буде використовуватися при створення конфігурації FastSpeech;
- Base_dataset.py – файл в якому реалізовано базовий клас для роботи з мовними наборами даних;

- Base_model.py – файл в якому реалізовано базовий клас моделі;
- Base_processor.py – файл, в якому знаходиться допоміжний клас, що використовується при обробці мовного набору даних в uk_dataset.py;
- Creaners.py – допоміжний файл, який містить методи, що роблять уніфікацію вхідного тексту, перетворюючи його в формат ascii, перетворюють великі літери на малі, перетворюють числа в текст та заміняють абревіатури на повні назви;
- Config_fast.py – файл в якому створюється клас з конфігураціями FastSpeech та задаються усі параметри моделі;
- Dataset.py – файл в якому створюється клас обробки мовного набору даних
- Main.py – файл в якому знаходиться стартова точка проекту, в ньому задано параметри які можливо використовувати при запуску проекту;
- Model_fast.py – файл в якому знаходиться клас моделі FastSpeech. Більш докладно цей клас розглянуто в розділі 3.3.2
- Number_norm.py – файл з допоміжними методами, які використовуються для перетворення чисел та валютних символів до тексту;
- Strategy.py – файл в якому реалізовані функції втрати;
- Trainer.py – файл в якому знаходяться класи для навчання моделі;
- Utils.py – файл з допоміжними методами для праці з шляхами системи і методами для зберігання вагових коефіцієнтів;
- Optimizer.py – файл в якому реалізовано механізм власної уваги.

3.5. Основні класи проекту

В цьому розділі більш детально розглянуто усі основні класи проекту.

3.5.1. Клас FastSpeechConfig

В цьому класі задаються параметри для кодера та декодера що використовуються в реалізації моделі FastSpeech. Базові значення перечисленні у таблиці 3.1.

Таблиця 3.1

Параметри класу FastSpeechConfig

Параметр	Значення
Кількість лексичних одиниць	Сумарна кількість літер алфавіту, знаків пунктуації та спеціальних символів
Кількість спікерів	1
Розмір прихованого слою кодера	384
Кількість прихованих слоїв кодера	4
Розмір власної уваги кодера	192
Розмір проміжного слою кодера	1024
Розмір ядра проміжного слою кодера	3
Розмір прихованого слою декодера	384
Кількість прихованих слоїв декодера	4

Кількість механізмів власної уваги декодера	2
Розмір механізму власної уваги декодера	192
Розмір проміжного слою декодера	1024
Розмір ядра проміжного слою декодера	3
Ймовірність вилучення вузла з скритого слою	0.1
Ймовірність вилучення проблеми механізму власної уваги	0.1
Діапазон для створення усіченого нормального розподілу	0.02
Відхилення	1e-5
Максимальне вбудовування	2048

3.5.2. Клас TFFastSpeechSelfAttention

В цьому класі реалізовано модуль власної уваги для FastSpeech. Клас створюється із параметрів взятих з класу FastSpeechConfig. В ньому створюються три слої: «query», «key» та «value», та задається механізм вилучення «dropout». В класі є два методи:

- transpose_for_scores – метод, що робить транспонування для обчислення показника уваги
- call – основний метод класу, в якому знаходиться уся логіка роботи механізму власної уваги

3.5.3. Клас TFFastSpeechIntermediate

Це клас – модуль проміжного слою моделі. Він складається із двох одновимірних згорткових слоїв. Логіка праці модулю розташована в методі call.

3.5.4. Клас TFFastSpeechOutput

TFFastSpeechOutput – це модуль виходу нейронної мережі. Він складається з одного слою, що нормалізує значення активації попереднього слою. Тобто застосовує перетворення, яке перетворює середнє значення активації в кожному прикладі близько до 0, а стандартне відхилення активації – близько до 1.

В модулі TFFastSpeechOutput також присутній механізм вилучення «dropout».

3.5.5. Клас TFFastSpeechLayer

Цей клас – абстракція слою FastSpeech, що складається з описаних раніше модулів:

- TFFastSpeechSelfAttention
- TFFastSpeechIntermediate
- TFFastSpeechOutput

В методі call налаштована взаємодія між усіма модулями, що є в класі. Метод приймає вхідні данні, обробляє їх завдяки слоїв нейронної мережі в модулях та передає далі.

3.5.6. Клас TFFastSpeech

TFFastSpeech – це клас моделі FastSpeech, він містить в собі усі інші класи, що містять в собі слої. В цьому класі з різних слоїв та компонентів, створюється робоча модель.

3.5.7. Клас FastSpeechTrainer

FastSpeechTrainer – це клас, що виконує навчання моделі FastSpeech. Цей клас приймає на вхід мовний набір даних, та обробляє його в моделі. Допоміжні методи класу:

- compute_per_example_losses – вираховує втрату кожного шагу, та повертає масив із втратами
- generate_and_save_intermediate_result – цей метод потрібен для збереження проміжного результату

3.6. Вибір вокодера

Вокодер – інструмент, що використовується для синтезу мовлення на основі довільного сигналу з багатим спектром. В нашому випадку, він використовується для того, щоб перетворити результуючу мел-спектрограму на звуковий сигнал.

В даній роботі, було обрано використати вокодер Parallel WaveGAN.

