VIETNAM NATIONAL UNIVERSITY, HANOI
**INTERNATIONAL SCHOOL**

# FINAL PROJECT

# GROUP 10

# A Deep Learning Approach to Emotion-Driven Music Generation from Vietnamese Text

**Lecturer: Ha Manh Hung**

**Course Name: Artificial Intelligence**

**Course ID: INS3080**

**Group leader : Nguyễn Duy Thức**

***Hanoi, June 23rd 2025***

# LIST OF MEMBERS AND WORK BREAKDOWN STRUCTURE

| No | Name | Student code | Class | Task | % contribution |
|---|---|---|---|---|---|
| 1 | Nguyen Duy Thuc | 21070278 | BDA2021A | Training Logistic Regression for emotion recognition<br>Developing Music Generation Model with emotion constraints | 20 |
| 2 | Mac Pham Thien Long | 22070503 | MIS2022B | Training PhoBERT for emotion recognition<br>Collected and labeled the Vietnamese emotion dataset<br>Applied data augmentation techniques on the VGMIDI music dataset | 20 |
| 3 | Nguyen Dinh Duy | 22 070367 | MIS2022B | Building a web application that transfer Vietnamese text into music<br>Training BiLSTM for emotion recognition<br>Collecting and labeling Vietnamese emotion dataset | 20 |
| 4 | Nguyen Thi Ngoc Anh | 22070457 | MIS2022B | Finding and synthesize literature of previous works<br>EDA Vietnamese emotion dataset<br>Collecting and labeling Vietnamese emotion dataset | 20 |
| 5 | Vu Hong Thuy | 22070360 | BDA2022A | Developing the main pipeline<br>Collecting and labeling Vietnamese emotion dataset | 20 |
| TOTAL | | | | | 100% |

# ACKNOWLEDGEMENTS

We would like to extend our sincere thanks to our course instructor, PhD. Ha Manh Hung, for his insightful guidance, valuable feedback, and continued support throughout the development of this project. His mentorship played a vital role in shaping the direction of our work and helping us maintain academic rigor.

We are also grateful to the Faculty of Applied Sciences, VNU-IS, for providing the learning environment, academic resources, and infrastructure necessary to complete this assignment successfully.

In addition, we wish to thank any individuals or institutions whose tools, publications, or datasets have contributed indirectly to the progress of this project.

Finally, we truly appreciate the dedication, accountability, and collaboration of every team member. Their consistent effort and support made this challenging yet rewarding journey not only possible, but meaningful.

# ABSTRACT

This project develops an end-to-end deep learning system for generating emotionally-driven music from Vietnamese text. The core of the system is a robust Natural Language Processing (NLP) module designed to overcome the challenge of accurately interpreting emotion in the Vietnamese language. For this purpose, we constructed a custom dataset of over 37,000 Vietnamese social media comments, manually annotated with six basic emotions. A fine-tuned PhoBERT model, pre-trained specifically on Vietnamese text, was employed for the emotion classification task, achieving a high accuracy of 90%. The classified emotion is then mapped to a valence-arousal vector. This vector serves as a conditional input for an LSTM-based generative model, which composes a musical piece in MIDI format that reflects the text's emotional tone. This work establishes a novel pipeline for Vietnamese Creative AI, successfully bridging linguistic analysis and musical generation, and creating a strong foundation for future research in cross-modal emotional expression.

# I. Introduction

## 1.1. Problem Statement

Music is becoming a powerful tool for personalized experiences in entertainment, content creation, and mental health support. However, generating emotionally aligned music from Vietnamese text remains a technical challenge due to the language's rich context and expressiveness.

The core issues in converting Vietnamese text into emotion-driven music are:

**(1) Understanding and accurately extracting the diverse emotional features from Vietnamese text**: This is the first step, where deep learning models are used to analyze Vietnamese text and identify underlying emotions such as happiness, sadness, anger, calmness, etc. This process leverages linguistic features such as vocabulary, semantics, syntax, and context, which are highly specific to the Vietnamese language. Modern models such as LSTM, CNN, BERT, or the combination of CNN-LSTM have proven effective in extracting and classifying emotions from text, especially with well-labeled datasets.

**(2) Generating novel, natural music that truly reflects the desired emotions:** After identifying the emotions in the text, the system will use generative models such as GAN, LSTM, or Transformer to create music with tonal qualities matching the emotional state. This is a crucial step in transforming linguistic emotions into a genuine musical experience.

The research problem is to design an integrated system capable of converting Vietnamese text into an emotion-driven musical experience, creating a bridge between language and sound in the digital content space.

This problem is not only significant in the Creative AI field but also opens up new research directions for AI systems for the Vietnamese language, which still faces challenges regarding data availability and modeling platforms.

Successfully solving this problem is not only a technological innovation but also a significant contribution to promoting automated creativity in the music industry. By combining the ability to recognize emotions from language with generative music models, the system opens up an entirely new approach to digital art - where the emotional content of language is transformed into a profound and relatable auditory experience. From a technological standpoint, the system establishes a breakthrough form of interaction between humans and machines, where artificial intelligence not only analyzes data but also has the ability to "understand" and "feel" emotional content, then express it through music - a non-verbal form of expression. This not only advances the field of Creative AI but also paves the way for the development of more specialized AI systems for the Vietnamese language, which still faces many challenges in terms of data availability and modeling infrastructure.

This important role is clearly demonstrated through its diverse and valuable **practical applications:**

- In Entertainment & Media: The system can automate the creation of emotional background music for films, video games, advertisements, or platforms that adapt music based on the user's mood [1]
- In Psychological Therapy & Mental Health Care: Personalized music based on emotions can be used to support individuals with psychological issues or emotional disorders, helping them express and regulate their moods effectively [2]
- In Automated Content Creation & Social Media: It can automatically generate music matching the emotional tone of textual content (captions, scripts, posts), supporting platforms like TikTok, YouTube Shorts, podcasts, or services offering personalized user experiences [3]

Beyond immediate applications, the project's potential also opens up breakthrough directions in AI research tailored to the linguistic and cultural characteristics of Vietnam. In a context where most existing models are built and trained on English-language data, developing a system that understands and expresses Vietnamese emotions through music represents a significant advancement in the localization of AI technology. Furthermore, when scaled up, the system can play a key role in building deep learning models for emotion regulation, thereby enabling the generation of more diverse and adaptive musical compositions based on user context, situations, and moods. This lays the foundation for developing personalized AI applications, ranging from therapeutic music and education to digital content creation. Finally, this research also creates opportunities to develop multi-modal AI models, where language, emotion, and sound are tightly integrated, serving fields such as human–machine interaction, social robotics, emotional assistants, music education, and interactive art.

## 1.2. Literature Review

### 1.2.1. Emotion Classification Techniques for Vietnamese Text

Emotion classification from text is a crucial step in building a system that generates music based on linguistic emotions. In the field of Natural Language Processing (NLP), there are three main approaches commonly used for emotion classification: (1) lexicon-based methods, (2) classical machine learning, and (3) deep learning. Each approach has its own characteristics, advantages, and limitations, as summarized in the table below:

Table 1: Comparison of Emotion Classification Methods for Vietnamese Text

| Method | Description | Advantages | Limitations |
|---|---|---|---|
| **Lexicon-based Emotion Dictionary** | Based on a list of emotionally charged words, such as "sad", "happy", "hate" etc. Scoring systems calculate the overall emotional intensity in the text. | Easy to build, does not require training data, works with small datasets. | Does not understand context, prone to errors in cases of metaphor, sarcasm, or ambiguity; low accuracy. |
| **Classical Machine Learning (SVM, Naive Bayes, Random Forest)** | Based on manually extracted features such as TF-IDF, Bag of Words, combined with linear classification models or decision trees. (UIT-VSMEC Corpus)[4] | Simple, performs well on small to medium-sized datasets; easy to control and adjust. | Lacks the ability to represent complex semantics, does not capture context well, does not scale well. |
| **Deep Learning (CNN, BiLSTM, BERT/PhoBERT, ViSoBERT)** | Uses end-to-end deep learning models to automatically learn features and context. PhoBERT and ViSoBERT [5] are pre-trained models for Vietnamese. | High accuracy, understands context and complex semantic structures, adapts well to Vietnamese text. | Requires large resources for training, full labeled data, and results are hard to interpret. |

### 1.2.2. Deep Learning-Based Music Generation Methods

In recent years, the advancement of deep learning has enabled new possibilities in automatic music generation, particularly within the domain of Creative AI. Three prominent approaches commonly used for this task are: (1) Generative Adversarial Networks (GAN), (2) Long Short-Term Memory (LSTM), and (3) Transformer models. Each of these methods offers a unique perspective on how to process and generate musical data, with distinct strengths and limitations in modeling time, melody, and complex musical structures. The following table provides a comparative overview of these methods:

Table 2: Comparison of Deep Learning-Based Music Generation Methods

| Method | Description | Advantages | Limitations |
|---|---|---|---|
| **GAN (Generative Adversarial Network)** | Uses two adversarial neural networks (Generator and Discriminator) to generate highly realistic music data [6]. Some prominent models include MuseGAN [7] and MidiNet. | Capable of generating unique, flexible music based on style or emotion; effective in creating highly creative musical segments. | Very difficult to train stably; requires a large amount of labeled music data; prone to "mode collapse" if not carefully controlled. |
| **LSTM (Long Short-Term Memory)** | A type of RNN that can remember long-term information in sequences, suitable for learning musical note sequences over time. | Simple implementation, easy to train, suitable for data with clear sequential structure. | Limited in modeling long-term relationships or complex musical structures such as harmony and multi-instrumentation. |
| **Transformer** | Based on the self-attention mechanism, this model can learn relationships between musical notes regardless of proximity, for example, Music Transformer [8] by Google. | High representation ability, good modeling of long-term relationships, generates complex and multi-instrumental musical segments. | Requires significant resources for training; difficult to optimize for short musical segments; needs careful handling of timing and rhythmic structure. |

### 1.2.3. Research Gaps and Motivation

Although individual components such as emotion recognition and music generation have been widely studied globally, there are still clear gaps, especially in the context of Vietnamese:

**(1) Lack of synchronized integration of the two modules:** Most existing studies tend to isolate these two tasks, focusing either on identifying emotions from text or on generating music based on predefined styles or features. There is a clear lack of end-to-end systems that automatically convert emotional content in Vietnamese text into corresponding musical output. In many cases, the integration of emotion into music generation is done manually or heuristically, without deep learning-based mechanisms that learn this mapping directly from data.

**(2) No emotion-conditioned LSTM-based model for Vietnamese music generation:** While Although LSTM models have been successfully applied to emotion-conditioned music generation in high-resource languages such as English - for example, generating melodies from lyrics or symbolic music conditioned on valence-arousal - there is currently no known research that explores the use of LSTM for generating music from Vietnamese emotional text. Notable studies such as "An Emotional Symbolic Music Generation System based on LSTM" [9], "Conditional LSTM-GAN for Melody Generation from Lyrics" [10], and "LSTM-Based Melody Harmonization with Hierarchical VAE" [11] have demonstrated the potential of LSTM in generating expressive music conditioned on emotional inputs. However, these models are primarily developed using English-language datasets and symbolic music formats, while Vietnamese - a tonal and morphologically distinct language with rich emotional nuance - remains unrepresented in this field. As a result, applying emotion-conditioned LSTM architectures to generate music directly from Vietnamese emotional text represents a largely unexplored but highly promising research direction

## 1.3. Project Objectives

The overall objective of this project is to build an Artificial Intelligence (AI) system capable of generating emotionally aligned music from Vietnamese text data. This system aims to foster personalized content creation and deepen the emotional interaction between humans and machines by creating music that is unique to the user's own emotions. To achieve this goal, the project is structured around the following specific, measurable objectives:

- Build a complete modular framework that transforms Vietnamese input text into expressive instrumental music. The pipeline consists of: (1) Text Preprocessing and Emotion Extraction, (2) Emotion-to-Vector Mapping, (3) LSTM-based Music Generation, and (4) Music Output and Refinement. Each module plays a critical role in ensuring that the final output reflects the emotional nuances embedded in the original text.
- Collect and construct a Vietnamese emotion-labeled dataset for both training and evaluation. The dataset will contain at least 5,000 high-quality samples annotated with six key emotional categories: Happiness (Vui vẻ), Sadness (Buồn bã), Anger (Tức giận), Surprise (Ngạc nhiên), Fear (Sợ hãi), and Disgust (Ghê tởm). A rigorous preprocessing pipeline will be applied, including diacritic restoration, normalization of informal writing (e.g., teencode), and segmentation, to ensure linguistic consistency and clarity.
- Develop a high-performing deep learning model for emotion classification from Vietnamese text. Techniques such as PhoBERT will be evaluated and compared, with the goal of achieving at least 90% accuracy on the validation/test set. Evaluation metrics will include Accuracy, Precision, Recall, F1-score, and Confusion Matrix to ensure robustness across emotion classes.
- Implement an LSTM-based music generation model that takes emotion vectors as input conditions and composes music in MIDI format. The model will learn musical structure and expression consistent with emotional features, aiming to generate emotionally coherent and musically diverse works. Potential architectures include Music LSTM for symbolic music generation.

Ultimately, the system aims to convert textual emotional content into rich and resonant musical expressions, enabling a wide range of applications in emotional entertainment, digital media, and mental wellness support.

# II. Methodology

## 2.1.Project Pipeline

The proposed system is designed with an end-to-end pipeline architecture, comprising a sequence of processing modules that work in succession to transform Vietnamese text into emotionally reflective music. Specifically:

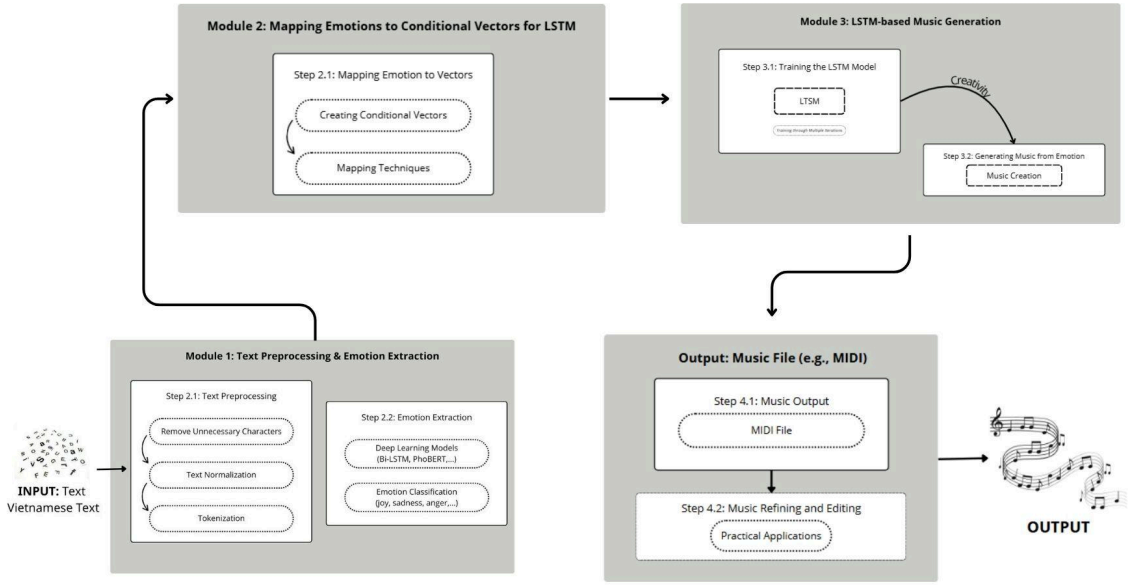



Figure 1. Overall System Architecture for Emotion-Driven Music Generation from Vietnamese Text

- **Module 1** – Text Preprocessing & Emotion Extraction
  Input Vietnamese text undergoes a cleaning process that includes removing special characters, normalizing "teencode" (slang), tokenization, and Unicode encoding. Subsequently, a deep learning model such as Bi-LSTM or PhoBERT is used to recognize the emotion, classifying it into one of six labels: Happiness, Sadness, Anger, Surprise, Fear, or Disgust.
- **Module 2** – Emotion-to-Conditional Vector Mapping for the LSTM
  After recognizing the emotion, it is translated into a condition vector - a numerical representation that captures the emotional characteristics of the input. This vector reflects the valence (positivity/negativity) and arousal (intensity) levels of the detected emotion, often based on Russell's Circumplex Model of Affect. This conditional vector will serve as an input signal to control the emotional tone of the music generated by the LSTM.

- **Module 3** – Music Generation using a LSTM Model
  A LSTM-based model (such as Music LSTM) is employed to generate symbolic music (e.g., in MIDI format). The model receives the emotion condition vector as an input context and generates note sequences that reflect the emotional dynamics of the input text. Due to its ability to model long-term temporal dependencies and expressive patterns, the LSTM ensures musical compositions that are emotionally aligned, coherent, and structurally rich.
- **Module 4** – Music Export (MIDI) and Post-processing
  The output is a music file in MIDI format. This file is then refined for aspects like volume and basic rhythm before being applied to specific purposes, such as video background music, user experience enhancement, or emotional therapy.
- **Module 5** – Application and Evaluation
  The generated musical outputs are evaluated through qualitative surveys (using Likert scales) and quantitative metrics (e.g., IS/FID, if applicable). Concurrently, the system is deployed as a simple web demo, allowing users to input text and listen to the corresponding music.

## 2.2. Data Collection and Preprocessing

### 2.2.1. Data Collection

**a.1. Data Source**

The dataset for our project was collected from a public and natural source: user comments on the YouTube platform. Specifically, the data was extracted from the comment sections of several public videos containing entertaining or controversial content that attracted a large number of views and interactions. A representative example is the video at: https://www.youtube.com/watch?v=BwN3NiZt-PU. Selecting videos with a high potential to evoke strong emotions increases the likelihood of obtaining comments that clearly express sentiment, such as "trời ơi xúc động ghê" ("oh my god, so touching") or "mắc cười quá!" ("this is so funny!").

