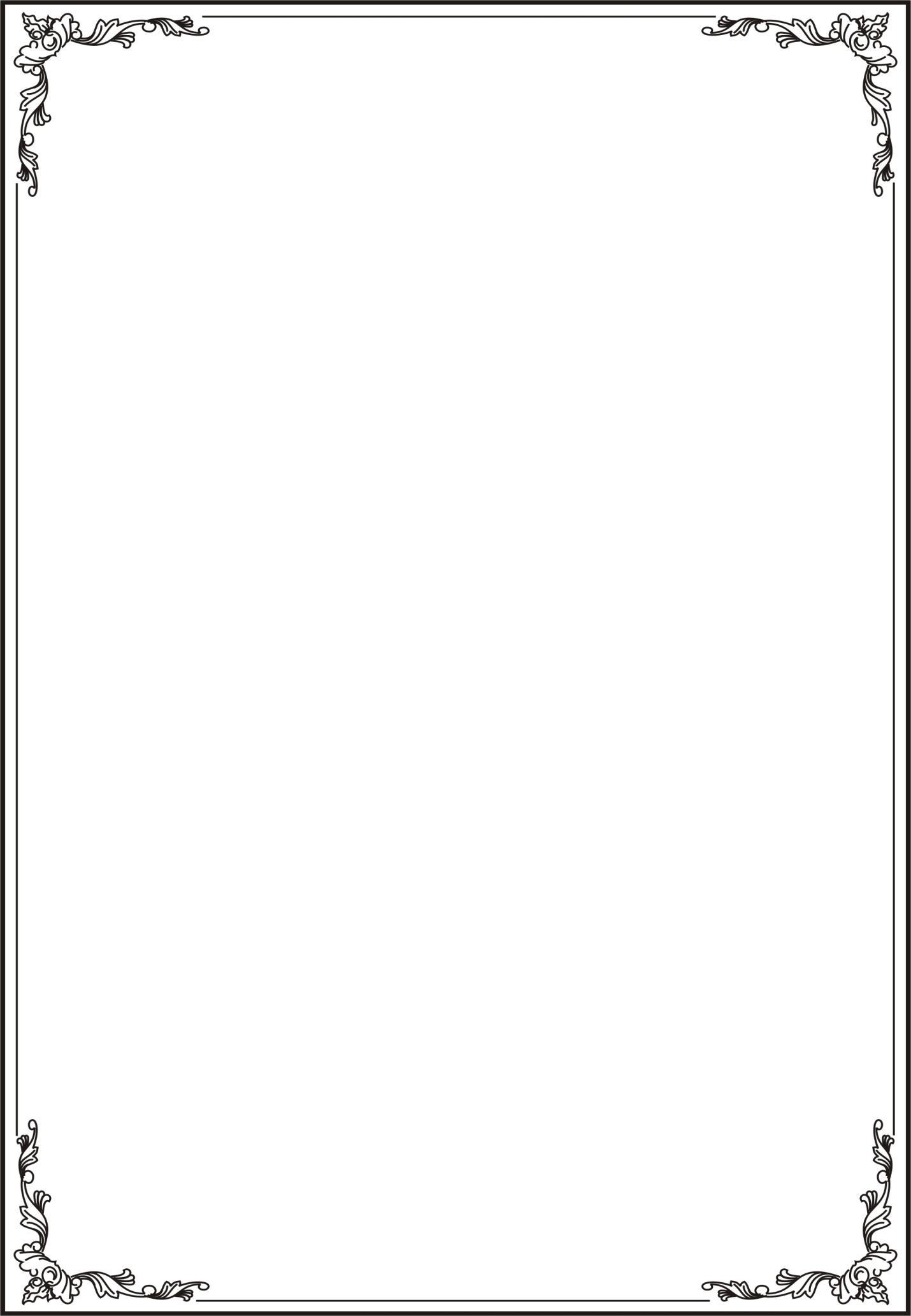
****

**HO CHI MINH UNIVERSITY OF TECHNOLOGY AND EDUCATION**

**FACULTY FOR HIGH-QUALITY TRAINING**

**----------------------------**



**Capstones Project**

**Topic: Building Remider Mobile Application**

**using Machine Learning**

**Instructor: Tran Nhat Quang, MSc**

|  |  |
| --- | --- |
| Nguyen Hoang Anh Khoa | 19110514 |
| Nguyen Tan Dat | 19110116 |
| Thai Thi Thu Thao | 19110515 |