Parallel WaveGAN це швидкий метод, що генерує хвилі невеликого розміру і використає генеративну змагальну мережу. У запропонованому методі неавторегресійна WaveNet навчається шляхом паралельної обробки спектрограм та функції втрати, що допомагає ефективно фіксувати частотно-часовий розподіл реалістичної форми сигналу. Оскільки метод не потребує дистиляції щільності, яка використовується у звичайній структурі «вчитель-учень», усю модель можливо легко навчити навіть із невеликою кількістю параметрів. Приклад структури вокодера можливо побачити на рисунку 3.1

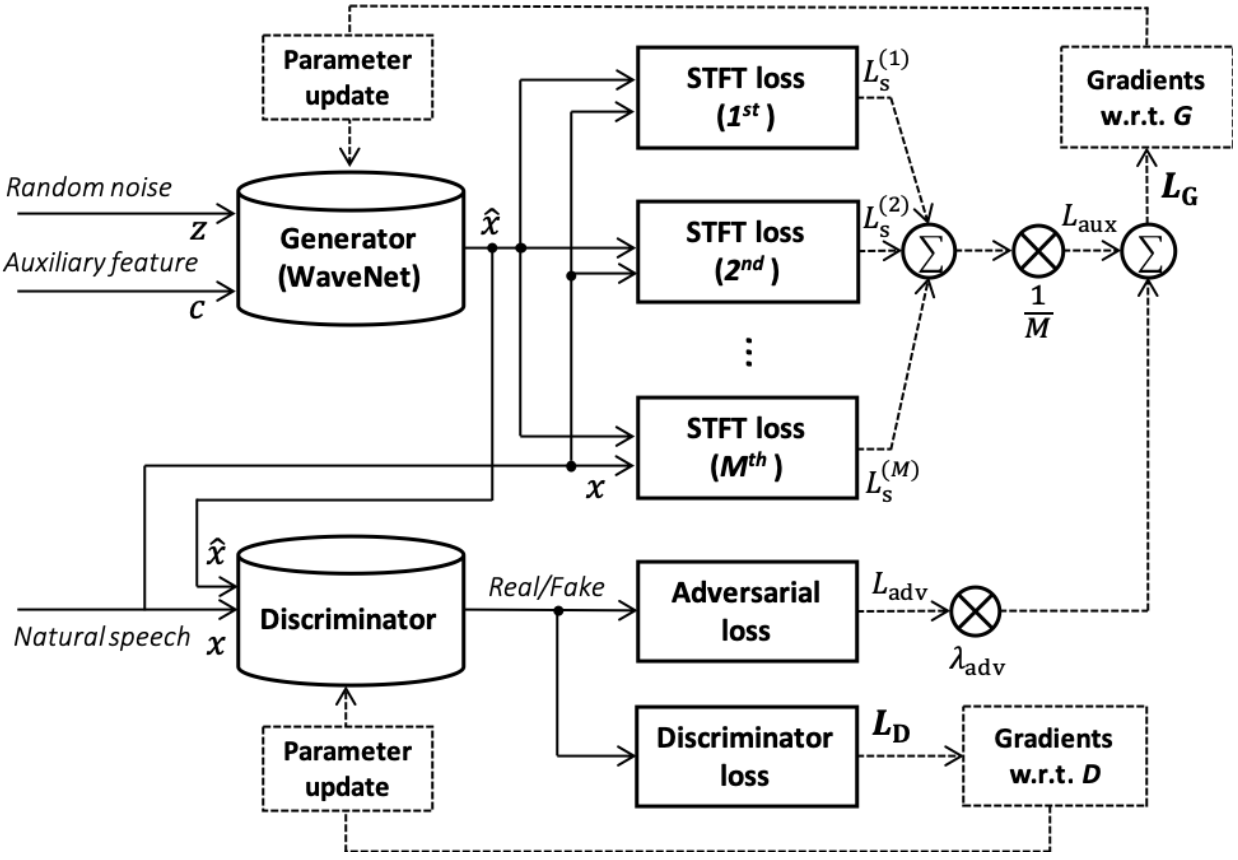

Рис. 3.1 Приклад структури вокодера Parallel WaveGAN

3.7. Підготовка мовного набору даних

Для навчання було обрано набір мовних даних «Ukrainian Open Speech To Text Dataset». Цей набір мовних даних знаходиться в відкритому доступі на ресурсі github. В цьому сеті є багато озвучених книг різними спікерами.

Через те, що система буде навчатися на даних одного спікера, перед початком роботи, мовний набір даних було відфільтровано за спікерами. Оскільки у різних спікерів звукові файли були різної якості, було переслухано по кілька аудіо файлів кожного з них, для вибору найбільш якісного звуку.

Як результат, було обрано одного з спікерів, але через те, що в набору мовних даних для ідентифікації спікеру використовується цифровий індекс, знайти ім’я спікера не вдалося.

3.8. **Результати навчання моделі трансляції тексту до мовлення**

В цьому розділі представленні різні графіки та спектрограми для демонстрації результатів роботи моделі синтезу мовлення.

На рисунку 3.2 зображено спектрограму Ground truth, тобто очікуваний результат на основі вхідного мовного набору даних. Як можливо побачити на спектрограмі, вхідний набір даних не ідеальної якості. При подальшому розгляді теми, необхідно буде знайти новий мовний набір даних. Судячи із спектрограми Ground truth, зміна мовного набору даних може поліпшити якість синтезу.