YouTube was chosen as the data source because it is one of the most popular social media platforms in Vietnam, where users are free to express their opinions, reactions, and emotions in a natural and unrestricted way - without being influenced by structured surveys or pre-defined frameworks. The comment section on YouTube, in particular, contains various forms of informal language, filled with personal tones, humor, anger, sympathy, or sarcasm – all of which are well-suited for sentiment analysis and emotion extraction tasks.

```
video_url = "https://www.youtube.com/watch?v=BwN3NiZt-PU"
```

Figure 2: YouTube Video URL for Data Collection

Once collected, the data was in raw text format and filtered to include only comments written in Vietnamese with diacritics, ensuring linguistic consistency and higher accuracy in subsequent natural language processing tasks. This step is critical for developing a sentiment analysis model specifically tailored for Vietnamese, accurately capturing real-world language usage and emotional expression within the Vietnamese online community.

```
def is_vietnamese(text):
    return bool(re.search(r'[\u00C0-\u017F\u1E00-\u1EFF]', text))
```

Figure 3: Vietnamese Comment Filtering

**a.2. Data Collection Process**

The data collection process was automated using the **Web Scraping** method, implemented in the Python programming language with the help of the **Selenium** library — a powerful tool that enables automated browser control. This approach was chosen because YouTube is a **dynamic website**, where comment content is progressively loaded via JavaScript as users interact with the page. Therefore, conventional static scraping techniques would not be effective in capturing the full set of data.

To overcome this challenge, our group developed a script that emulates real user behavior to ensure comprehensive data retrieval. The process begins with the initialization of an automated Chrome browser session, which then navigates to the predefined YouTube video URL.

```python
def main():
    options = Options()
    options.add_argument("--start-maximized")
    options.add_argument('--disable-dev-shm-usage')
    options.add_argument('--no-sandbox')
    options.add_argument('log-level=3')

    driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=options)
    wait = WebDriverWait(driver, 15)

    video_url = "https://www.youtube.com/watch?v=xwDXSWnebF0&t=35s"

    print(f"Đang truy cập video: {video_url}")
    driver.get(video_url)
```

Figure 4 : Browser automation with predefined YouTube video URL

Once the target page is loaded, the script continuously scrolls down to trigger YouTube's infinite scroll feature. This mechanism forces the platform to load additional comment threads as if a human user were interacting with the page. The script monitors the scroll height of the page and only stops when no new content is detected after several consecutive checks, ensuring that all visible comments are fully loaded.

```python
def scroll_to_bottom(driver, scroll_pause_time=2, max_scroll_attempts=100):

    print("\nBắt đầu cuộn trang để tải bình luận chính...")
    last_height = driver.execute_script("return document.documentElement.scrollHeight")
    attempts = 0

    for i in range(max_scroll_attempts):
        driver.execute_script("window.scrollTo(0, document.documentElement.scrollHeight);")
        time.sleep(scroll_pause_time)
        new_height = driver.execute_script("return document.documentElement.scrollHeight")

        if new_height == last_height:
            attempts += 1
            if attempts >= 3: # Thử 3 lần liên tiếp không thay đổi thì dừng
                print("Đã cuộn đến cuối trang.")
                break
        else:
            attempts = 0 # Reset bộ đếm nếu có nội dung mới

        last_height = new_height
        if (i + 1) % 5 == 0:
            print(f"Đã cuộn {i + 1} lần...")
    print("Hoàn tất cuộn trang.")
```

Figure 5 : Scrolling to load full comments

After loading the main comments, the script searches for and clicks all "View replies" buttons. This step is essential to ensure that no hidden replies are missed, thereby enhancing the dataset's completeness and richness. JavaScript is used to trigger these clicks, improving stability and compatibility across different environments.

```python
def expand_all_replies(driver, wait):
    print("\nBắt đầu mở rộng các bình luận trả lời...")
    try:
        reply_buttons = wait.until(EC.presence_of_all_elements_located(
            (By.CSS_SELECTOR, "#more-replies ytd-button-renderer")
        ))

        clicked_count = 0
        for button in reply_buttons:
            try:
                driver.execute_script("arguments[0].scrollIntoView(true);", button)
                time.sleep(20)
                driver.execute_script("arguments[0].click();", button)
                clicked_count += 1
                if clicked_count % 10 == 0:
                    print(f"Đã mở rộng {clicked_count} nhóm trả lời...")
                time.sleep(1) # Chờ để các trả lời được tải
            except Exception as e:
                # Bỏ qua nếu nút không còn tồn tại hoặc không thể click
                pass
        print(f"Hoàn tất! Đã mở rộng tổng cộng {clicked_count} nhóm trả lời.")
    except TimeoutException:
        print("Không tìm thấy nút 'Xem câu trả lời' nào, hoặc đã mở rộng hết.")
```

Figure 6 : Expanding hidden replies

Once all main comments and replies have been loaded and expanded, the script iterates through each comment element on the page to extract the textual content. Immediately after extraction, each comment is evaluated using a regular expression (regex) to enforce the Vietnamese-language filtering criterion described in the Data Source section. Only comments containing Vietnamese characters with diacritics are retained, while irrelevant content is discarded.

```python
comment_threads = driver.find_elements(By.CSS_SELECTOR, "ytd-comment-thread-renderer")
if not comment_threads:
    print("⚠️ Không tìm thấy bình luận nào. Kết thúc.")
    driver.quit()
    return
print(f"🔎 Tìm thấy {len(comment_threads)} luồng bình luận. Đang xử lý...")
for thread in comment_threads:
    try:
        main_comment_element = thread.find_element(By.CSS_SELECTOR, "#comment #content-text")
        main_comment_text = main_comment_element.text.strip()
        if is_vietnamese(main_comment_text):
            all_comments_data.append([index, main_comment_text, "Bình luận chính"])
            index += 1

        reply_elements = thread.find_elements(By.CSS_SELECTOR, "#replies #content-text")
        for reply_element in reply_elements:
            reply_text = reply_element.text.strip()
            if is_vietnamese(reply_text):
                all_comments_data.append([index, reply_text, "Bình luận trả lời"])
                index += 1
    except StaleElementReferenceException:
        continue
    except Exception as e:
        print(f"Lỗi khi xử lý một luồng bình luận: {e}")
        continue
```

Figure 7: Extracting and filtering Vietnamese-language comments

Finally, all valid Vietnamese comments are collected and saved into an Excel file (.xlsx format), clearly distinguishing between “Main Comments” and “Replies.” Storing the data in Excel facilitates easy management, review, and preparation for subsequent steps such as sentiment labeling or preprocessing.

```python
# --- Lưu vào file Excel ---
if not all_comments_data:
    print("Không có bình luận tiếng Việt nào được tìm thấy để lưu.")
    return
wb = Workbook()
ws = wb.active
ws.title = "YouTube Comments (Vietnamese)"
ws.append(["Index", "Comment", "Type"])
for comment_data in all_comments_data:
    ws.append(comment_data)

for col in ws.columns:
    max_length = 0
    column = col[0].column_letter
    for cell in col:
        try:
            if len(str(cell.value)) > max_length:
                max_length = len(cell.value)
        except:
            pass
    adjusted_width = (max_length + 2)
    ws.column_dimensions[column].width = adjusted_width
wb.save(output_file)
print(f"\n✅ Thành công! Đã lưu {len(all_comments_data)} bình luận tiếng Việt vào file '{output_file}'")
```

Figure 8: Storing structured comments in Excel format

### a.3. Emotion Labels

The textual data collected from YouTube initially exists in raw format and has not yet been labeled. To build a sentiment classification model, the next step involves manually annotating this dataset. This labeling process is conducted by human annotators based on a predefined set of guidelines and emotion categories to ensure consistency and reliability across the dataset.

The selection of emotion labels is based on the classic psychological model of six basic emotions proposed by ***Paul Ekman***, a widely recognized and applied framework in the fields of artificial intelligence and sentiment analysis. According to this model, the dataset is categorized into the following six primary emotions:

- **Joy:** Comments expressing positivity, satisfaction, excitement, affection, or admiration.
    - *Examples: "Video hay quá, xem giải trí thật sự!", "Idol của mình đỉnh quá!"*
- **Sadness:** Comments reflecting disappointment, empathy, sorrow, or regret about an event or a character.
    - *Examples: "Nghe xong buồn quá.", "Thật tiếc cho nhân vật chính."*
- **Anger:** Comments showing strong dissatisfaction, criticism, outrage, or aggression.
    - *Examples: Làm ăn thế này không chấp nhận được!", "Thật là bực mình khi xem."*
- **Fear:** Comments conveying anxiety, worry, insecurity, or fear about a situation being mentioned.
    - *Examples: "Xem cảnh này thấy ghê quá.", "Tôi lo là sẽ có chuyện không hay xảy ra."*
- **Disgust:** Comments expressing aversion, disdain, contempt, or a strong negative reaction to an act, opinion, or object perceived as offensive or immoral.
    - *Examples: "Hành động đó thật kinh tởm.", "Không thể tin được có người lại suy nghĩ như vậy, thật đáng khinh."*
- **Surprise:** Comments showing astonishment or disbelief (either positive or negative) about an unexpected event or piece of information.
    - *Examples: "Ồ, không ngờ kết quả lại như vậy!", "Cái gì? Thật không thể tin nổi!"*

Once fully annotated, the dataset becomes a **“ground truth”** resource for training and evaluating the performance of sentiment analysis models in subsequent stages of our project.

**a.4. Music Dataset**

To generate music, we utilized the Lakh MIDI Dataset—a large-scale collection of 176,581 unique MIDI files, of which 45,129 are matched and aligned with tracks from the Million Song Dataset (MSD). The Lakh MIDI dataset was designed to support large-scale research in music information retrieval (MIR), enabling both symbolic analysis (via MIDI) and audio content-based studies (via MIDI-aligned audio).

The Clean MIDI Subset includes a smaller collection of MIDI files with more reliable filenames that indicate the artist and title (though minor inaccuracies may exist). This cleaned version has been used in multiple academic studies and is particularly suitable for projects that require well-organized symbolic music data.

From music generated from the Lukh dataset, we fine-tuned that model on the VGMIDI, a dataset of piano arrangements of video game soundtracks created by Ferreira et al. It contains 200 MIDI pieces labelled according to emotion. Particularly, each MIDI file was assigned a value from -1 to 1 in Valence and Arousal.

## 2.2.2. EDA (Exploratory Data Analysis)

### b.1. Distribution of Emotion Labels

Figure 9 illustrates the distribution of sentence counts across emotion labels. Among all classes, **anger** appears most frequently, followed by **happiness, sadness, and fear**. Although **disgust** and **surprise** have relatively fewer samples, the overall distribution remains fairly balanced. This suggests the dataset is well-suited for supervised learning without requiring major class imbalance corrections.

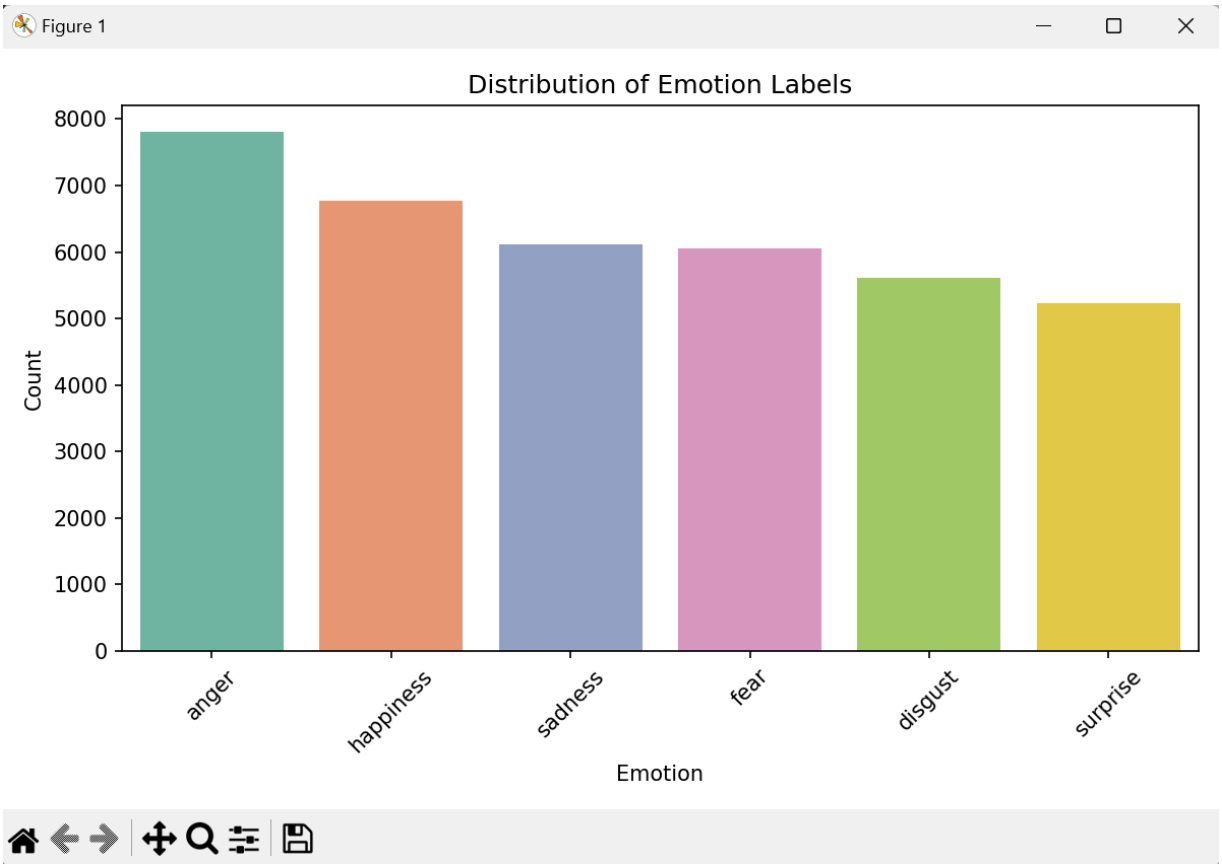



Figure 9: Distribution of sentences by emotion label

### b.2. Sentence Length Distribution

Figure 10 shows the distribution of sentence length in terms of word count. Most sentences are short, with the majority falling within the 5 to 20 words range. The distribution exhibits a strong right skew, which is typical in user-generated content. A few extreme outliers exceed 100 words, indicating the need for careful preprocessing such as truncation or padding to handle exceptionally long inputs.

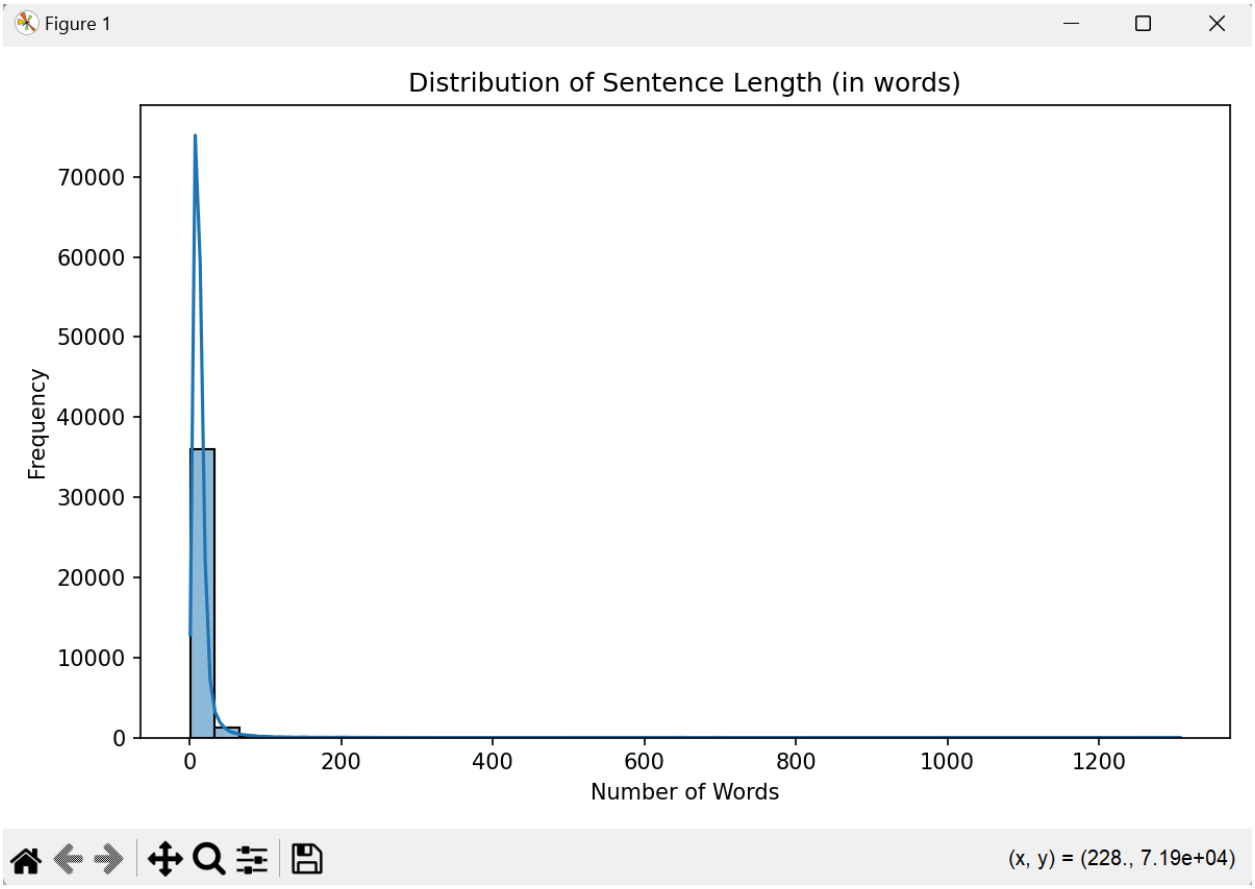



Figure 10 : Distribution of sentence length (in words)

### b.3. WordCloud by Emotion Label

To visualize the most frequently used words per emotion class, we generated a series of WordClouds. In each plot, word size correlates with its frequency.