**Members:**

*Ho Chi Minh City, Dec 1st 2023*

**SOCIALIST REPUBLIC OF VIETNAM**

**Independence - Freedom – Happiness**

**\*\*\*\*\*\*\***

**GRADUATION THESIS TASK**

**Student’s Name:**

|  |  |  |
| --- | --- | --- |
| 1.Nguyen Hoang Anh Khoa | ID: 19110514 | Class: 1911CLA5 |
| 2.Nguyen Tan Dat | ID: 19110116 | Class: 1911CLA4 |
| 3. Thai Thi Thu Thao | ID:19110515 | Class: 1911CLA4 |

Major: **Information Technology**

Lecturer: **Tran Nhat Quang, MSc**  Phone: ………………

The date of receiving the topic: 02/09/2023 Thesis submission date: 01/12/2023

Project name: **Building Remider Mobile Application using Machine Learning**

Content implementation of the topic:

- Research and analysis stock market behavior by reading papers.

- Study models that have performing well on forecasting time series dataset.

- Training and Evaluation models have well performance.

- Build a mobile application that can help user get notification with time specify, using NLP and voice regconization

LECTURER

**Tran Nhat Quang**

**SOCIALIST REPUBLIC OF VIETNAM**

**Independence – Freedom – Happiness**

\*\*\*\*\*\*\*

**INSTRUCTOR’S COMMENTARY**

**Students’ names:**

|  |  |  |
| --- | --- | --- |
| 1.Nguyen Hoang Anh Khoa | ID: 19110514 | Class: 1911CLA5 |
| 2.Nguyen Dat | ID: | Class: |
| 3. Thai Thi Thu Thao | ID: | Class: |

Major: **Information Technology**

Project’s name: **Building Remider Mobile Application using Machine Learning**

Reviewer’s full name: **Tran Nhat Quang, MSc**

**COMMENTARY**

Project’s contents and workload:

**Theory:**

- Research and learn Javascript, Python, Machine Learning, Flask, Flutter, Dart.

**Implementation:**

- Build a mobile application that can help user get notification with time specify, using NLP and voice regconization.

**Results:**

- The contents and workload are appropriate for an undergraduate capstone project. Students can apply the technologies learned to successfully build a system with basic features.

**Link:**

https://github.com/anhkhoanghg/gpt.git

**Advantages**:

* The technologies learned and applied are contemporary and powerful.
* The features of the system can somehow fulfill the business process.
* The models performing well on entire datasets.

**Disadvantages**:

* The features are quite simple. The group should have built more advanced features for better support to the users.
* Text Analysis has to be improve.
* Website should update.

Lack Approving for dissertation defense or not? Yes.

**Achievement level:** Good.

*Ho Chi Minh, Thu December 25th 2023*

*Supervisor*

*(Sign, write full name)*

**Tran Nhat Quang**

**SOCIALIST REPUBLIC OF VIETNAM**

**Independence – Freedom – Happiness**

\*\*\*\*\*\*\*

**REVIEWER’S COMMENTARY**

**Students’ names:**

|  |  |  |
| --- | --- | --- |
| 1.Nguyen Hoang Anh Khoa | ID: 19110514 | Class: 1911CLA5 |
| 2.Nguyen Dat | ID: | Class: |
| 3. Thai Thi Thu Thao | ID: | Class: |

Major: **Information Technology**

Project’s name: **Building Remider Mobile Application using Machine Learning** Reviewer’s full name:

**COMMENTARY:**

1. Regarding the content of the topic and the volume of implementation:

1. Strength:

1. Drawback:

1. Recommend for defending or not?

5. Grade:

6. Mark:

*Ho Chi Minh, December 25st 2023*

Reviewer

*(Sign, write full name)*

***Contents***

[**Chapter 01: Related work** 8](#_Toc137516930)

[**1.1. Stock Price Prediction Using LSTM on Indian Share Market** [1]**:** 8](#_Toc137516931)

[**1.2. AEI-DNET: A Novel DenseNet Model with an Autoencoder for the Stock Market Predictions Using Stock Technical Indicators** [2]**:** 8](#_Toc137516932)

[**1.3. Predict stock prices with ARIMA and LSTM** [3]**:** 8](#_Toc137516933)

[**1.4. Convolutional neural networks for time series classification** [4]**:** 8](#_Toc137516934)

[**1.5. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling** [5]**:** 8](#_Toc137516935)

[**Chapter 02: Introduction** 10](#_Toc137516936)

[**Chapter 03: Theory Fundamental** 12](#_Toc137516937)