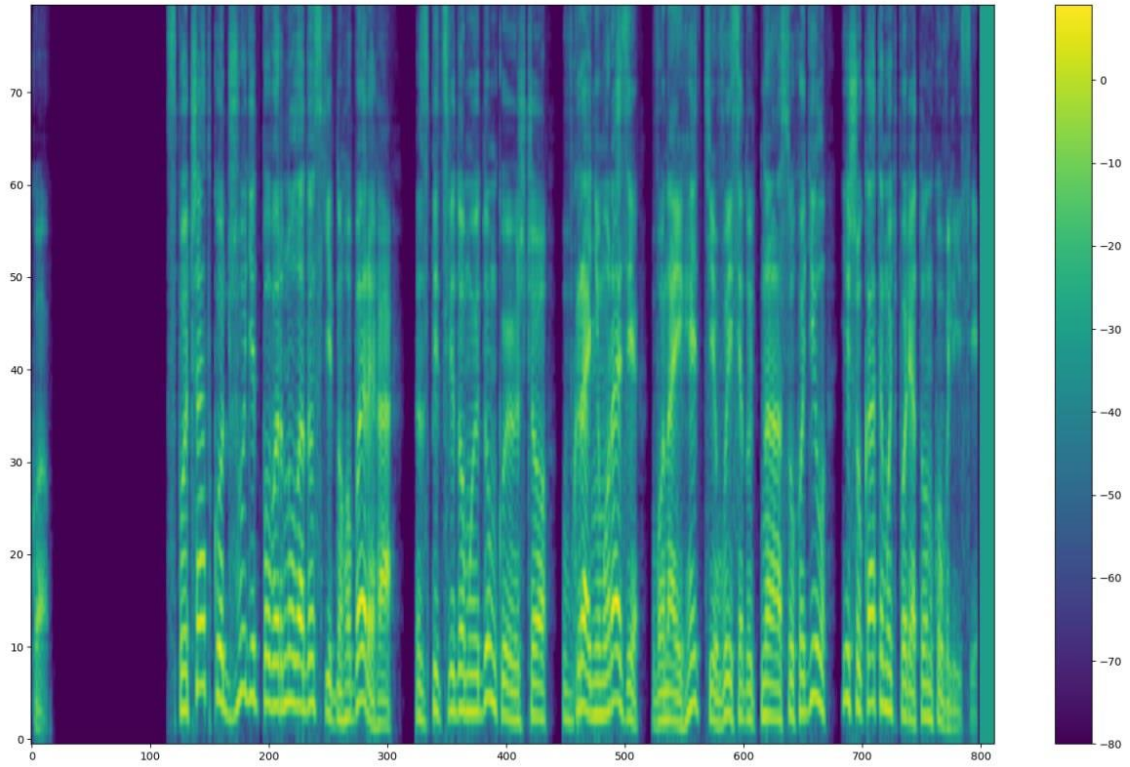

Рис 3.2. Спектрограма Ground truth

В наступних двох графіках, зображено спектрограму передбачення тексту в самому початку навчання (рис. 3.3) та спектрограму передбачення тексту в кінці навчання (рис. 3.4).