- Anger: Common words like “chửi”, “bực”, “tức”, “ghét”, “không”, reflecting intense negative sentiment.


Figure 11: Word Cloud for Anger emotion

- Happiness: Includes positive expressions such as “vui”, “thích”, “yêu”, “cảm ơn”, “hay quá”.


Figure 12: Word Cloud for Happiness emotion

- Fear: Associated with terms like “kinh dị”, “giật mình”, “bóng tối”, “sợ”, “ám ảnh”.

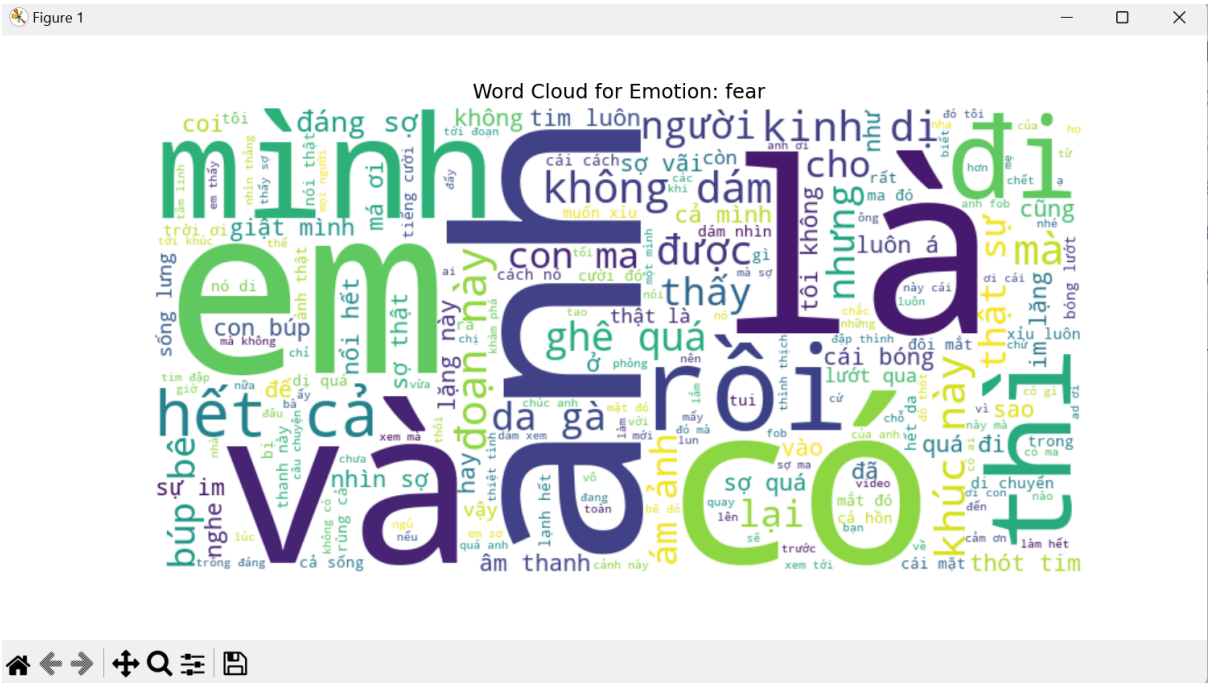



Figure 13: Word Cloud for Fear emotion

- Disgust: Common words include “kinh”, “rợn người”, “khó chịu”, “buồn nôn”.

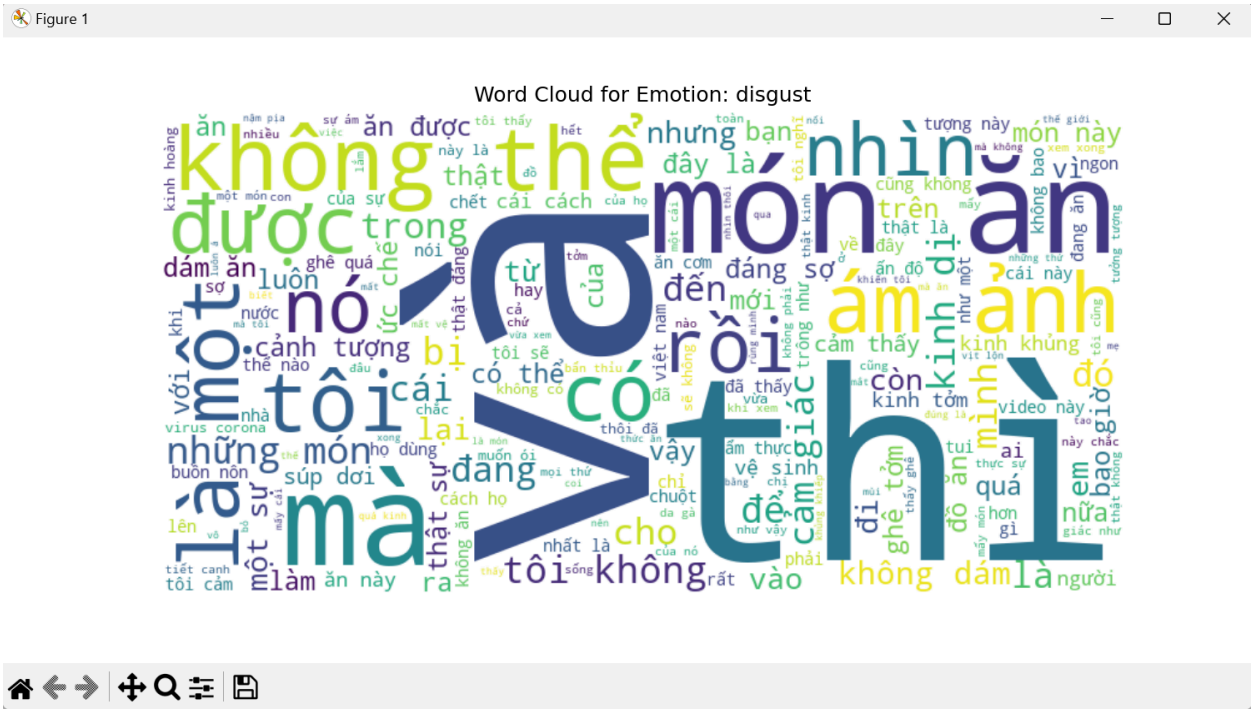



Figure 14: Word Cloud for Disgust emotion

- Sadness: Frequently contains words like “mất”, “buồn”, “nước mắt”, “tiếc thương”.


Figure 15: Word Cloud for Sadness emotion

- Surprise: Characterized by phrases such as “không thể tin”, “há hốc”, “quá bất ngờ”.

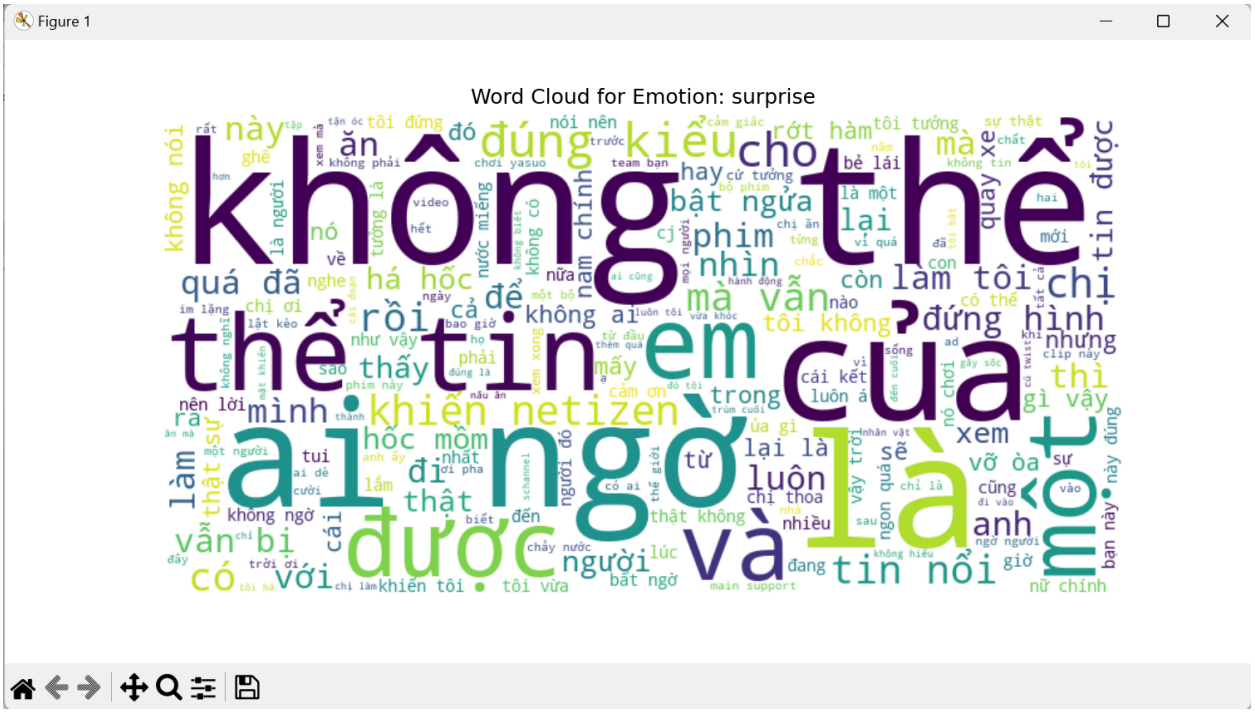



Figure 16: Word Cloud for Surprise emotion

These visualizations not only reveal the semantic nature of each emotion class but also provide valuable insights for feature engineering in deep learning-based NLP models.

### b.4. Sentence Length by Emotion Label (Boxplot)

Figure 17 presents a boxplot of sentence lengths grouped by emotion. Notably, **happiness** and **fear** tend to have longer sentences on average, while **surprise** and **anger** feature shorter responses. The presence of extreme outliers in all groups highlights the importance of normalization steps during data preprocessing, especially for models sensitive to input length.

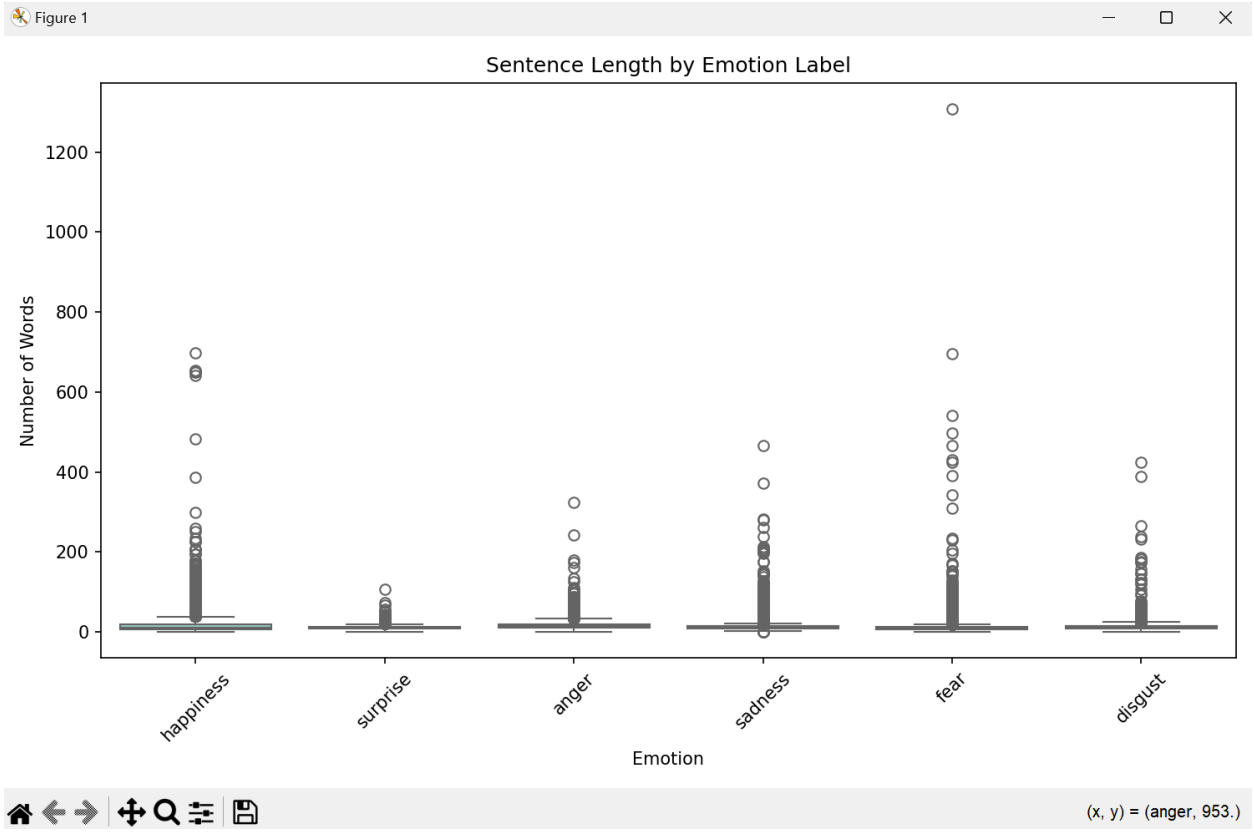



Figure 17. Sentence length by emotion label (boxplot)

### b.5. Average Sentence Length by Emotion

Figure 18 compares the average sentence length across all emotion labels. The happiness and anger classes exhibit the longest sentences (averaging over 16 words), suggesting greater elaboration or expressive tendency. In contrast, the surprise category has the shortest average (around 11 words), consistent with the concise and impulsive nature of that emotion.

These results are useful when choosing the appropriate input length for downstream models, especially sequence-based architectures like RNN, LSTM, or Transformer-based models (for our project is PhoBert).

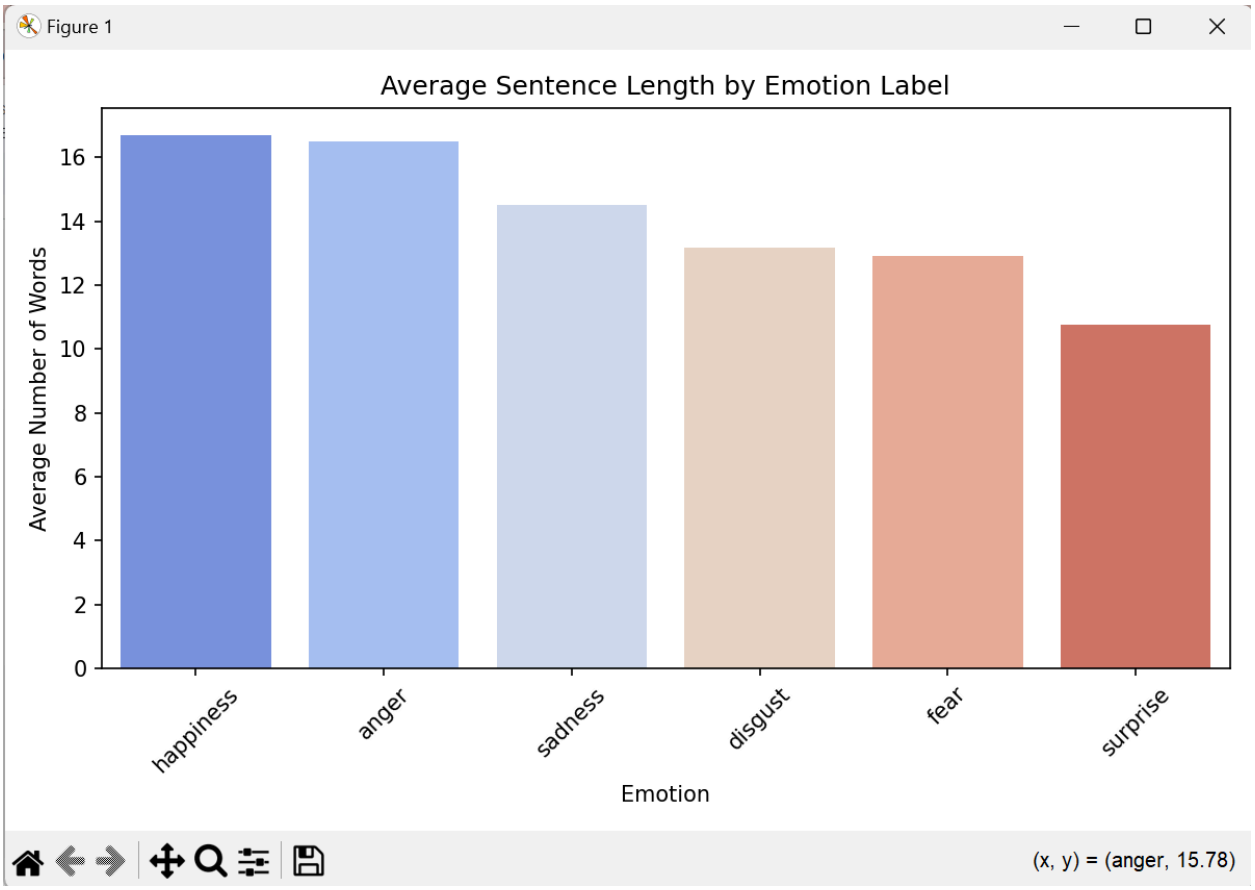



Figure 18: Average sentence length by emotion label

### b.6. Most Frequent Words in All Sentences

Figure 19 illustrates **the Top 20 most frequent words** appearing across all sentences in the dataset. These words include highly common tokens such as *"không" (no), "là" (is), "này" (this), "một" (one), "có" (have),* among others. Most of them are function words that are grammatically necessary in Vietnamese but carry little emotional or semantic weight.

This finding suggests that many of the highest-frequency terms are not informative for emotion classification and should be considered stopwords. During preprocessing, it may be beneficial to remove or downweight these common tokens in order to highlight emotionally salient vocabulary. Doing so can improve feature selection and model performance, particularly in tasks like sentiment analysis or emotion detection.