[**3.1.** **System Architecture** 12](#_Toc137516938)

[**3.2.** **Library** 12](#_Toc137516939)

[**3.3.** **Technologies** 14](#_Toc137516940)

[**3.3.1.** **Flask** 14](#_Toc137516941)

[**3.3.2.** **TensorFlow** 14](#_Toc137516942)

[**3.3.3.** **Torch** 14](#_Toc137516943)

[**Chapter 04: System Analysis and Design** 14](#_Toc137516944)

[**4.1. Use-case diagram** 15](#_Toc137516945)

[**4.2. Use-case specification** 15](#_Toc137516946)

[**5.1. Introduction** 18](#_Toc137516947)

[**5.2. Visual Design and Layout** 18](#_Toc137516948)

[**5.3. Key Sections and Features:** 18](#_Toc137516949)

[**5.4. User Experience and Responsiveness:** 20](#_Toc137516950)

[**5.5. Conclusion** 21](#_Toc137516951)

[**Chapter 06: Data preparation** 22](#_Toc137516952)

[**6.1 Data Collection** 22](#_Toc137516953)

[**6.2. Data Preprocessing** 23](#_Toc137516954)

[**6.3. Time Series Data Preparation** 26](#_Toc137516955)

[**6.4. Indicators:** 27](#_Toc137516956)

[6.4.1. Momentum 27](#_Toc137516957)

[6.4.2. William Percentage R 28](#_Toc137516958)

[6.4.3. RSI 28](#_Toc137516959)

[6.4.4. Commodity Channel Index 29](#_Toc137516960)

[6.4.5. Moving Average Convergence/Divergence 29](#_Toc137516961)

[6.4.6. Directional Movement 30](#_Toc137516962)

[6.4.7. Forecast Oscillator 31](#_Toc137516963)

[6.4.8. Chande Momentum Oscillator 31](#_Toc137516964)

[6.4.9. Efficiency Ratio 32](#_Toc137516965)

[6.4.10. Rate of Change 33](#_Toc137516966)

[6.4.11. The Schaff Trend Cycle 34](#_Toc137516967)

[6.4.12. Elder's Bulls Ray Index 34](#_Toc137516968)

[6.4.14. Simple Moving Average 35](#_Toc137516969)

[6.4.15. Exponential Moving Average 36](#_Toc137516970)

[6.4.16. Volume Weighted Average Price 37](#_Toc137516971)

[6.4.17. Hull Moving Average 37](#_Toc137516972)

[6.4.18. Chaikin Money Flow 38](#_Toc137516973)

[6.4.19. Ultimate Oscilators 39](#_Toc137516974)

[**6.4.20. Slope 39**](#_Toc137516975)

[**Chapter 07: Methodology** 41](#_Toc137516976)

[**7.1. LSTM** 41](#_Toc137516977)

[**7.2. Random Forest** 41](#_Toc137516978)

[**7.3. Support Vector Machine** 42](#_Toc137516979)

[**7.4. Natural Language Processing** 42](#_Toc137516980)

[**7.5. XGBoost** 48](#_Toc137516981)

[**Chapter 08: Evaluation** 50](#_Toc137516982)

[**8.1. Sectors Predicted** 50](#_Toc137516983)

[**8.2. Model Evaluation** 50](#_Toc137516984)

[**Chapter 09: Conclusion** 66](#_Toc137516985)

[**References** 68](#_Toc137516986)

# **Chapter 01: Related work**

## **1.1. Discord Reminder Bot with the NVIDIA Jetson Nano**

Achyut Ghosh et al. conducted a study using LSTM to identify the ideal window for predicting future share prices. at various banks and sectors over various time periods. The study concludes that businesses in the same industry have comparable growth rates and dependencies. A larger dataset can be used to train the model, which will increase prediction accuracy. The findings indicate that the prediction error will gradually decrease over time, and that the longer the prediction period, the less error there will be.

## **1.2. AEON: A Method for Automatic Evaluation of NLP Test Cases**

These test cases require extensive manual checking effort, and instead of improving NLP software, they can even degrade NLP software when utilized in model training. To address this problem, we propose AEON for Automatic Evaluation Of NLP test cases. For each generated test case, it outputs scores based on semantic similarity and language naturalness. We employ AEON to evaluate test cases generated by four popular testing techniques on five datasets across three typical NLP tasks. The results show that AEON aligns the best with human judgment. In particular, AEON achieves the best average precision in detecting semantic inconsistent test cases, outperforming the best baseline metric by 10%. In addition, AEON also has the highest average precision of finding unnatural test cases, surpassing the baselines by more than 15%. Moreover, model training with test cases prioritized by AEON leads to models that are more accurate and robust, demonstrating AEON's potential in improving NLP software.