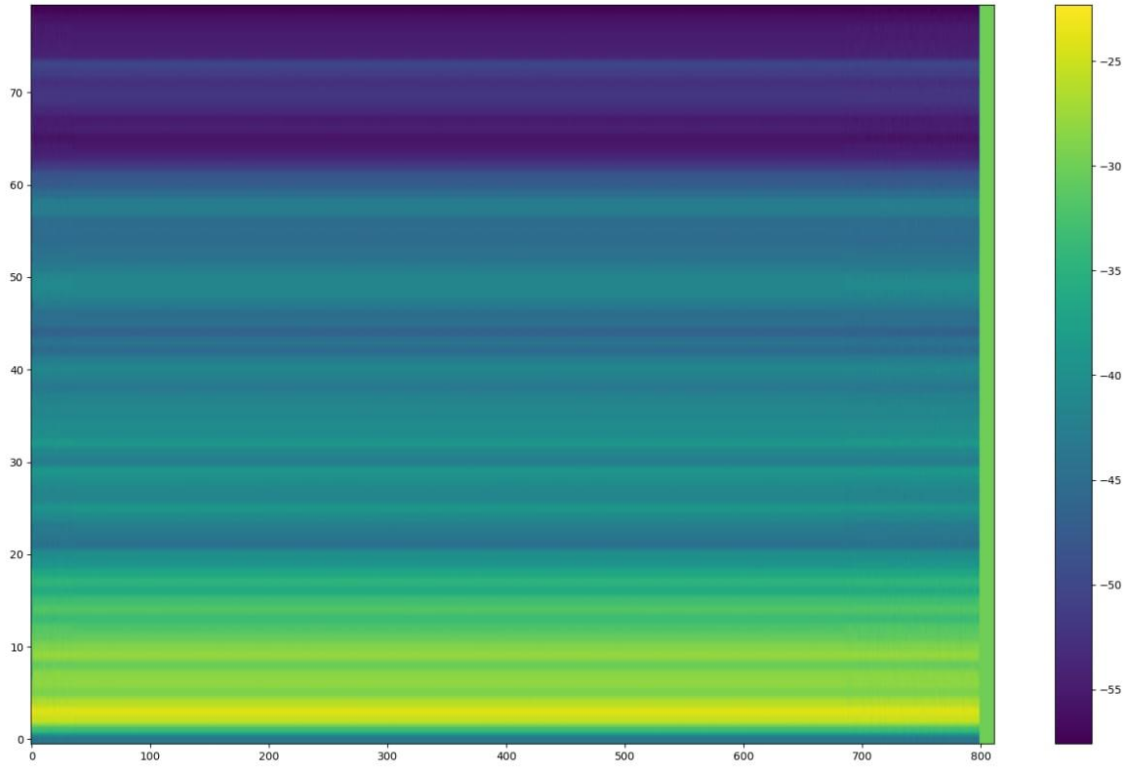

Рис. 3.3. Спектрограма передбаченого тексту на начальному етапі навчання

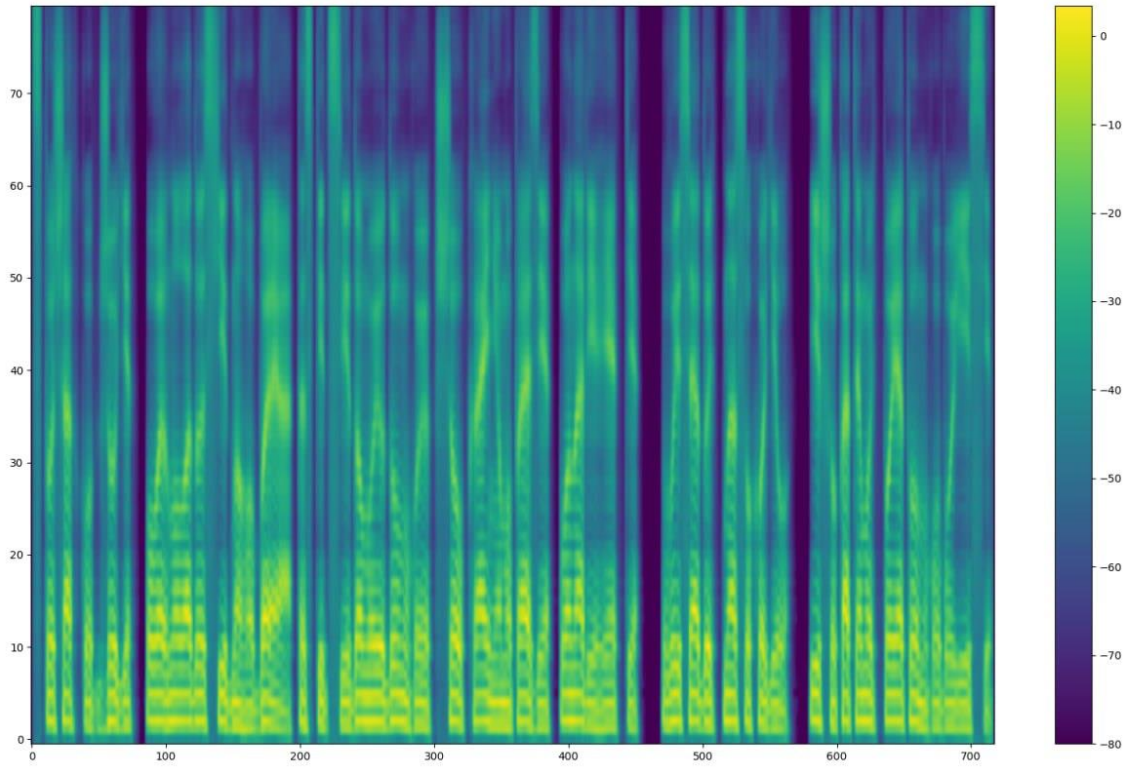

Рис. 3.4. Спектрограма передбаченого тексту в кінці навчання

На наступних графіках відображено механізм уваги на етапі навчання (рис. 3.5) та на етапі перевірки (рис. 3.6). Ці графіки були зроблені на початку навчання. Можливо чітко бачити, що проблема в мовних даних також впливає на механізм уваги.