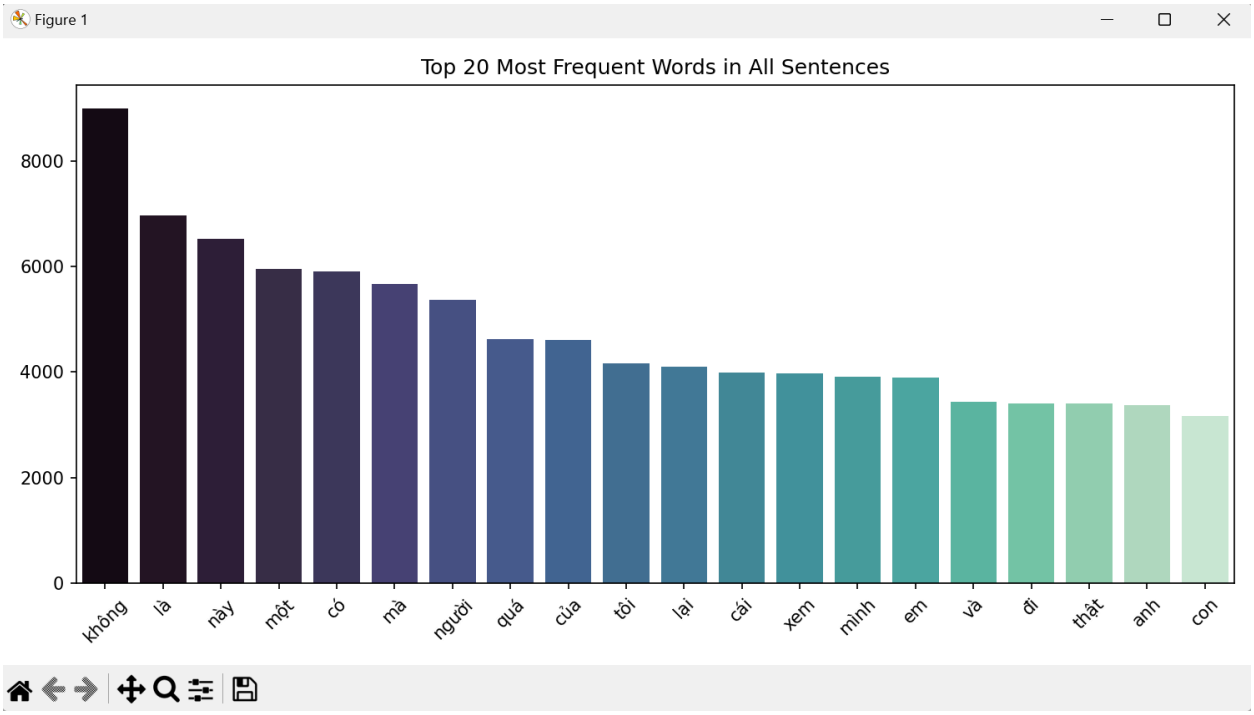



Figure 19. Top 20 most frequent words in the dataset

### b.7. Music EDA

**Dataset Overview:**

The dataset includes 240 unique MIDI file entries, each containing 8 attributes. There are no missing values, indicating that the dataset is clean and well-structured. Final Fantasy is the most frequently appearing series (40 tracks) followed closely by Legend of Zelda (31 tracks) and Banjo-Kazooie (20 tracks).

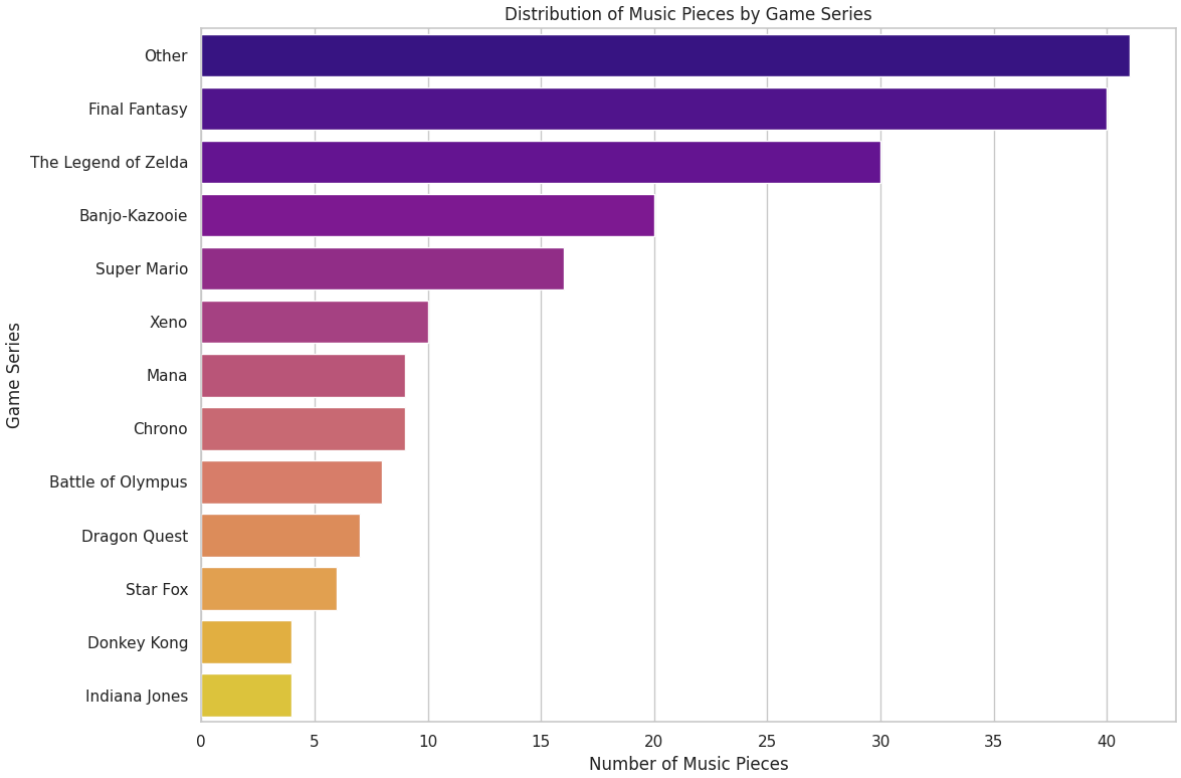



Figure 20: Dataset Overview

**Overall Emotional Tone:**

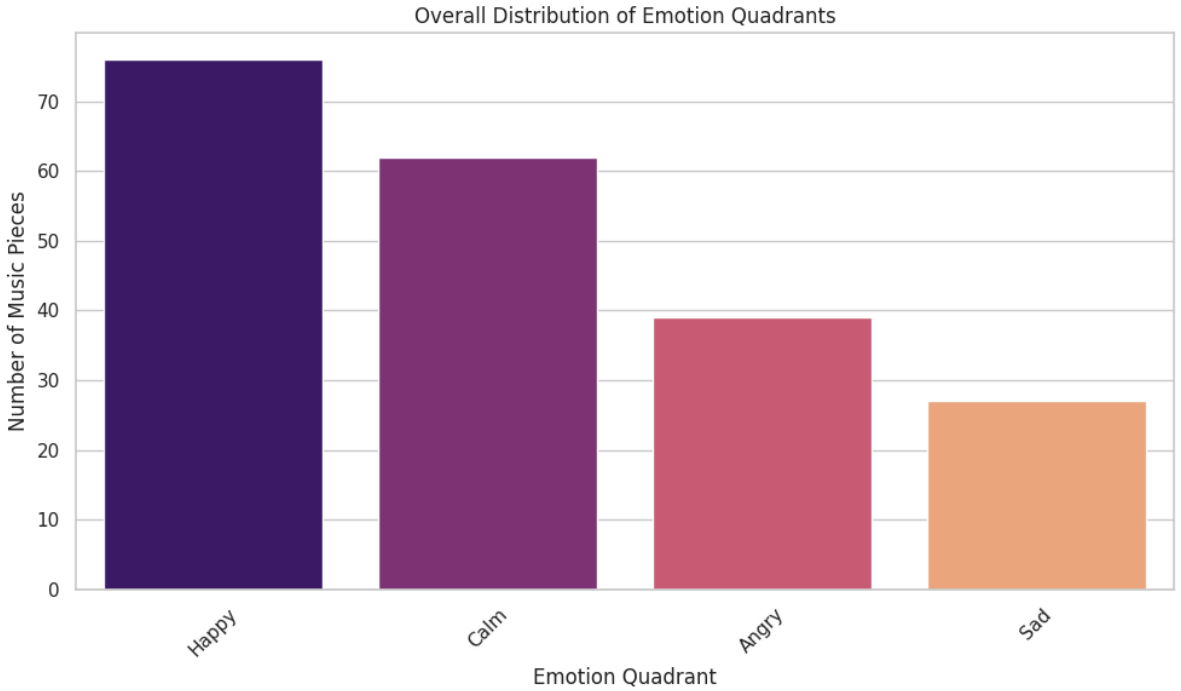



Figure 21: Overall Emotional Tone

The dataset leans heavily toward positive emotions. ‘Happy’ (high valence, high arousal) is the most common emotional quadrant, with 82 tracks. ‘Calm’ (high valence, high arousal) follows closely with 61 tracks. Negative emotional tones are less frequent, with ‘Angry’ at 39 tracks and ‘Sad’ being the least common at 26 tracks.

**Emotional Profiles by Game Series:**

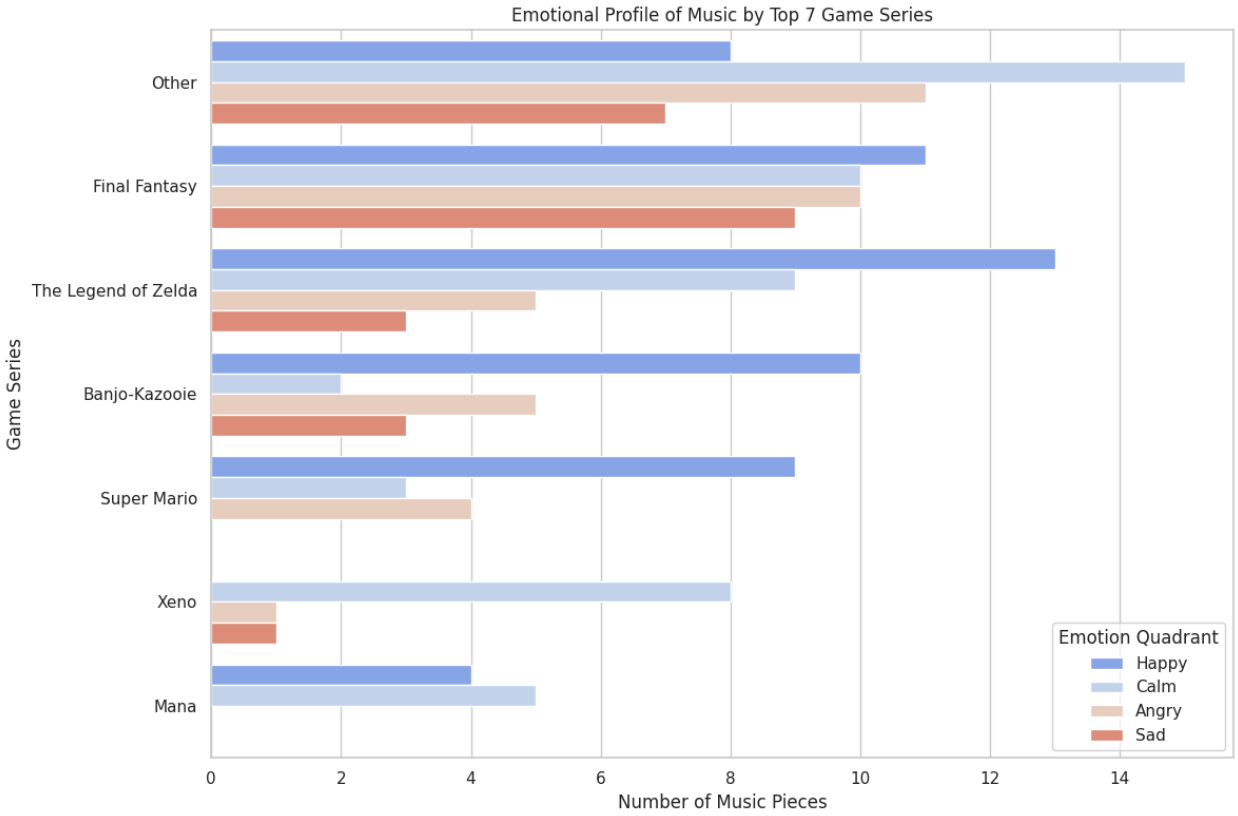



Figure 22: Emotional Profiles by Game Series

Final Fantasy shows a wide emotional range with tracks spread across all quadrants, reflecting its varied gameplay, it also dominates the ‘Tense/Angry’ category with battle themes and dramatic pieces, while also contributing many ‘Calm/Content’ tracks.

Super Mario is overwhelmingly positive, with most tracks falling into the ‘Happy/Excited’ category - consistent with the series’ upbeat and cheerful tone.

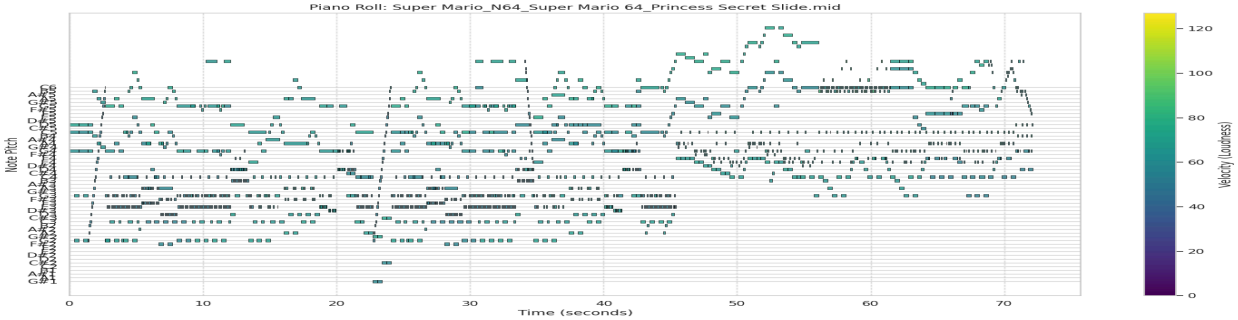



Figure 23: Music representation of the Mario Bros theme song

### 2.2.3. Data Preprocessing

In building reliable NLP tools, especially those dealing with rich linguistic words like Vietnamese, meticulous preprocessing of data is a cornerstone. It is akin to tidying up a giant messy library prior to any attempt to place its contents into effective cataloguing. The inherent multifaceted nature of natural language, coupled with the informal character of communications and script peculiarities, necessitates a wide-ranging regime of cleaning and normalization procedures to secure textual data quality and consistency prior to model training.

The first step is to define a list of Vietnamese stopwords, i.e., function words such as prepositions, conjunctions, pronouns, and interjections, which are usually semantically irrelevant and typically discarded during the course of natural language processing. Besides the stopword list, a "teencode" dictionary, i.e., Vietnamese teen lingo or colloquial net slang, is also defined. This dictionary helps to normalize teencode words like "k" or "iu" into formal words ("không" and "yêu", respectively). The pattern of a regular expression is then compiled to search for any of these teencode tokens efficiently during preprocessing. A regular expression is built to find these slang words as single words so that substitution can be context-sensitive and efficient. For instance, "k" or "ko" is translated to "không", and "tks" is translated to "cảm ơn".

```
# --- VIETNAMESE STOPWORDS ---
VIETNAMESE_STOP_WORDS = set([
    "và", "hoặc", "là", "có", "của", "trong", "theo", "này", "đây", "với", "cho", "mà", "được",
    "cùng", "bởi", "từ", "nếu", "cũng", "sẽ", "khi", "không", "để", "đi", "vì", "mới", "cả",
    "hơn", "nhiều", "ít", "thì", "như", "các", "vào", "bằng", "ra", "lên", "xuống", "qua", "lại",
    "anh", "em", "chị", "bạn", "tôi", "mình", "nó", "họ", "chúng ta", "chúng tôi", "chúng nó",
    "ai", "gì", "đâu", "nào", "sao", "bao nhiêu", "lúc nào", "tại sao", "ở", "tại", "trên",
    "dưới", "trước", "sau", "ấy", "những", "một", "hai", "ba", "vài", "rằng", "ạ", "à", "ừ",
    "dạ", "vâng", "ơi", "nhỉ", "nhé", "nha", "đó", "đây", "kia", "ấy"
])

# --- TEENCODE DICTIONARY ---
TEENCODE_MAP = {
    "k": "không", "ko": "không", "khum": "không", "hok": "không", "hem": "không", "hong": "không",
    "j": "gì", "g": "gì", "z": "gì", "zậy": "vậy", "zay": "vậy", "v": "vậy", "zô": "vào", "zo": "vào",
    "r": "rồi", "roi": "rồi", "wá": "quá", "wa": "quá", "iu": "yêu", "luv": "yêu",
    "thks": "cảm ơn", "tks": "cảm ơn", "thanks": "cảm ơn", "ty": "cảm ơn",
    "ok": "được", "oke": "được", "oki": "được", "okie": "được", "dc": "được", "đc": "được",
    "vl": "rất", "vkl": "rất", "vcl": "rất", "vch": "rất", "vs": "với", "mn": "mọi người",
    "bik": "biết", "bjt": "biết", "bit": "biết", "bb": "tạm biệt", "bye": "tạm biệt",
    "h": "giờ", "hjo": "giờ", "ng": "người", "nguoi": "người", "ntn": "như thế nào",
    "a": "anh", "e": "em", "ib": "nhắn tin", "inbox": "nhắn tin", "s": "sao",
    "đk": "được không", "dk": "được không", "t": "tôi", "b": "bạn", "m": "mày",
}

# Compile regex for efficiency
teencode_pattern = re.compile(r'\b(' + '|'.join(re.escape(key) for key in TEENCODE_MAP.keys()) + r')\b', re.IGNORECASE)
```

Figure 24: Define Vietnamese Stopwords and teencode

Next, to normalize text, we create a special function, which replaces all recognized teencode terms with their standard Vietnamese counterparts. This function looks for every instance of teencode in the input text using the previously built regex pattern, then replaces them with a mapping from the teencode we defined before. In addition to improving consistency and model performance during training or analysis, this normalizes informal or abbreviated language.

```python
def normalize_teencode(text: str) -> str:
    """Standardizes teencode words to their proper form."""
    return teencode_pattern.sub(lambda m: TEENCODE_MAP[m.group(0).lower()], text)
```

Figure 25: Teencode Normalization Function

In an effort to maintain the linguistic integrity of the dataset, one would add a utility function to check whether a given text contains Vietnamese diacritical marks. Vietnamese has a set of diacritics that are employed to distinguish between various tones and meanings and thus are quite pivotal when it comes to semantic interpretation. The function employs a regular expression to check for the presence of any such character. Scripts that lack these diacritics are noisy or colloquial and are removed from the dataset before further processing.

```python
def has_vietnamese_diacritics(text: str) -> bool:
    return re.search(r'[áàảãạăắằẳẵặâấầẩẫậđéèẻẽẹêếềểễệíìỉĩịóòỏõọôốồổỗộơớờởỡợúùủũụưứừửữựýỳỷỹỵ]', text, re.IGNORECASE) is not None
```

Figure 26: Checking diacritical marks Function

We build a function that encapsulates the whole text preprocessing process. Before doing the teencode normalization, it first changes the input text to lowercase. Then it removes HTML elements and URLs. Next, Unicode normalization is carried out in an effort to maintain character uniformity. The function then eliminates unnecessary characters while leaving Vietnamese letters, numbers, and basic punctuation intact. Additionally, it eliminates superfluous punctuation and whitespace from the sentence borders. Lastly, a Vietnamese tokenizer is used to tokenize the cleaned text, and the tokens are then reassembled into a coherent phrase. Standardized, noise-free text is produced by this pipeline for use in subsequent NLP applications.

```python
def preprocess_pipeline(text: str) -> str:
    if not isinstance(text, str):
        return ""

    # Convert to lowercase
    s = text.lower()
    # Normalize teencode
    s = normalize_teencode(s)
    # Remove HTML tags and URLs
    s = re.sub(r'<[^>]+>', ' ', s)
    s = re.sub(r'http\S+|www.\S+', ' ', s)
    # Normalize Unicode
    s = unicodedata.normalize("NFC", s)
    # Remove special characters, keeping letters, numbers, whitespace, and standard punctuation
    vietnamese_chars = 'áàảãạăắằẳẵặâấầẩẫậđéèẻẽẹêếềểễệíìỉĩịóòỏõọôốồổỗộơớờởỡợúùủũụưứừửữựýỳỷỹỵ'
    allowed_chars = r'[^a-z0-9' + vietnamese_chars + r'\s.,?!:;\'"]'
    s = re.sub(allowed_chars, ' ', s)
    # Remove extra whitespace
    s = re.sub(r'\s+', ' ', s).strip()
    # Remove any leading/trailing punctuation and whitespace
    punctuation_to_trim = r'.,?!:;\'"\s'
    s = re.sub(r'^[' + punctuation_to_trim + r']+|[' + punctuation_to_trim + r']+$', '', s)

    # Tokenization
    tokens = word_tokenize(s)
    s = " ".join(tokens) # Join tokens back into a string

    return s
```

Figure 27: Text Preprocessing Pipeline

After reading the input data, rows containing null or missing values in the specified text or label columns are deleted, along with any repeated sentences for avoiding redundancy. Furthermore, because they are essential to accurate linguistic analysis, only sentences with Vietnamese diacritical marks are kept, ensuring that the data set is grammatically accurate, clean, and prepared for further training.

```python
try:
    print(f"Reading Excel file from: {INPUT_EXCEL_FILE}")
    df = pd.read_excel(INPUT_EXCEL_FILE, engine='openpyxl')
    print(f"File read successfully. Initial rows: {len(df)}")


    df.dropna(subset=[TEXT_COLUMN, LABEL_COLUMN], inplace=True)
    df.drop_duplicates(subset=[TEXT_COLUMN], inplace=True, keep='first')
    df[TEXT_COLUMN] = df[TEXT_COLUMN].astype(str)
    initial_rows = len(df)
    df = df[df[TEXT_COLUMN].apply(has_vietnamese_diacritics)].copy()
    rows_removed = initial_rows - len(df)
    print(f"Data cleaned. Removed {rows_removed} sentences without diacritics. Remaining rows: {len(df)}")
```

Figure 28: Load and initial data cleaning

The pipeline function we defined before then processes each sentence in the dataset after the raw data has been cleaned. This process ensures consistent formatting and lowers noise by converting

each input text to its cleaned, normalized, and tokenized form. In order to use it in further analysis or model training phases, the processed text is subsequently recorded in a new column. This is essential in order to convert unstructured raw text into structured data for training process

```
print("\nApplying preprocessing pipeline...")
df['processed_text'] = df[TEXT_COLUMN].apply(preprocess_pipeline)
```

Figure 29: Processing data

After preprocessing the text, data cleanup is performed to avoid data inconsistency. Rows that produced empty strings during processing are removed. Emotion labels are normalized by converting various alternative forms and spellings to a standard set of categories. For instance, labels such as "buon ba" are changed to "buồn bã". For accurate categorization and assessment, this phase keeps every instance in the label column inside a consistent labeling framework.

```
df.replace('', np.nan, inplace=True)
df.dropna(subset=['processed_text'], inplace=True)
print(f"Rows after final cleaning: {len(df)}")

# 6. Standardize label column
label_map = {
    "buồn bã": "buồn bã", "buon ba": "buồn bã",
    "tức giận": "tức giận", "tuc gian": "tức giận",
    "vui vẻ": "vui vẻ", "vui ve": "vui vẻ",
    "sợ hãi": "sợ hãi", "so hai": "sợ hãi",
    "ngạc nhiên": "ngạc nhiên", "ngac nhien": "ngạc nhiên",
    "ghê tởm": "ghê tởm", "ghe tom": "ghê tởm",
}
df[LABEL_COLUMN] = df[LABEL_COLUMN].astype(str).str.strip().str.lower().replace(label_map)
```