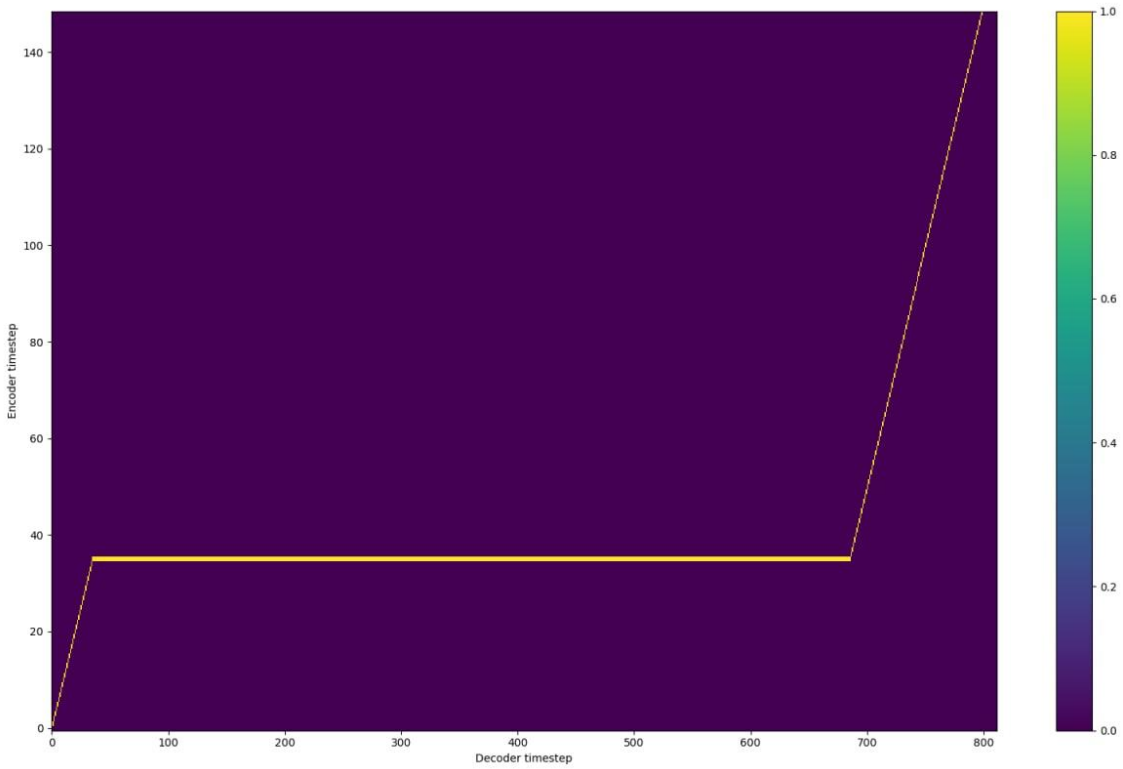

Рис. 3.5. Механізм уваги на початку навчання

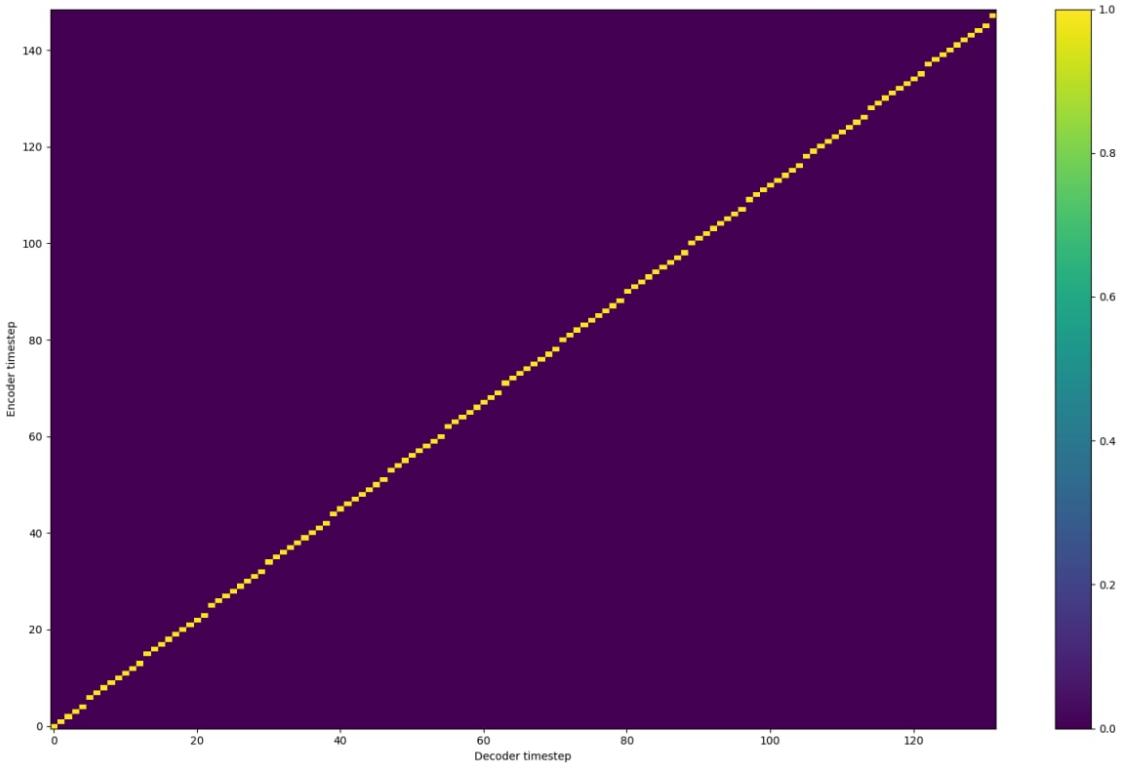

Рис. 3.6. Механізм уваги в початку навчання на етапі перевірки

На наступних графіках зображено механізм уваги в кінці навчання та на етапі перевірки.