Figure 30: Standardize label

Finally, we export only the cleaned text and the standardized emotion label. Before being exported into a new Excel file, the columns are renamed and rearranged from the original structure. This output file then will be used for further training.

```
    final_df = df[['processed_text', LABEL_COLUMN]]
    final_df.columns = [TEXT_COLUMN, LABEL_COLUMN]

    final_df.to_excel(
        OUTPUT_EXCEL_FILE,
        index=False
    )

    print("\n--- PREPROCESSING COMPLETE ---")
    print(f"Processed data saved to Excel file: {OUTPUT_EXCEL_FILE}")
    print(f"Final dataset contains {len(final_df)} rows.")

except FileNotFoundError:
    print(f"Error: Input file not found at {INPUT_EXCEL_FILE}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

Figure 31: Export data

**Music Data Preprocessing**

The goal of data preprocessing was to convert 200 MIDI files and their corresponding [Valence, Arousal] labels into a format suitable for a conditional LSTM model. Using the music21 library, we first parsed all MIDI files to build a comprehensive vocabulary of musical events. Individual notes were tokenized by their pitch, chords by their normal order, and critically, musical rests were encoded as a special 'rest' token to enable the model to learn silence.

This vocabulary was used to convert the MIDI streams into integer sequences. We then generated input-output pairs using a sliding window approach with a sequence length of 100. For conditional training, each input consisted of two parts: the 100-event musical sequence and its associated emotion vector, duplicated across all 100 time steps.

Input sequences were normalized, and target outputs were one-hot encoded. To resolve a significant CPU bottleneck and maximize GPU utilization, this entire processed dataset was saved to disk and loaded during training via an optimized tf.data pipeline, which included shuffling, batching, and prefetching.

## 2.3. Model Architecture (Emotion: PhoBERT, Music: Transformer)

### 2.3.1. Emotion Recognition

#### a.1. PhoBERT

##### a.1.1. Overview

We selected PhoBERT, specifically the VinAI version. Our reason for doing so lies in its state-of-the-art performance on a wide range of Vietnamese Natural Language Processing (NLP) tasks. Unlike multilingual models, PhoBERT is a monolingual Transformer pre-trained on a massive and diverse Vietnamese corpus (over 20GB of text). We felt that this intensive pre-training would endow our model with a rich, fine-grained knowledge of Vietnamese grammar, meaning, and idiomatic expression and therefore would be a super-strong baseline for our sentiment analysis task. It contains an underlying RoBERTa architecture, an optimized variant of BERT, that has superior pre-training methods that enhance its representational power.

##### a.1.2. Core Architecture

PhoBERT follows the common architecture of a "base" Transformer model. Its essence is a repetition stack of the same encoder layer, which is engineered to take in a sequence of token embeddings and produce contextually enhanced representations. The most important specifications of this architecture are as follows [12]:

- Number of Layers: Our model has 12 layers of stacked Transformer encoder. In every layer, our model processes the previous layer's output, allowing our model to build progressively more complex and higher-level representations of the input text.
- Hidden Size (Units): Hidden state size in all layers is 768 units. This means that each token of an input sequence is represented as a vector of 768 dimensions in each layer of the network.
- Attention Mechanism: Every encoder layer within our model includes a multi-head self-attention mechanism with 12 attention heads. It enables the model to assign different importance values to various words within the input sequence while encoding a word, thus allowing it to effectively capture long-range dependencies and contextual relationships.
- Activation Function: The feed-forward network of each encoder block uses the GELU (Gaussian Error Linear Unit) activation function. We noticed from existing literature that GELU, as a smooth non-linear activation function, has been shown to deliver performance gains over the more traditional ReLU in Transformer models.

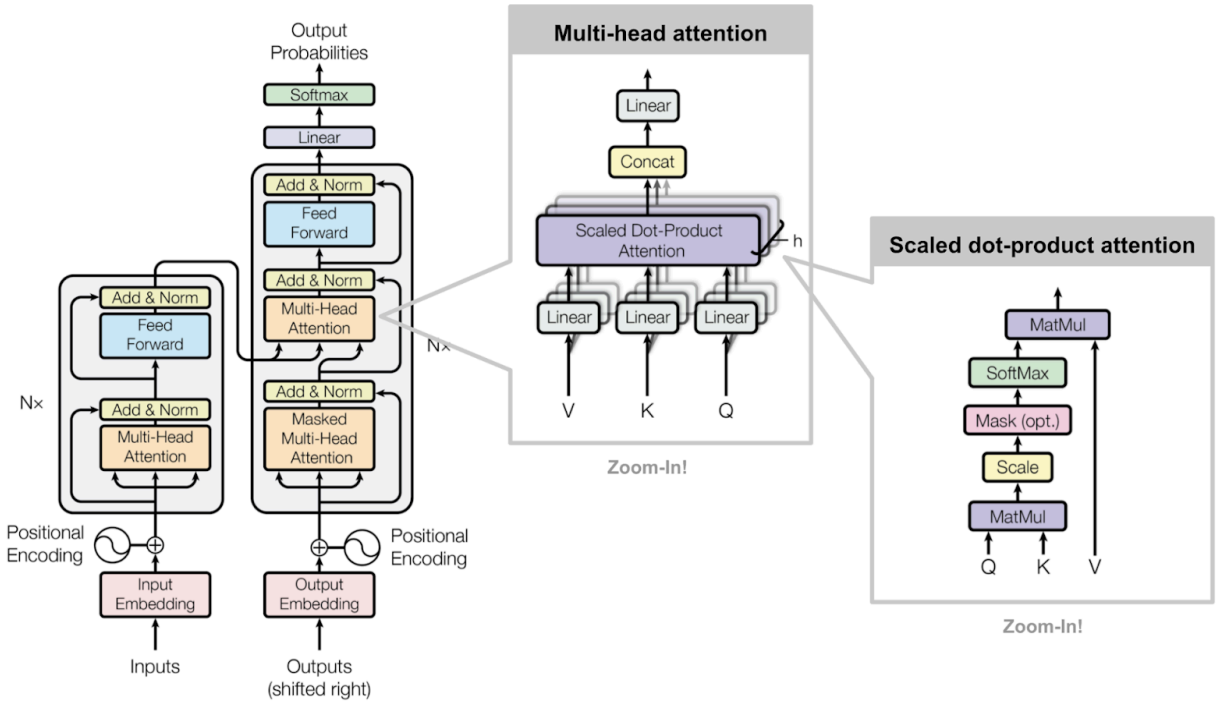



Figure 32: Model Architecture of Transformer in general

a.1.3. Custom Classification Head

The input to this classification head is a fixed-size single vector that represents the global meaning of the whole input sentence. We get this vector via a default pooling method to BERT-type models: we use the final hidden state of the special [CLS] token. The token is added to every input sequence at tokenization and is especially designed at pre-training time to capture sentence-level meaning. By extracting the 768-dimensional output vector of this token, we effectively have a dense representation of the input text.

Our classification head then passes this vector into two consecutive components:

- A Dropout Layer: The first component is a dropout layer, which we initialize with probability p=0.3. The layer is a crucial regularization technique that we incorporated to avoid overfitting. During each forward pass during training, this layer zeroes (sets to zero) 30% of the values of the input feature vector at random. This behavior prevents the model from developing complex co-adaptations between neurons, forcing it to learn a more generalized and robust set of features. By reducing the network sensitivity to the individual weights of any one neuron, dropout significantly enhances generalizability of the model from the training examples to new examples. This layer is disabled automatically at test time to give deterministic outputs for inference.
- A Linear Classification Layer: The second and final component is a straightforward, fully-connected neural network layer, which in PyTorch is represented by nn.Linear. It performs the final classification. It takes the 768-dimensional, regularized feature vector from the dropout layer and does a linear transformation, mapping it onto a final output

vector of dimensionality 6, the six sentiment categories in our dataset (buồn bã, ghê tởm, etc.). The output of this layer is a vector of raw, unnormalized scores known as "logits". Each logit is the model's calculated evidence for a given class. These logits themselves are not probabilities but are then input into the CrossEntropyLoss function, which does a Softmax transformation internally to compute the final classification loss.

#### a.1.4. Operation Mechanism

The process starts with a batch of numerically encoded text sequences being given to the model. Each sequence is accompanied by auxiliary information such that the model can differentiate informative content from non-informative padding tokens which have been added so that the data is of uniform length.

The model then analyzes these inputs. The input undergoes a detailed contextual examination via its multi-layer Transformer architecture. It converts the original, static token representations into a series of highly-contextualized feature vectors using its feed-forward networks and layered self-attention. At this stage, each vector in the sequence does not just represent a word, but the sense of the word based on its whole surrounding context.

In order to facilitate classification, a subsequent aggregation step is performed. For each sequence in the batch, our approach summarizes the information contained in the whole sequence of feature vectors into a single fixed-size summary vector. That vector is made to preserve the aggregate semantic content of the source sentence and serve as a holistic representation for the classification head.

This summary vector then goes through a regularization step that we used to improve the model's generalization. This active step only when training prevents overfitting by injecting some structured noise in the feature encoding. This forces the network to learn more stable and less inter-dependent features.

Finally, the regularized vector is passed through the terminal classification layer. This layer performs the essential task of mapping the high-dimensional feature representation to the ultimate 6-dimensional output space of our problem. The output is a list of unnormalized scores, with each score being the model's calculated evidence for one of the six classes of sentiment. This output vector is the end product of the forward pass, providing the input to which the loss function can apply measurement of prediction error and subsequently propel the model's learning process through backpropagation.

### a.2.Bi-LSTM

#### a.2.1.Overview

Apart from PhoBERT, we have used a Bidirectional Long Short-Term Memory (Bi-LSTM) network to classify emotions from Vietnamese textual input. A specialized Recurrent Neural Network (RNN) that can learn from sequential data is called a Bi-LSTM. In order to predict a corresponding emotional category (such as Happy, Sad, Angry, or Neutral), the model in this

project examines a string of Vietnamese words (a sentence or phrase). The parameters of the next music generation module are then determined by the anticipated emotion.

a.2.2.Rationale for Model Selection

The choice of a Bi-LSTM model was driven by its strengths in handling the nuances of natural language, which are particularly relevant for the complexities of the Vietnamese language:

- Contextual Knowledge in Vietnamese: Emotion is strongly influenced by context. Only past context is taken into account by a typical, unidirectional LSTM. A Bi-LSTM is able to capture a more comprehensive contextual picture by processing data both forward and backward. For Vietnamese, where word order and negating particles are important, this is essential. The backward pass, for instance, shows the model that the word "không" (not) modifies "vui" (happy) in the sentence "Tôi không vui" (I am not happy), which is necessary for a proper classification.
- Managing Long-Term Dependencies: Complex and lengthy sentences can be used to develop emotional sentiment. Because of the "vanishing gradient problem," simple RNNs have trouble recalling information from earlier in a sequence. This is specifically addressed by LSTMs, which enable the model to retain pertinent information across lengthy Vietnamese sentences through their gating mechanism (input, forget, and output gates).
- Adaptability to Sequential Data: Text is sequential by nature. For such data, RNN architectures - especially LSTMs - are the most advanced option available, outperforming models that treat words as separate features.

a.2.3.Working Principle

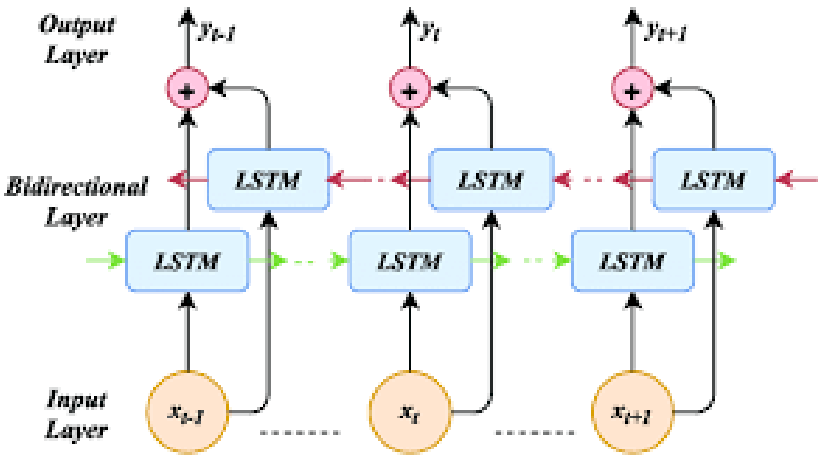
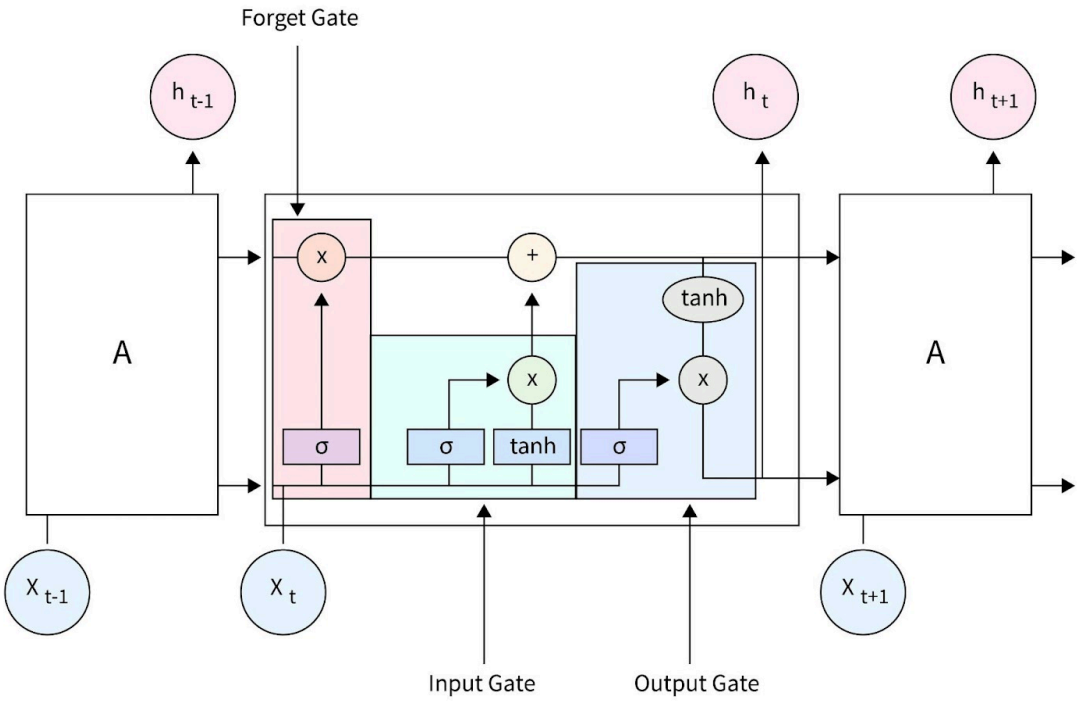



Figure 33:Bi-LSTM

The working mechanism of the Bi-LSTM model is illustrated in the figure. The procedure is specifically designed to process Vietnamese text and can be described according to the following steps, which correspond to sections shown in the figure:

- Input Layer:The procedure begins with the input of Vietnamese text, which must be preprocessed through, 1. word segmentation into tokens, and 2. creating a numerical vector for each word, through a Vietnamese embedding model that was created ahead of time for the word tokens. This series of vectors acts as our Input Layer. In the figure: each ***x_t (ex. x_{t-1}, x_t, x_{t+1})*** represents the vector for the t-th word in the sentence.
- Bidirectional Layer:After the sequence of ***x_t*** has been created, and for this specific procedure it was input into the two parallel LSTM together, this forms our Bidirectional Layer, as represented in the figure:
    - 1. Forward LSTM (Bottom Layer): The first LSTM layer processes the input sequence from left to right (left to right, ***x_{t-1}, x_{t}, x_{t+1}***). It collects the "past" context of each word.
    - 2. Backward LSTM (Top Layer): The second LSTM layer processes the input from right to left. This layer collects the "future" context of each word.
- Output Combination:At time step t, the output (hidden state) of the forward LSTM and the output of the backward LSTM are "combined." As shown by the ⊕ symbol in the diagram, we usually mean concatenated. This step combines the past and the future context to create a richer representation of each word. The result of combining the output of the forward LSTM and backward LSTM at time step t is vector y_t.
- Final classification (Output Layer): The output combination ***(y_{t-1}, y_t, y_{t+1}, ...)*** is passed to the final stage. For a sentence classification task, we usually either take the output of the last time step or the aggregation (average or max-pool) of all outputs. ***Y_t*** is thus passed through the dense layer with a Softmax output. The output layer is the layer where the probability is calculated for each emotion class (e.g., Happy, Sad, Angry) for the entire input sentence and chooses the predicted emotion from the entire input sentence.

## 2.3.2.Music Generation




LSTM Architecture

The Long Short-Term Memory (LSTM) cell is an advanced Recurrent Neural Network (RNN) architecture designed to learn long-term dependencies. Its core feature is the cell state, a horizontal line acting as a "conveyor belt" that carries information through the sequence, allowing it to be preserved or modified over time. The flow of information is meticulously controlled by three "gates": the Forget Gate, Input Gate, and Output Gate. Each gate uses a sigmoid activation function to output values between 0 and 1, effectively acting as a filter to decide how much information should pass through.

At each time step, the Forget Gate first determines which information from the previous cell state is no longer relevant and should be discarded. Concurrently, the Input Gate decides which new information from the current input is important enough to be added to the cell state. After the cell state is updated through these forgetting and adding operations, the Output Gate produces a filtered version of this new state. This filtered version becomes the hidden state (h_t), which

serves as both the output for the current time step and an input for the next, ensuring that only relevant information is passed on.

To make the output more interesting, we use sampling techniques:

- **Temperature:** A higher temperature increases randomness, encouraging the model to pick less obvious notes, leading to more "creative" or "surprising" results. A low temperature makes it more conservative and predictable.
- **Top-k / Top-p Sampling:** Instead of just picking the single best note, the model considers a small set of the most likely next notes (e.g., the top 5) and chooses randomly from that pool. This maintains quality while introducing variation.

## 2.4. Evaluation Metrics

### 2.4.1. Metrics for Emotion Recognition: Accuracy, Precision, Recall, F1-score, Confusion Matrix.

We use several metrics to evaluate performance: Accuracy, Precision, Recall, F1-score and Confusion Matrix. Each metric provides a different perspective on our model performance.

First metric is Accuracy. Accuracy is The total percentage of accurate predictions the model makes. Formally, accuracy is defined as follows for a classifier with N total predictions (if $N_{correct}$ is the number of words whose emotion was correctly predicted in a dataset of size N):

$$Accuracy = \frac{N_{correct}}{N}$$

i.e. the number of correct predictions divided by the total number of predictions.

High accuracy (such as 90%) means that 90% of the time, the model correctly recognizes the emotion in Vietnamese input text. For instance, if most inputs really are "vui vẻ" (happy) and the model is correct on most of them, together with being correct on other emotions too, then accuracy will be high. Accuracy by itself, however, does not tell us how well or poorly the different emotions are done. A model might be very accurate by being very good at classifying the majority class but terrible at the minority classes. For example, if "vui vẻ" appears most frequently in the set, a goofy model that labels all text as happy would label all "vui vẻ" texts accurately (perhaps boosting accuracy wildly) but mislabel all "sợ hãi", "buồn bã", etc. If "vui vẻ" made up 80% of the test set, this worthless model would have an 80% accuracy – seemingly good – but it completely fails to identify the other emotion. So while accuracy gives a quick overall assessment, it conceals differences in performance between classes

Second metric is Precision. Precision determines the accuracy of the classifier for a given class – it determines how many of the items predicted to be a given class actually are in the given class. Precision for a given emotion class C is defined as:

$$Precision_c = \frac{TP_C}{TP_C + FP_C}$$

where $TP_C$ (true positives) are the texts which truly have emotion c and were predicted correctly as c, and $FP_C$ (false positives) are the texts predicted to be c but really being another emotion. That is, $Precision_c$ is the number of predicted-c texts which are really c. This per-class accuracy is in precisely the same format as in binary accuracy, except we compute it for every class sequentially considering class "positive" and all others "negative".

In the Vietnamese text example with a focus on emotions, precision indicates tendencies to misclassify. Let's suppose that our model has a Precision of label “Sadness” value of 0.80. That is, the model predicts that a text is sad 80% of the time, and indeed the text was sad; the model predicts that a text is sad 20% of the time that it actually is not (those might be texts that were really “tức giận” or “sợ hãi” but were classified incorrectly on the sad side). Low precision on “ngạc nhiên” (surprise) on, say, a value of 0.50 would indicate that fifty percent of the "surprise"-predicted texts were not surprise – perhaps the model predicts other sentiments (fear or anger, say) as surprise a great deal. If we look at precision per class, we see which sentiments the model over-predicts (low precision tends to indicate that the model is too liberal with that classification, getting too many false positives). Let's suppose, say, that “vui vẻ” (happy) has high precision. The model is very rarely predicatively wrong about non-happy texts labeling those happy (when the model does classify non-happy texts happy, it's mostly correct). If instead “ghê tởm” (disgust) has low precision, then the model might be too liberal to classify things "disgust" when the text is really some other sentiment so that confusion occurs with other sentiments in Vietnamese text (perhaps confusing expressions of disgust with expressions of anger).

Third metric is Recall. Recall measures the completeness of the classifier for a particular class – it quantifies how many of the items of a given true class the model successfully captures. For a specific emotion class c, Recall is defined as:

$$Recall_c = \frac{TP_C}{TP_C + FN_C}$$

where $FN_C$ is the number of texts with underlying emotion c but predicted some other emotion. Recall is thus the number of texts that actually belong to class c and were also predicted to belong to class c. Alternatively, in the confusion table, $Recall_c$ is the c-th diagonal element divided by the c-th row sum (because the row sum is all occurrences of class c really happening). In our context, recall tells us about false negatives. Let's use "ghê tởm" (disgust) as an example: if $Recall_{disgust}$ is low, that means many texts that truly express disgust are labeled as something else (perhaps anger or fear), which means the model is bad at recognizing disgust when it exists. High recall for "tức giận" (anger), e.g., 0.95, would imply the model accurately captures 95% of all angry texts – it does not fail to capture an angry expression often. But high recall alone does not guarantee that those predictions are correct (they might include false positives). For instance,

the model can classify a wide range of negatively toned texts as "anger" so that it will get all the true cases of anger, but in doing so it might also get some of the “sad” or “disgust" texts wrongly as anger (thus lowering precision for anger). In general, recall is the metric to check if the model is sensitive enough to each emotion. In emotion detection applications (e.g., analyzing customer feedback or social media for emotions), high recall for, say, “fear” means the system is capable of finding most of the messages where fear is being expressed, which could be critical in safety or mental health applications.

Fourth metric is F1-Score. F1-Score is the harmonic mean of precision and recall. For a given class c:

$$F1_c = \frac{2Precision_c * Recall_c}{Precision_c + Recall_c}$$

Or can be expressed in terms of confusion matrix components:

$$F1_c = \frac{2TP_c}{2TP_c + FP_c + FN_c}$$

In Vietnamese emotion recognition, F1-score is typically the one measure of preference for model-to-model comparison because it takes precision and recall performance per emotion and rolls them up into one number. Good F1 for every emotion class suggests the model is catching most occurrences of that emotion (good recall) and not incorrectly labeling many others as that emotion (good precision). For example, an $F1_{fear}$ of 0.70 suggests the model has a good precision-recall trade-off on fear (perhaps precision = 0.75, recall = 0.65 or vice versa). If $F1_{surprise}$ is significantly lower, e.g., 0.40, it suggests a problem with surprise detection – perhaps the model isn't labeling many instances of surprise (bad recall) or labelling many other emotions as surprise (bad precision), or both. Macro F1 over the six emotions might be reported as, e.g., 0.68 (68%). Macro F1 gives a rough sense of overall model performance across all emotions; practitioners find it convenient to use to compare models or track improvements, since it will only be high if the model tends to do well on each class.

The final metric we used is Confusion Matrix. A confusion matrix is not a single number but a table that reports on the classifier's performance by showing how often each class has been predicted and what the actual class really is.

In a K-class multi-class problem (in our example, K=6 emotions), the confusion matrix is a K by K matrix. Each row is for the true (actual) class and each column for the predicted class (or vice versa, depending on convention).

The value at cell (i, j) of this matrix is the number of texts that actually have emotion class i but were predicted as class j. Diagonal entries $M_i$(predicted class = actual class) then tally the number of correct predictions per class. Off-diagonal entries indicate how many there are where an error was committed – e.g., $M_{sadness,\,anger}$ would tally the number of sad texts which were incorrectly tagged as angry. The name "confusion matrix" is a prompt that helps us to look at which classes the classifier tends to confuse. For our example of six emotions, the confusion matrix would be a 6×6 matrix whose rows could be headed (true) and columns headed

(predicted) by the same list. Each cell counts one type of outcome (e.g., the number of "fear" texts that are going to be "surprise" will be located in the row *fear*, column *surprise*). Adding up each row is the total number of texts for the actual emotion; adding up each column is the total number that are predicted to be that emotion. The confusion matrix is where calculations like precision, recall, and accuracy are obtained – it presents a global view of classifier performance

### 2.4.2. Metrics for Music Generation:

Evaluating LSTM-based music generation requires both **objective** and **subjective** metrics.

- Objective metrics assess the model's predictive accuracy and structural quality with Cross-Entropy Loss. Additional analysis using tools like music21 can evaluate features like pitch range, note diversity, tonality, and note density.

- Subjective evaluation is essential, as low statistical error doesn't guarantee musical quality. This includes listening tests, Musical Turing Tests, and emotional accuracy checks. For conditional models, it's crucial that generated music effectively conveys the intended emotion.

## 2.5. Emotion Mapping

Emotion is mapped using Russell's circumplex model, which defines feelings along two primary dimensions: Valence and Arousal. Valence represents the emotional value of a feeling, essentially measuring how pleasant or unpleasant it is. This corresponds to the horizontal axis on the chart, where emotions like "Happy" and "Delighted" have a positive valence, while emotions like "Sad" and "Miserable" have a negative valence. The second dimension, Arousal, measures the level of physical or mental activation. This is represented by the vertical axis, with high-arousal emotions like "Excited" and "Alarmed" at the top, and low-arousal emotions like "Sleepy" and "Calm" at the bottom.

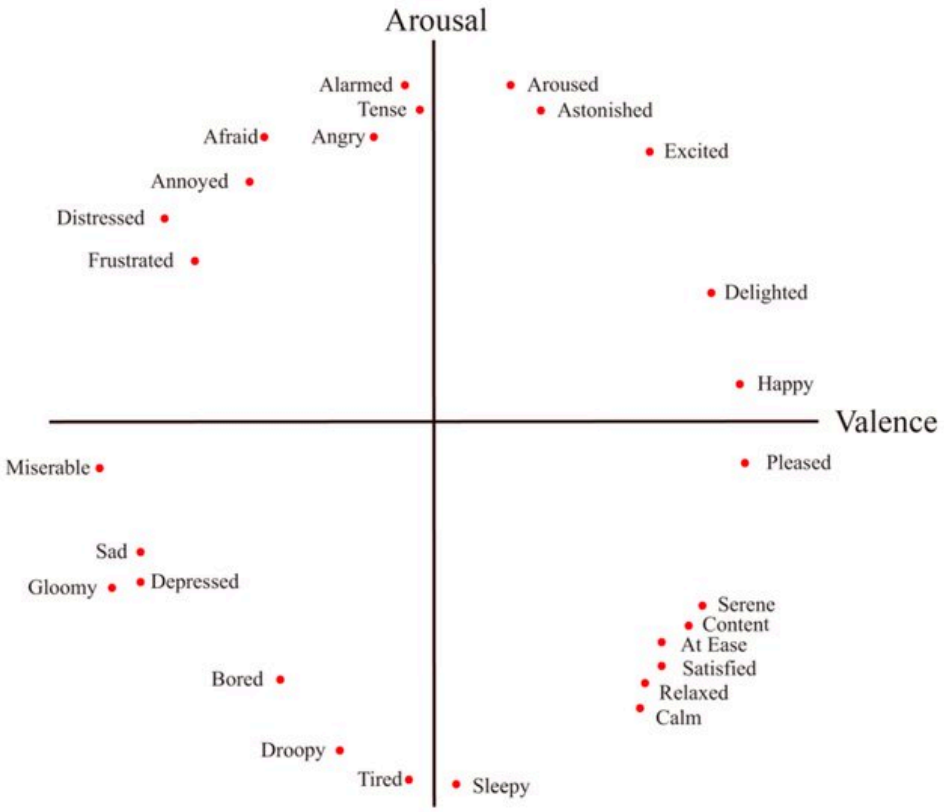



Figure 34: Russell's circumplex model

This two-dimensional system allows any emotion to be plotted as a point on the graph, providing a specific coordinate of valence and arousal. For example, feelings in the top-right quadrant, such as "Excited," are both pleasant (positive valence) and high-energy (high arousal). Conversely, feelings in the bottom-left quadrant, like "Gloomy" or "Bored," are unpleasant (negative valence) and low-energy (low arousal). The overall valence and arousal are calculated as below:

$$A = \sum_{i=0}^{5} p_i y_{Ai} \quad \text{and} \quad V = \sum_{i=0}^{5} p_i y_{Vi}$$

By quantifying emotions in this way, the model creates a comprehensive map that visually organizes the complex spectrum of human feelings based on their core components of pleasantness and activation level.

# III. Experimental results

## 3.1. Dataset

- Emotion Dataset

The final annotated dataset contains a total of 37,601 comments, distributed across the six emotion categories as follows:

- **Anger**: 7814 comments
- **Happiness**: 6769 comments
- **Sadness**: 6122 comments
- **Fear:** 6055 comments
- **Disgust**: 5614 comments
- **Surprise**: 5227 comments

This relatively balanced distribution across labels ensures the quality and fairness of model training and evaluation, reducing the risk of bias toward any particular emotion class.

## 3.2. Model Evaluation

### 3.2.1.Emotion Recognition

a.1. Experimental Environment:

All training and testing were conducted in the Kaggle cloud notebook setting, which supports consistent machine learning research. To accelerate the computationally intensive training process, the training process was executed on an NVIDIA P100 GPU with 16GB of VRAM (GPU Accelerator of Kaggle). The core software stack was set up on top of the PyTorch deep learning framework. The deployment heavily leveraged the Transformers library by Hugging Face to access the pre-trained model architecture and its associated tokenizer, and the scikit-learn library for the calculation of performance metrics.

a.2. Hyperparameter Configuration:

Hyperparameter is a critical factor that impacts model convergence and final performance. Each parameter was chosen to optimize the balance between model stability, training effectiveness, and generalization performance. Following standard procedure in fine-tuning Transformer models and after some initial setting exploration with a sequence of small-scale experiments, the following key hyperparameters were set:

- Max Sequence Length: Each input sequence was padded to a uniform length of 128 tokens. This was chosen as a balance between maintaining sufficient contextual information to be accurately classified and handling the computational expense (in terms of memory and processing time) of longer sequences.