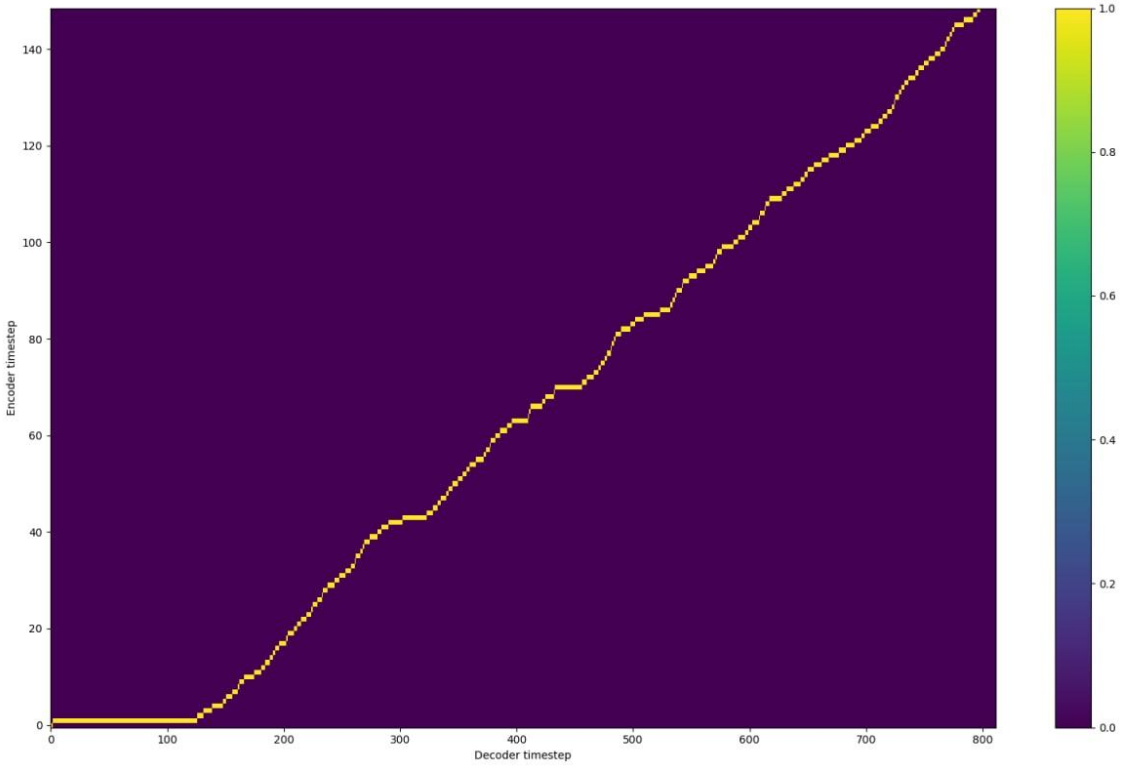

Рис. 3.6. Механізм уваги в кінці навчання

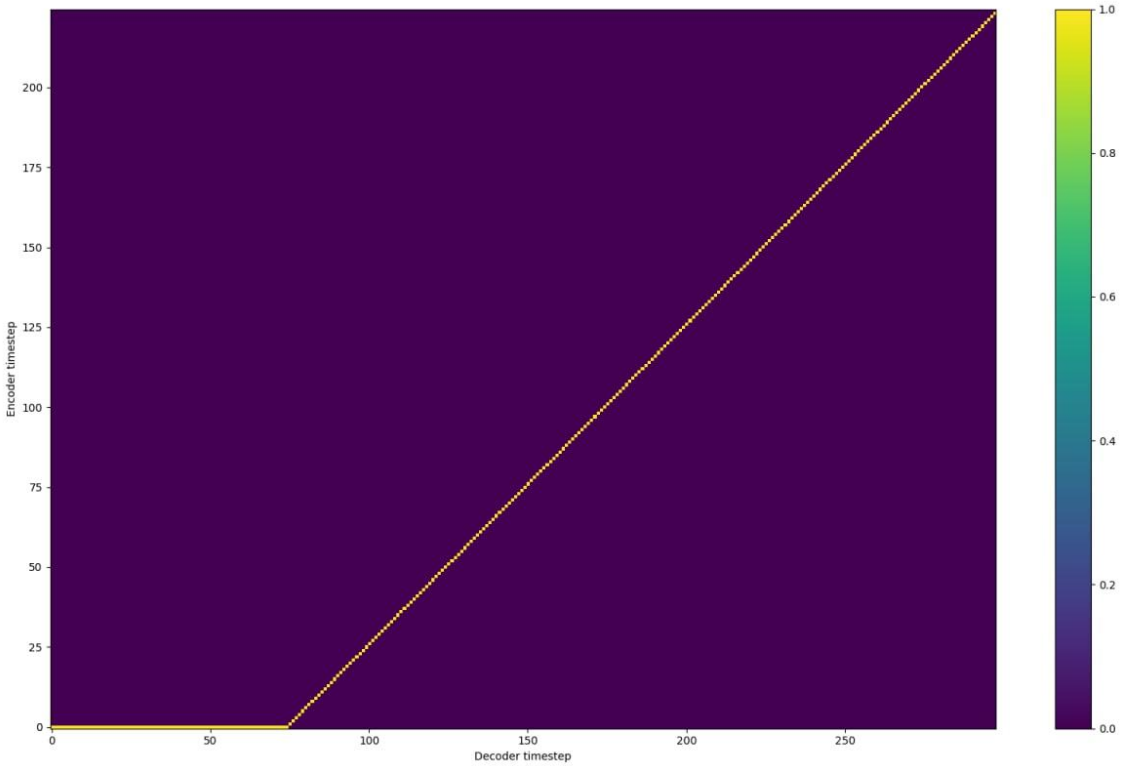

Рис. 3.7. Механізм уваги в кінці навчання на етапі перевірки

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було проведено дослідження моделей синтезу мовлення, модифіковано модель FastSpeech із проекту TensorFlowTTS для підтримки української мови та сучасної версії бібліотеки TensorFlow. Оцінювання якості отриманих результатів синтезу мовлення було проведено за шкалою MOS. Для візуалізації отриманих результатів було представлено таблиці та графіки.

Для розв'язання поставлених задач використані: нейронні мережі, хмарні технології, функціональне програмування, моделі трансляції тексту, методи обробки сигналів та операційна система Unix з використанням Wsl.

Результат кваліфікаційної роботи може використатися в сферах, що потребують синтез українськой мови, а також для подальшого розвитку у дослідженнях створення моделей на базі сімейства нейронних мереж FastSpeech.

Подальші дослідження в даній сфері дозволять створювати моделі трансляції тексту до мовлення з урахуванням отриманого досвіду та з покращенням частин реалізації, наведених раніше, а саме: якості синтезу завдяки покращенню мовного набору даних.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, et al. Tacotron: Towards end-to-end speech synthesis. arXiv preprint arXiv:1703.10135, 2017
2. Jonathan Shen, Ruoming Pang, Ron J Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, et al. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 4779–4783. IEEE, 2018.
3. Wei Ping, Kainan Peng, Andrew Gibiansky, Sercan O. Arik, Ajay Kannan, Sharan Narang, Jonathan Raiman, and John Miller. Deep voice 3: 2000-speaker neural text-to-speech. In International Conference on Learning Representations, 2018.
4. Wei Ping, Kainan Peng, and Jitong Chen. Clarinet: Parallel wave generation in end-to-end text-to-speech. In International Conference on Learning Representations, 2019.
5. Daniel Griffin and Jae Lim. Signal estimation from modified short-time fourier transform. IEEE Transactions on Acoustics, Speech, and Signal Processing, 32(2):236–243, 1984.
6. Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. SSW, 125, 2016.
7. Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C Cobo, Florian Stimberg, et al. Parallel wavenet: Fast high-fidelity speech synthesis. arXiv preprint arXiv:1711.10433, 2017.

8. Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 3617–3621. IEEE, 2019.
9. Andrew J Hunt and Alan W Black. Unit selection in a concatenative speech synthesis system using a large speech database. In 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings, volume 1, pages 373–376. IEEE, 1996.
10. Naihan Li, Shujie Liu, Yanqing Liu, Sheng Zhao, Ming Liu, and Ming Zhou. Close to human quality tts with transformer. arXiv preprint arXiv:1809.08895, 2018.
11. Jonathan Shen, Ruoming Pang, Ron J Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, et al. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In 2018 IEEE International Conf
12. Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In Advances in Neural Information Processing Systems, pages 1171–1179, 2015.
13. Wei Ping, Kainan Peng, Andrew Gibiansky, Sercan O. Arik, Ajay Kannan, Sharan Narang, Jonathan Raiman, and John Miller. Deep voice 3: 2000-speaker neural text-to-speech. In International Conference on Learning Representations, 2018.
14. Jonathan Shen, Ruoming Pang, Ron J Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, et al. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 4779–4783. IEEE, 2018.
15. Yiren Wang, Fei Tian, Di He, Tao Qin, ChengXiang Zhai, and Tie-Yan Liu. Non-autoregressive machine translation with auxiliary regularization. In AAAI, 2019.

16. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems, pages 5998–6008, 2017.
17. Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1243–1252. JMLR. org, 2017.
18. Michael McAuliffe, Michaela Socolof, Sarah Mihuc, Michael Wagner, and Morgan Sonderegger. Montreal forced aligner: Trainable text-speech alignment using kaldi. In Interspeech, pp. 498– 502, 2017.
19. Andrew Gibiansky, Sercan Arik, Gregory Diamos, John Miller, Kainan Peng, Wei Ping, Jonathan Raiman, and Yanqi Zhou. Deep voice 2: Multi-speaker neural text-to-speech. In Advances in neural information processing systems, pp. 2962–2970, 2017.
20. FastSpeech: Fast, Robust and Controllable Text to Speech, NeurIPS 2019

Додаток А

КОД ПРОГРАМИ

```
class TFFastSpeechEmbeddings(tf.keras.layers.Layer):
    """Construct charactor/phoneme/positional/speaker embeddings."""

    def __init__(self, config, **kwargs):
        """Init variables."""
        super().__init__(**kwargs)
        self.vocab_size = config.vocab_size
        self.hidden_size                                                    =
config.encoder_self_attention_params.hidden_size
        self.initializer_range = config.initializer_range
        self.config = config

        self.position_embeddings = TFEmbedding(
            config.max_position_embeddings + 1,
            self.hidden_size,
            weights=[
                self._sincos_embedding(
                    self.hidden_size,
self.config.max_position_embeddings
                )
            ],
            name="position_embeddings",
            trainable=False,
        )

        if config.n_speakers > 1:
            self.encoder_speaker_embeddings = TFEmbedding(
                config.n_speakers,
                self.hidden_size,

embeddings_initializer=get_initializer(self.initializer_range),
                name="speaker_embeddings",
            )
```

```
            self.speaker_fc = tf.keras.layers.Dense(
                units=self.hidden_size, name="speaker_fc"
            )

    def build(self, input_shape):
        """Build shared charactor/phoneme embedding layers."""
        with tf.name_scope("charactor_embeddings"):
            self.charactor_embeddings = self.add_weight(
                "weight",
                shape=[self.vocab_size, self.hidden_size],
                initializer=get_initializer(self.initializer_range),
            )
        super().build(input_shape)

    def call(self, inputs, training=False):
        """Get charactor embeddings of inputs.
        Args:
            1. charactor, Tensor (int32) shape [batch_size, length].
            2. speaker_id, Tensor (int32) shape [batch_size]
        Returns:
            Tensor (float32) shape [batch_size, length, embedding_size].
        """
        return self._embedding(inputs, training=training)

    def _embedding(self, inputs, training=False):
        """Applies embedding based on inputs tensor."""
        input_ids, speaker_ids = inputs

        input_shape = tf.shape(input_ids)
        seq_length = input_shape[1]

        position_ids = tf.range(1, seq_length + 1,
dtype=tf.int32)[tf.newaxis, :]

        # create embeddings
        inputs_embeds = tf.gather(self.charactor_embeddings, input_ids)
        position_embeddings = self.position_embeddings(position_ids)
```

```
        # sum embedding
        embeddings = inputs_embeds + tf.cast(position_embeddings,
inputs_embeds.dtype)
        if self.config.n_speakers > 1:
            speaker_embeddings                                    =
self.encoder_speaker_embeddings(speaker_ids)
            speaker_features                                      =
tf.math.softplus(self.speaker_fc(speaker_embeddings))
            # extended speaker embeddings
            extended_speaker_features = speaker_features[:, tf.newaxis,
:]
            embeddings += extended_speaker_features

        return embeddings

    def _sincos_embedding(
        self, hidden_size, max_positional_embedding,
    ):
        position_enc = np.array(
            [
                [
                    pos / np.power(10000, 2.0 * (i // 2) / hidden_size)
                    for i in range(hidden_size)
                ]
                for pos in range(max_positional_embedding + 1)
            ]
        )

        position_enc[:, 0::2] = np.sin(position_enc[:, 0::2])
        position_enc[:, 1::2] = np.cos(position_enc[:, 1::2])

        # pad embedding.
        position_enc[0] = 0.0

        return position_enc

    def resize_positional_embeddings(self, new_size):
        self.position_embeddings = TFEmbedding(
```

```
            new_size + 1,
            self.hidden_size,
            weights=[self._sincos_embedding(self.hidden_size,
new_size)],
            name="position_embeddings",
            trainable=False,
        )


class TFFastSpeechSelfAttention(tf.keras.layers.Layer):
    """Self attention module for fastspeech."""

    def __init__(self, config, **kwargs):
        """Init variables."""
        super().__init__(**kwargs)
        if config.hidden_size % config.num_attention_heads != 0:
            raise ValueError(
                "The hidden size (%d) is not a multiple of the number of
attention "
                "heads        (%d)"        %        (config.hidden_size,
config.num_attention_heads)
            )
        self.output_attentions = config.output_attentions
        self.num_attention_heads = config.num_attention_heads
        self.all_head_size      =      self.num_attention_heads      *
config.attention_head_size

        self.query = tf.keras.layers.Dense(
            self.all_head_size,

kernel_initializer=get_initializer(config.initializer_range),
            name="query",
        )
        self.key = tf.keras.layers.Dense(
            self.all_head_size,

kernel_initializer=get_initializer(config.initializer_range),
            name="key",
```

```
        )
        self.value = tf.keras.layers.Dense(
            self.all_head_size,

kernel_initializer=get_initializer(config.initializer_range),
            name="value",
        )

        self.dropout                                                    =
tf.keras.layers.Dropout(config.attention_probs_dropout_prob)
        self.config = config

    def transpose_for_scores(self, x, batch_size):
        """Transpose to calculate attention scores."""
        x = tf.reshape(
            x,
            (batch_size,          -1,          self.num_attention_heads,
self.config.attention_head_size),
        )