```
train_encodings = tokenizer(train_texts, truncation=True, padding='max_length', max_length=128)
val_encodings = tokenizer(val_texts, truncation=True, padding='max_length', max_length=128)
```

Figure 35: Max Sequence Length

- Epoch: An epoch is one complete pass through the whole training data set. Training for at most 11 epochs was utilized for the model. This limit was selected to be high enough so that the model could learn the pattern in the data without risking too high a chance of overfitting—a situation where the model memorizes the training data but loses its ability to generalize to new instances. Above all, the model from the final epoch wasn't selected by default. Instead, there was an implicit early stopping strategy with the aid of checkpointing. After each epoch, the model's accuracy on the validation set was calculated. The model's state (i.e., its learned parameters) was only stored if its validation accuracy was better than the highest previously seen accuracy. This ensures that the ultimately selected model is the best generalization-performing model.

```
for epoch in range(1, 12):
    train_loss = train_one_epoch(model, train_loader, optimizer, scheduler, loss_fn, device, epoch)
```

Figure 36: Number of epochs used for training

- Batch Size: The batch size, which is the quantity of training examples driven into the network prior to an update of the parameters, was set to 64. This was a deliberate balance between accuracy of gradient estimation and available computational resources. The best batch size was 16 for the GPU Accelerator, preempting overflow memory error but just enough to provide a stable gradient estimation for reliable model convergence. While smaller batch sizes can sometimes assist with generalization by introducing noise, the value chosen provided a more stable and efficient training trajectory for this problem.

```
train_dataset = PhoBERTDataset(train_encodings, train_labels)
val_dataset = PhoBERTDataset(val_encodings, val_labels)
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16)
```

Figure 37: Batch Size

- Dropout rate: We choose the rate as 0.3. During training, the classification layer randomly removes 30% of its activations at each update step. It is an effective technique that keeps neurons from becoming overfitting and forces the network to learn more robust patterns.

```python
class PhoBERTClassifier(nn.Module):
    def __init__(self, num_labels):
        super().__init__()
        self.phobert = AutoModel.from_pretrained(model_name, config=config)
        self.dropout = nn.Dropout(0.3)
        self.classifier = nn.Linear(self.phobert.config.hidden_size, num_labels)

    def forward(self, input_ids, attention_mask):
        outputs = self.phobert(input_ids=input_ids, attention_mask=attention_mask)
        pooled = outputs.last_hidden_state[:, 0]
        return self.classifier(self.dropout(pooled))
```

Figure 38: Dropout Rate

- Gradient Clipping: In an effort to stabilize the numerical backpropagation, gradient clipping was applied. Gradient clipping is a significant protection against the "exploding gradients" phenomenon, in which gradients computed get too big and lead to unstable and diverging training dynamics. In our training cycle, after the gradients are computed through loss.backward(), their L2-norm is computed as well. If the sum of this combined norm exceeds a maximum of 1.0, the gradients are renormalized so that they have a precise norm of 1.0. What this does is effectively limit the size of any one parameter update step, promoting smoother and more stable convergence of the model.

```python
loss.backward()
torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
optimizer.step()
```

Figure 39: Gradient Clipping code

- Learning Rate: The AdamW (Weight Decay Adam) optimizer was utilized to train the model, with the initial learning rate set to 2e-5. A low learning rate is a standard and favored approach to fine-tuning large, pre-trained models as it enables smooth transition to the new task without the risk of damaging good, pre-existing knowledge in the layers of the model. In order to further enhance training stability and performance, a linear learning rate scheduler with warm-up was adopted. The learning rate rises from zero to the desired rate of 2e-5 over an initial warm-up phase. The learning rate thereafter decays linearly over the course of the remaining training process. This strategy prevents large, interruptive parameter updates at the start of training and prefers more subtle corrections as the model converges toward an optimum solution.

```python
optimizer = AdamW(model.parameters(), lr=2e-5)
```

Figure 40: Learning Rate

- Loss Function: This is a multi-class classification problem, and therefore, CrossEntropyLoss was employed as the loss function. It is standard for such cases since it is a LogSoftmax activation and Negative Log-Likelihood Loss in one numerically stable

class that computes the distance between the model's output probability distribution and one-hot encoded true labels.