        return tf.transpose(x, perm=[0, 2, 1, 3])

    def call(self, inputs, training=False):
        """Call logic."""
        hidden_states, attention_mask = inputs

        batch_size = tf.shape(hidden_states)[0]
        mixed_query_layer = self.query(hidden_states)
        mixed_key_layer = self.key(hidden_states)
        mixed_value_layer = self.value(hidden_states)

        query_layer    =    self.transpose_for_scores(mixed_query_layer,
batch_size)
        key_layer      =      self.transpose_for_scores(mixed_key_layer,
batch_size)
        value_layer    =    self.transpose_for_scores(mixed_value_layer,
batch_size)
```

```
        attention_scores = tf.matmul(query_layer, key_layer,
transpose_b=True)
        dk = tf.cast(
            tf.shape(key_layer)[-1], attention_scores.dtype
        )  # scale attention_scores
        attention_scores = attention_scores / tf.math.sqrt(dk)

        if attention_mask is not None:
            # extended_attention_masks for self attention encoder.
            extended_attention_mask = attention_mask[:, tf.newaxis,
tf.newaxis, :]
            extended_attention_mask = tf.cast(
                extended_attention_mask, attention_scores.dtype
            )
            extended_attention_mask = (1.0 - extended_attention_mask) *
-1e9
            attention_scores = attention_scores +
extended_attention_mask

        # Normalize the attention scores to probabilities.
        attention_probs = tf.nn.softmax(attention_scores, axis=-1)
        attention_probs = self.dropout(attention_probs,
training=training)

        context_layer = tf.matmul(attention_probs, value_layer)
        context_layer = tf.transpose(context_layer, perm=[0, 2, 1, 3])
        context_layer = tf.reshape(context_layer, (batch_size, -1,
self.all_head_size))

        outputs = (
            (context_layer, attention_probs)
            if self.output_attentions
            else (context_layer,)
        )
        return outputs


class TFFastSpeechSelfOutput(tf.keras.layers.Layer):
```

```
    """Fastspeech output of self attention module."""

    def __init__(self, config, **kwargs):
        """Init variables."""
        super().__init__(**kwargs)
        self.dense = tf.keras.layers.Dense(
            config.hidden_size,

kernel_initializer=get_initializer(config.initializer_range),
            name="dense",
        )
        self.LayerNorm = tf.keras.layers.LayerNormalization(
            epsilon=config.layer_norm_eps, name="LayerNorm"
        )
        self.dropout                                                    =
tf.keras.layers.Dropout(config.hidden_dropout_prob)

    def call(self, inputs, training=False):
        """Call logic."""
        hidden_states, input_tensor = inputs

        hidden_states = self.dense(hidden_states)
        hidden_states = self.dropout(hidden_states, training=training)
        hidden_states = self.LayerNorm(hidden_states + input_tensor)
        return hidden_states


class TFFastSpeechAttention(tf.keras.layers.Layer):
    """Fastspeech attention module."""

    def __init__(self, config, **kwargs):
        """Init variables."""
        super().__init__(**kwargs)
        self.self_attention     =     TFFastSpeechSelfAttention(config,
name="self")
        self.dense_output       =       TFFastSpeechSelfOutput(config,
name="output")
```

```
    def call(self, inputs, training=False):
        input_tensor, attention_mask = inputs

        self_outputs = self.self_attention(
            [input_tensor, attention_mask], training=training
        )
        attention_output = self.dense_output(
            [self_outputs[0], input_tensor], training=training
        )
        masked_attention_output = attention_output * tf.cast(
            tf.expand_dims(attention_mask,                                2),
dtype=attention_output.dtype
        )
        outputs = (masked_attention_output,) + self_outputs[
            1:
        ]  # add attentions if we output them
        return outputs


class TFFastSpeechIntermediate(tf.keras.layers.Layer):
    """Intermediate representation module."""

    def __init__(self, config, **kwargs):
        """Init variables."""
        super().__init__(**kwargs)
        self.conv1d_1 = tf.keras.layers.Conv1D(
            config.intermediate_size,
            kernel_size=config.intermediate_kernel_size,

kernel_initializer=get_initializer(config.initializer_range),
            padding="same",
            name="conv1d_1",
        )
        self.conv1d_2 = tf.keras.layers.Conv1D(
            config.hidden_size,
            kernel_size=config.intermediate_kernel_size,

kernel_initializer=get_initializer(config.initializer_range),
```

```
            padding="same",
            name="conv1d_2",
        )
        if isinstance(config.hidden_act, str):
            self.intermediate_act_fn = ACT2FN[config.hidden_act]
        else:
            self.intermediate_act_fn = config.hidden_act

    def call(self, inputs):
        """Call logic."""
        hidden_states, attention_mask = inputs

        hidden_states = self.conv1d_1(hidden_states)
        hidden_states = self.intermediate_act_fn(hidden_states)
        hidden_states = self.conv1d_2(hidden_states)

        masked_hidden_states = hidden_states * tf.cast(
            tf.expand_dims(attention_mask,                          2),
dtype=hidden_states.dtype
        )
        return masked_hidden_states


class TFFastSpeechOutput(tf.keras.layers.Layer):
    """Output module."""

    def __init__(self, config, **kwargs):
        """Init variables."""
        super().__init__(**kwargs)
        self.LayerNorm = tf.keras.layers.LayerNormalization(
            epsilon=config.layer_norm_eps, name="LayerNorm"
        )
        self.dropout                                                =
tf.keras.layers.Dropout(config.hidden_dropout_prob)

    def call(self, inputs, training=False):
        """Call logic."""
        hidden_states, input_tensor = inputs
```

```
hidden_states = self.dropout(hidden_states, training=training)
hidden_states = self.LayerNorm(hidden_states + input_tensor)
return hidden_states
```

Додаток Б

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Дипломна_робота(Вахрушин Є.В.).docx	Пояснювальна записка роботи. Документ Word.
Дипломна_робота(Вахрушин Є.В.).pdf	Пояснювальна записка роботи. Документ PDF.
Програма	
Project.zip	Архів. Містить код програми.
Презентація	
Презентація(Вахрушин Є.В.).pptx	Презентація дипломної роботи.