```
loss_fn = nn.CrossEntropyLoss()
```

Figure 41: Loss Function

a.3. Training and Evaluation Process:

Model training and testing followed a systematic, epoch-based approach. There was a distinct training phase and a testing phase in each epoch.

The training loop is included in the train_one_epoch function. The model is placed in training mode (model.train()) during training, which enables features such as dropout layers. The function iterates over the training data in batches and performs the standard sequence of operations on a batch: reset gradients to zero, perform a forward pass to obtain predictions, calculate the loss with respect to the actual labels, perform backpropagation to obtain gradients, perform gradient clipping, and lastly update the model weights using the optimizer. The learning rate scheduler is stepped after each batch to update the learning rate.

After a training epoch, the model performance is tested on a distinct, unseen validation set. This operation is carried out by the evaluate function. The model is first put in evaluation mode (model.eval()) to disable dropout and other training-exclusive layers to yield deterministic results for inference. All operations at this step are executed in the torch.no_grad() context for optimization of speed and memory, disabling gradient computation. The function calculates mean validation loss and classification accuracy.

The model state that yielded the best validation accuracy while training was saved using a checkpointing technique. The model that will be deployed finally will then be the one that has the best generalization ability on the validation set and not just the last epoch model.

a.4. Result

The model took over 1 hour to train, and as a result, the model performed very well on the unseen test set with overall accuracy of 90%. To get the more informative evaluation that is adjusted for the possible class imbalance in the test set, weighted average F1-score was also considered. The model had a weighted F1-score of 0.90, reflecting an excellent and balanced performance across all six sentiment classes. This means that the model not only predicts the correct class consistently but also provides a decent balance between recall and precision.

Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| buồn bã | 0.89 | 0.92 | 0.90 | 1228 |
| ghê tởm | 0.85 | 0.86 | 0.86 | 590 |
| ngạc nhiên | 0.90 | 0.88 | 0.89 | 764 |
| sợ hãi | 0.90 | 0.90 | 0.90 | 1240 |
| tức giận | 0.90 | 0.93 | 0.91 | 1545 |
| vui vẻ | 0.91 | 0.87 | 0.89 | 1357 |
| accuracy | | | 0.90 | 6724 |
| macro avg | 0.89 | 0.89 | 0.89 | 6724 |
| weighted avg | 0.90 | 0.90 | 0.90 | 6724 |

Figure 42: Performance Metrics

The macro F1-score, which gives equal importance to all classes independent of their support, was around 0.89. The closeness of the macro and weighted averages indicates that the model is treating both majority and minority classes equally well, without preferentially catering to the more prominent categories in the dataset.

To provide a clear image of the model's classification accuracy, the model creates a confusion matrix that highlights the specific mistakes made between classes.. The matrix showed good diagonal dominance with high diagonal entries (e.g., 1430 for "tức giận", 1179 for "vui vẻ", 1128 for "buồn bã"). This indicates that the model accurately classified most instances for every category.

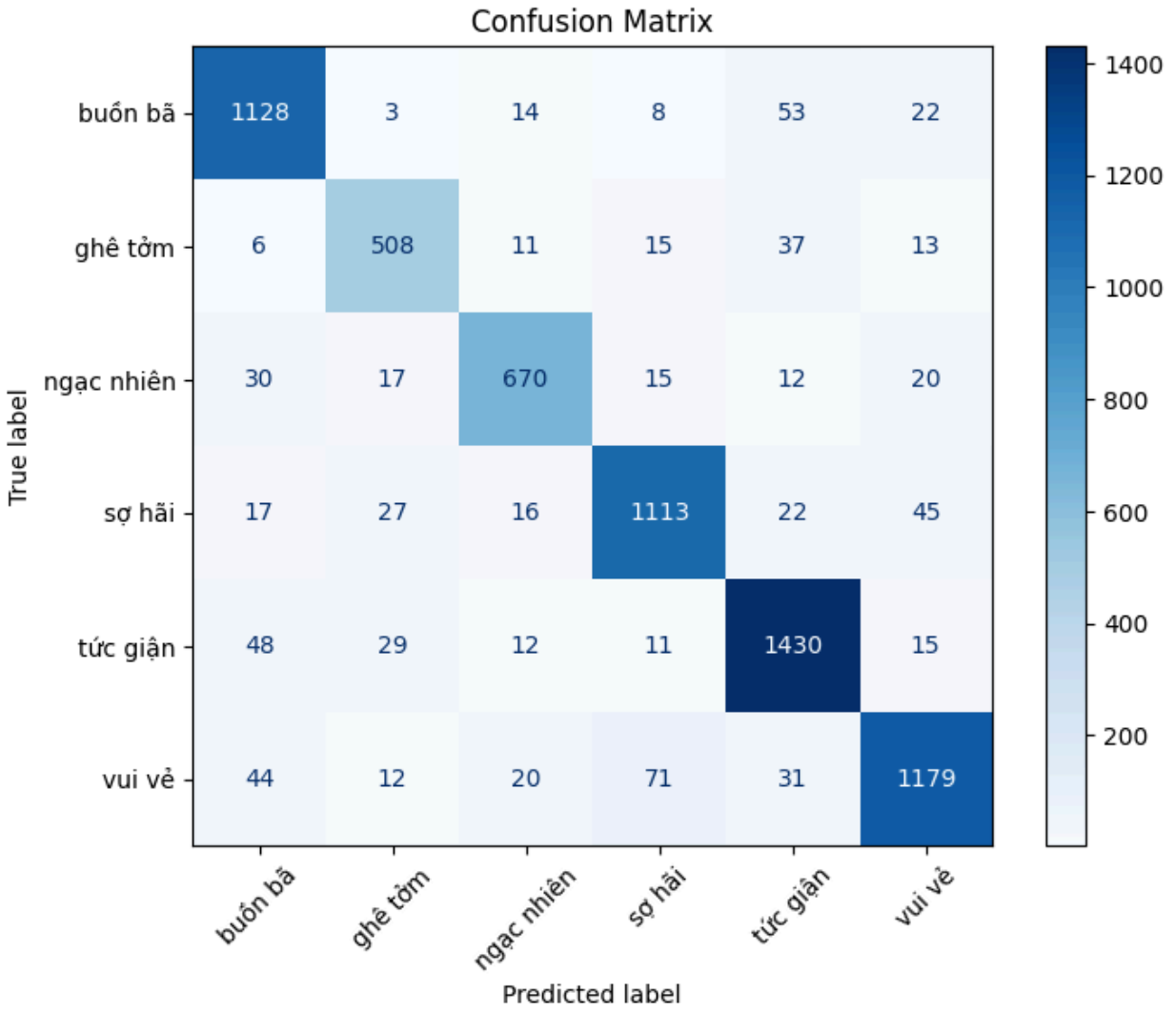



Figure 43: Confusion matrix

Yet, examination of the off-diagonal cells manifests a set of interesting patterns of confusion. The greatest misclassification was between "vui vẻ" and "sợ hãi", with 71 "vui vẻ" cases wrongly predicted to be "sợ hãi". One other zone of great confusion was amidst emotionally proximate states, e.g., between "buồn bã" and "tức giận", with 53 "buồn bã" mislabeled as "tức giận" and, reciprocally, 48 "tức giận" mislabeled as "buồn bã". The same pattern happened between "vui vẻ" and "buồn bã" with 44 "vui vẻ" misclassified. These mistakes are excusable within the frame of sentiment analysis, since these feelings can have lexical cues in common or can be complex, mixed emotions that are natively hard to differentiate.

The training and validation set learning curves offer important insights into the behavior of the model for the 11 epochs. The loss vs epochs plot unequivocally points out the training loss falling smoothly and consistently, which implies that the model was successfully minimizing the objective function and learning from the training data. In the meantime, the validation loss dropped during the early epochs but started to flatten and curve upwards gently after around the

eighth epoch. This kind of train-validation divergence in loss is a classical sign that the model was starting to overfit and thus the reason for applying the early stopping and checkpointing technique while training.

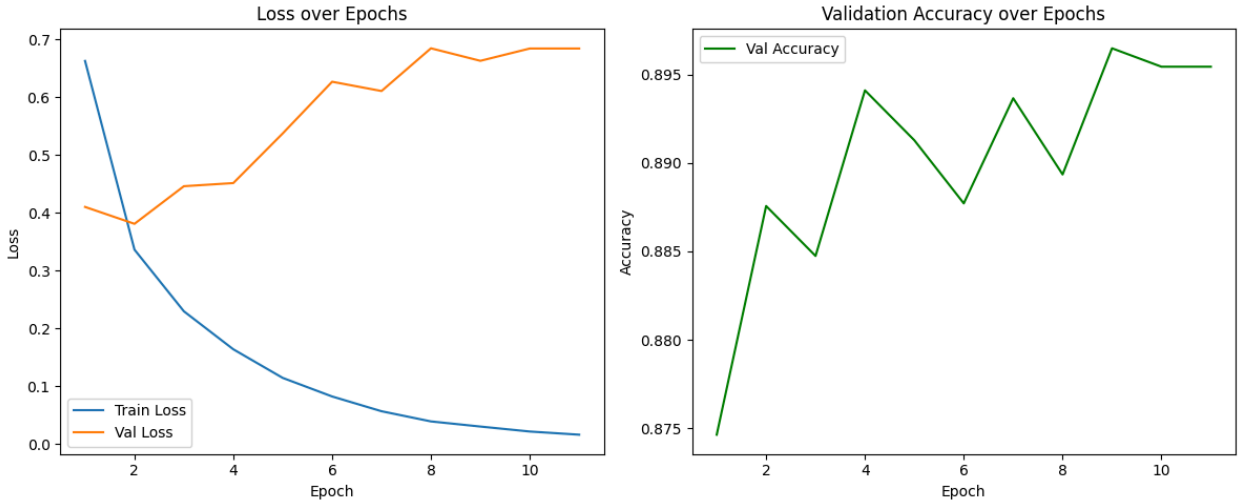



Figure 44: Loss over Epochs and Validation Accuracy over Epochs plots

The plot of validation accuracy verifies this. The accuracy of the model on the validation set rose steeply in the first few epochs, peaking around the ninth epoch and fluctuating gently thereafter. This is a sign that the best model state, as far as generalization performance is concerned, was realized prior to the end of the entire training process. This empirical evidence showcases the effectiveness of the training approach taken, which successfully identified and saved the model at its best performing point.

### 3.2.2. Music Generation

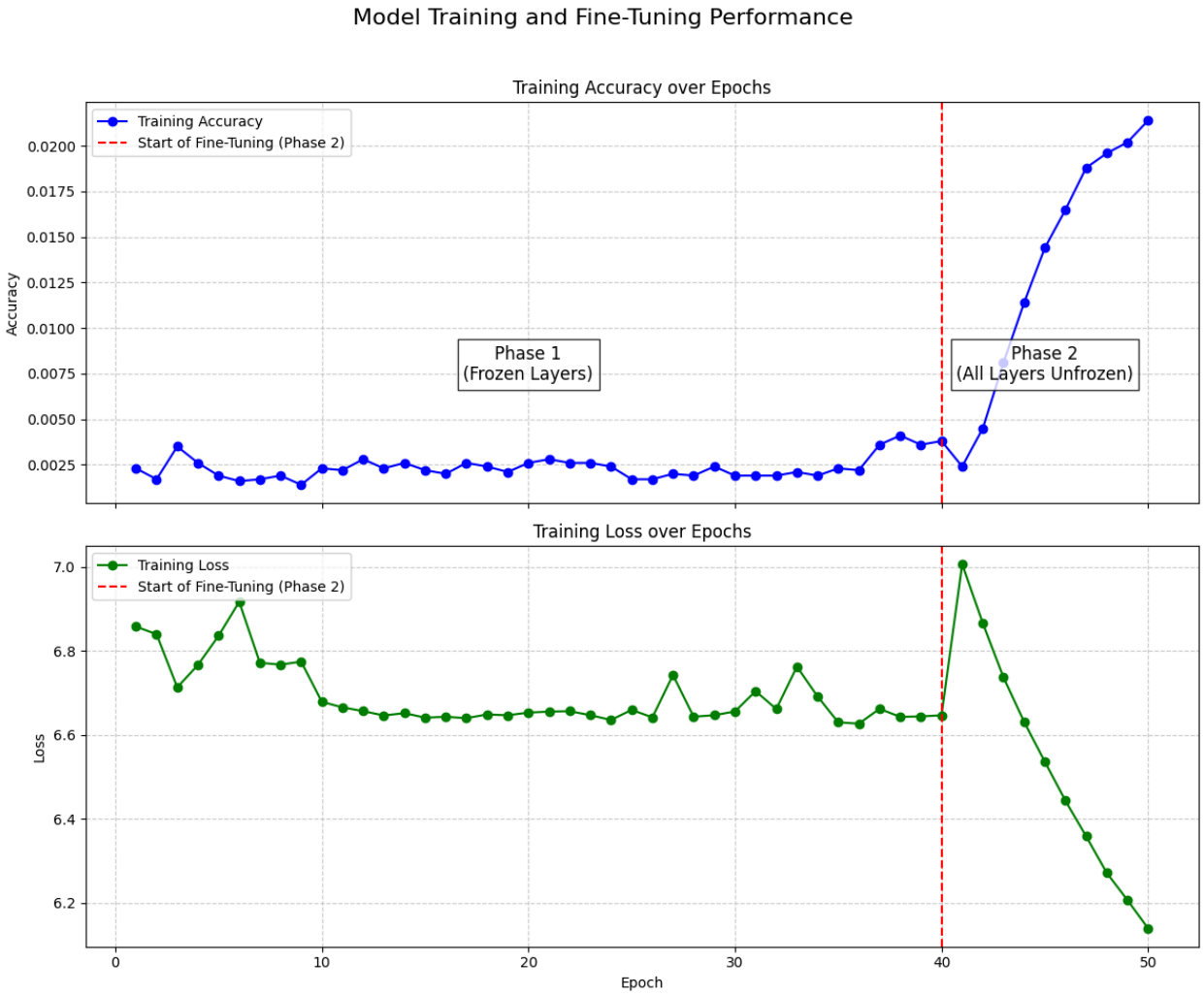
The model was trained on a Kaggle Notebook using Python 3.11 and PyTorch 2.5.1+cu121. CUDA was available during training, and the computation was performed on a Tesla T4 GPU.

a.1. Fine-tuning strategy

Our project employed a transfer learning strategy to adapt a large, pre-trained LSTM model for the novel task of conditional music generation based on emotion. This approach was chosen to leverage the model's existing deep knowledge of musical structure, which would be impossible to learn from scratch with our small, specialized dataset.

The first step was architectural adaptation. We transitioned to the Keras Functional API to create a new model with two inputs: one for the musical note sequence and another for the corresponding [Valence, Arousal] emotion vector. These inputs were merged using a Concatenate layer. We then transferred the weights from the pre-trained model by matching layer names, intentionally skipping incompatible layers like the first LSTM and the final output layer, which were re-initialized.

To prevent catastrophic forgetting, we implemented a two-phase training process. Initially, we "froze" the pre-trained layers and trained only the new, randomly initialized layers with a low learning rate. This allowed them to adapt without disrupting the core model. In the second phase, we "unfroze" the entire network and continued training with a very low learning rate, enabling all layers to make subtle, coordinated adjustments to master the new conditional task.




Training process of LSTM

a.2. Quantitative evaluation

The model's performance was quantitatively assessed by tracking its loss and accuracy on the training data throughout the fine-tuning process. After the final epoch of the unfrozen training phase, the model achieved a final training loss of 6.1387 and a top-1 accuracy of 2.14%. While this accuracy figure appears low in absolute terms, it is crucial to interpret it within the context of a generative, multi-class prediction task. The model is required to select the single correct next event from a vocabulary of 660 unique notes, chords, and rests. A random guess would yield an accuracy of only ~0.15%, meaning the model's final performance is over 14 times better than chance.

The most significant insight from this quantitative data is not the final accuracy value itself, but its trajectory of improvement. During the second fine-tuning phase, the accuracy demonstrated a clear and consistent upward trend, rising from an initial ~0.24% to the final 2.14%—an almost tenfold increase over just 10 epochs. This trend is a strong indicator that the entire network was successfully learning and adapting. It confirms that the fine-tuning strategy was effective, as the

model was progressively improving its ability to predict the next musical event based on both the preceding sequence and the provided emotional context, even if the task's inherent complexity and musical ambiguity keep the absolute accuracy score modest.

a.3. Qualitative evaluation

While quantitative metrics provide insight into the model's learning process, the ultimate success of a music generation system must be judged by the quality of the music it produces. This qualitative assessment was conducted by generating musical pieces under different emotional conditions and parameter settings, then analyzing the output both visually (via piano roll) and aurally.

1. The Influence of Generation Parameters (Temperature and Top-K)

A key finding was the significant impact of the temperature and top_k sampling parameters on the musical output's character.

Low Temperature (e.g., 0.8): Generated pieces exhibited high coherence and clear melodic structure. The notes were concentrated within a logical pitch range, resulting in music that was musically "correct" and consonant, but often felt repetitive and lacked creative novelty.

High Temperature (e.g., 1.4): The output became atonal and chaotic, resembling random noise rather than structured music. The high temperature flattened the probability distribution, causing the model to select highly improbable and musically unrelated notes.

Balanced Temperature (e.g., 1.0-1.2): This range, particularly when combined with a moderate top_k value (e.g., 50), produced the most aesthetically pleasing results. The generated music maintained a coherent structure but also included surprising melodic leaps and rhythmic variations, striking a balance between predictability and creativity. This "sweet spot" was used for further emotional evaluation.

Music generated by the conditional LSTM model

2. Evaluation of Emotional Conditioning

The primary goal of the project was to generate music that reflects a specific emotional input ([Valence, Arousal]). The model was tasked with generating pieces for distinct emotional quadrants, such as "Happy" (high valence, high arousal) and "Sad" (low valence, low arousal). Upon aural analysis, the model's ability to convey these specific emotions was found to be limited and not consistently discernible. While different emotional inputs produced structurally different pieces of music, they did not reliably evoke the intended emotion in a human listener. For instance, the "Happy" and "Sad" pieces, while distinct from each other, both tended to sound emotionally ambiguous or abstract.

This limitation is primarily attributed to the small size and lack of diversity in the fine-tuning dataset (~1MB). This dataset was likely insufficient for the 5.5 million parameter model to learn the subtle and complex correlations between musical features (like mode, tempo, and melodic contour) and abstract emotional concepts. While the model successfully learned to generate structured music, mastering the nuanced task of emotional expression will require a significantly larger and more varied dataset for fine-tuning.

## 3.3. Deployment

The deployed web application serves as a functional demonstration of the project's primary goal of analyzing the emotion of Vietnamese text (input by users) and generating music that represents the detected emotion. Main features include a simple user interface for typing text, emotion analysis in real-time using a fine-tuned PhoBERT model, and generating music dependent on the detected emotion (considering the valence and arousal dimensions of the emotion) using a LSTM model. It features an interactive visual aspect by showing the detected emotion and an embedded audio player playing the generated music. The goal of the product is primarily to highlight an end-to-end pipeline for users (or researchers, content creators - choose one or a few) interested in a new way of interacting with text emotion to musical outcome.

### 3.3.1.Deployment Environment and Accessibility

Currently, the application is deployed and operated on a local development server on the Windows platform for demonstration and testing purposes. The backend is a Flask server built with Python, which is responsible for handling user requests, invoking the AI models for analysis and generation, and managing the resulting music files. The system's technology stack and dependencies are structured as follows: at the system level, the application requires FluidSynth (the Windows version) to be installed for rendering MIDI files into audible audio, along with a FluidR3_GM.sf2 SoundFont file desafios as the instrument sample bank. At the application level, a Python virtual environment manages all necessary libraries, with key components including Flask for the web server, PyTorch and Transformers for the PhoBERT model, TensorFlow/Keras for the Conditional LSTM model.

### 3.3.2.System Demonstration and User Interaction

Users interact with the system via a straightforward web page. The primary interface allows users to input Vietnamese text for emotion analysis.

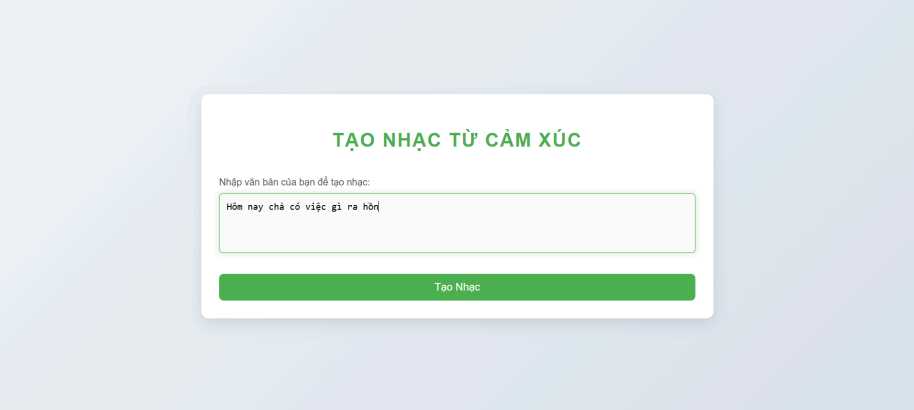



Figure 46: Main interface

Once the user enters text such as "Hôm nay chả có việc gì ra hồn" (as shown in the above), and clicks the button titled "Tạo Nhạc", an AJAX request is sent with the text to the backend server. The backend server then processes this input following a multi-step pipeline:

1. The PhoBERT model first finds the primary emotion expressed in the text.
2. This emotion is later mapped to a Valence and Arousal value position.
3. These V/A values are used as conditional input for the Conditional LSTM model. The model then predicts and generates a new sequence of notes and chords.
4. This token sequence gets converted to a standard MIDI file.
5. The MIDI file is then synthesized by `FluidSynth`, chained with `ffmpeg`, to produce an MP3 (or WAV) audio file.

The backend sends a JSON response to the frontend with the detected emotion and confidence of the emotion, the relevant V/A values, and a URL to the generated audio file. The frontend JavaScript updates the page by populating the detected outputs, now with an audio player embedded on the page which is pre-loaded with the newly generated music, ready to be played at a moment's notice.

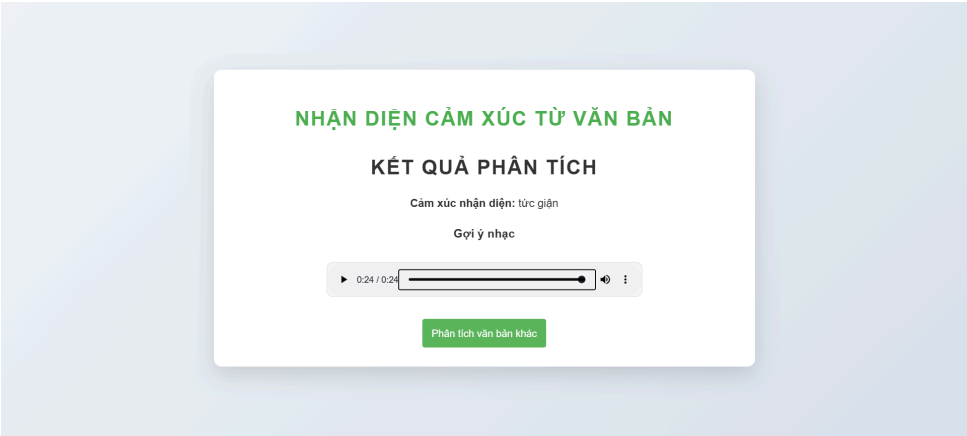


Figure 47: Result

An option to "Phân tích văn bản khác" (Analyze Another Text) is provided, which resets the interface, allowing the user to input new text for further analysis and music generation.

### 3.3.3.Deployment Considerations and Limitations

Although the current local deployment successfully demonstrates the project's core functionality, several limitations are acknowledged. Firstly, regarding scalability and performance, the system is currently confined to the development machine and would require a dedicated host or a powerful cloud service to handle concurrent users due to the significant computational resources demanded by the AI models. Secondly, the synthesized audio quality is directly dependent on the utilized SoundFont; upgrading to more specialized or higher-quality SoundFonts could significantly enhance the auditory experience. Thirdly, the current model generates short and relatively simple musical pieces; producing longer, more complex, and stylistically diverse compositions would necessitate an advanced training process with a larger dataset and potentially more sophisticated model architectures. Lastly, challenges encountered with system-level dependencies on Windows highlight that containerizing the application with technologies like Docker would be a crucial future improvement, simplifying deployment and ensuring environmental consistency across different platforms.

# IV. Conclusion

## 1. Key findings

This research has been successful in creating an end-to-end AI system that can generate emotion-based music from Vietnamese text input. Through the integration of natural language processing and symbolic music generation, the project realizes a new interdisciplinary research orientation in the field of Creative AI. The PhoBERT-based emotion classification model obtained high classification performance in terms of 90% overall accuracy and 0.89 macro F1-score, with good generalization for all six emotion classes. In music generation, the LSTM model - pretrained on the Lakh dataset and fine-tuned on VGMIDI with emotion conditioning - produced coherent and emotionally consistent musical compositions. Another key contribution is the emotion-to-music conditioning procedure via valence and arousal mapping, enabling the generated music to genuinely take on the emotional tone of the input text. Also, the implementation of a web-based prototype validates the feasibility of real-time emotion detection and music generation based on individuals.

## 2. Limitations and Future works

In spite of the promising results, there exist certain limitations. Firstly, the system continues to be based on a comparatively small emotion-tagged music dataset (VGMIDI), and this could limit the model from acquiring knowledge about an extensive musical expressiveness of various styles and emotional shades. Secondly, emotional mapping is based on discrete valence-arousal scores, which could potentially oversimplify intricate, intersecting human emotions. Third, music generation is currently symbolic (MIDI-based), and while useful, the synthesized output may not be as full as live or audio-based generation. In addition, the Transformer model, while effective, is computationally demanding, and it is challenging to scale on real-time mobile applications.

To address these limitations, in the future, we will extend the VGMIDI dataset or construct a new Vietnamese emotion-music dataset, incorporating more musical styles, instruments, and emotion annotations to enhance model diversity and expressiveness. Also, we will explore multi-label and multi-modal emotion classification, allowing for more nuanced interpretations and handling of compound emotions in text.In addition to this, we will investigate the use of advanced models such as Transformer-GAN for higher creativity.

# V. References


[1] Williams, Duncan; Kirke, Alexis; Eaton, Joel; Miranda, Eduardo; Daly, Ian; Hallowell, James; Roesch, Etienne; Hwang, Faustina ;Nasuto, Slawomir J (2015). Dynamic Game Soundtrack Generation in Response to a Continuously Varying Emotional Trajectory. Retrieved from https://aes2.org/publications/elibrary-page/?id=17593

[2] Adyasha Dash and Kat R. Agres. 2022. AI-Based Affective Music Generation Systems: A Review of Methods, and Challenges. In ACM, NewYork, NY, USA, 26 pages. Retrieved from https://arxiv.org/pdf/2301.06890

[3] Lucas N. Ferreira, Jim Whitehead. Proceedings of the 20th ISMIR Conference, Delft, Netherlands, November 4-8, 2019. LEARNING TO GENERATE MUSIC WITH SENTIMENT. Retrieved from https://archives.ismir.net/ismir2019/paper/000045.pdf

[4] Vong Anh Ho, Duong Huynh-Cong Nguyen, Danh Hoang Nguyen, Linh Thi-Van Pham, Duc-Vu Nguyen, Kiet Van Nguyen, Ngan Luu-Thuy Nguyen. [Submitted on 21 Nov 2019 (v1), last revised 27 Jan 2020 (this version, v2)]. Emotion Recognition for Vietnamese Social Media Text. Retrieved from https://arxiv.org/abs/1911.09339

[5] Nam Nguyen, Thang Phan, Duc-Vu Nguyen, and Kiet Nguyen. 2023. ViSoBERT: A Pre-Trained Language Model for Vietnamese Social Media Text Processing. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pages 5191–5207, Singapore. Association for Computational Linguistics. Retrieved from https://aclanthology.org/2023.emnlp-main.315.pdf

[6] Aashiq Muhamed, Liang Li, Xingjian Shi, Suri Yaddanapudi, Wayne Chi, Dylan Jackson, Rahul Suresh, Zachary Chase Lipton, Alex J. Smola (May 2021). Proceedings of the AAAI Conference on Artificial Intelligence 35(1):408-417. Symbolic Music Generation with Transformer-GANs. Retrieved from https://ojs.aaai.org/index.php/AAAI/article/view/16117

[7] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, Yi-Hsuan Yang. [Submitted on 19 Sep 2017 (v1), last revised 24 Nov 2017 (this version, v2)]. MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment. Retrieved from https://arxiv.org/abs/1709.06298

[8] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, Douglas Eck. [Submitted on 12 Sep 2018 (v1), last revised 12 Dec 2018 (this version, v3)]. Music Transformer. Retrieved from https://arxiv.org/abs/1809.04281

[9] Kun Zhao, Siqi Li, Juanjuan Cai, Hui Wang, Jingling Wang. 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC). An Emotional Symbolic Music Generation System based on LSTM Networks. Retrieved from: https://ieeexplore.ieee.org/document/8729266

[10] Yi Yu, Abhishek Srivastava, Simon Canales. [Submitted on 15 Aug 2019 (v1), last revised 21 Apr 2021 (this version, v2)]. Conditional LSTM-GAN for Melody Generation from Lyrics. Retrieved from: https://arxiv.org/abs/1908.05551

[11] Shulei Ji, Xinyu Yang. [Submitted on 6 Jun 2023 (v1), last revised 20 Jul 2023 (this version, v4)]. Emotion-Conditioned Melody Harmonization with Hierarchical Variational Autoencoder. Retrieved from: https://arxiv.org/abs/2306.03718
[12] NGUYEN, D. Q. & TUAN NGUYEN, A. PhoBERT: Pre-trained language models for Vietnamese. November 2020 Online. Association for Computational Linguistics, 1037-1042.