## **1.3. Increasing Students' Engagement to Reminder Emails Through Multi-Armed Bandits**

Using Multi-Armed Bandits (MAB) algorithms like Thompson Sampling (TS) in adaptive experiments can increase students' chances of obtaining better outcomes by increasing the probability of assignment to the most optimal condition (arm), even before an intervention completes. This is an advantage over traditional A/B testing, which may allocate an equal number of students to both optimal and non-optimal conditions. The problem is the exploration-exploitation trade-off. Even though adaptive policies aim to collect enough information to allocate more students to better arms reliably, past work shows that this may not be enough exploration to draw reliable conclusions about whether arms differ. Hence, it is of interest to provide additional uniform random (UR) exploration throughout the experiment.

# **Chapter 02: Introduction**

* 1. **Reason for chosen topic**
  2. **Purpose of project**
  3. **Object and Scope**
  4. **Expected Result**

# **Chapter 03: Theory Fundamental**

* 1. **System Architecture**
  2. **Library**
     1. **Front-end**

|  |  |  |
| --- | --- | --- |
| Library | Version | Description |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

* + 1. **Back-end**

|  |  |  |
| --- | --- | --- |
| Library | Version | Description |
| scikit-learn | 1.1 | data mining and data analysis |
| transformer | 4.29.2 | save you the time and resources required to train a model |
| torch | 2.0 | load data, build deep neural networks, train and save your models |
| pandas | 2.0.2 | repetitive tasks associated with working with data |
| numpy | 1.17.3 | perform a wide variety of mathematical operations on arrays |
| request | 2.31.0 | send HTTP requests using Python |
| tqdm | 4.65.0 | create progress bars, training machine learning models, multi-loop Python function, and downloading data |
| flask | 2.3.2 | developing web applications |
| subprocess | 0.0.8 | run new codes and applications by creating new processes |
| pathlib | 1.0 | provides a modern and Pythonic way of working with file paths, making code more readable and maintainable |
| math | 3.11.3 | use the built-in mathematical operators |
| beautifulsoup | 4.12.2 | used for web scraping purposes to pull the data out of HTML and XML files |
| parsedatetime | 2.6 | Parse human-readable date/time strings |
| python\_dateutil | 2.8.2 |  |
| pandas\_ta | 0.3.14b | Easier to use library |
| spacy | 3.5.3 | spaCy is designed for tasks such as part-of-speech tagging, named entity recognition, and dependency parsing. |
| gunicorn | 19.7.1 | Gunicorn provides a simple and efficient way to serve web applications with multiple worker processes. |

* 1. **Technologies**
     1. **Flask**

Flask is a popular and lightweight web framework for building web applications using the Python programming language. It is known for its simplicity, flexibility, and ease of use, making it a popular choice among developers, especially for small to medium-sized projects.

Flask supports template engines like Jinja2, which allows developers to separate the presentation logic from the application's code. Templating enables the creation of dynamic HTML pages by rendering data into predefined templates.

Flask uses a routing mechanism to map URLs to corresponding Python functions or views. Developers can define routes and associated functions to handle specific URLs and HTTP methods. This allows for the creation of clean and intuitive URL structures.

Flask-WTF is an extension that integrates with Flask to provide support for handling web forms. It simplifies form validation, CSRF protection, and other form-related tasks.

* + 1. **TensorFlow**

TensorFlow is an open-source machine learning framework developed by Google. It is widely used for building and training machine learning models, particularly deep neural networks. TensorFlow provides a comprehensive ecosystem of tools, libraries, and resources that enable developers to create and deploy machine learning applications efficiently.

* + 1. **Torch**

Like TensorFlow, Torch, or PyTorch, is an open-source machine learning framework widely used for developing and training deep learning models. It provides a flexible and efficient platform for building neural networks and conducting various machine learning tasks. Torch is known for its dynamic computational graph, extensive support for GPU acceleration, and easy-to-use APIs.

# **Chapter 04: System Analysis and Design**

## **4.1. Use-case diagram**

## **4.2. Use-case specification**

1. **View analysis chart**

|  |  |
| --- | --- |
|  | |
| **Description** |  |
| **Activation agent** |  |
| **Pre-conditions** |  |
| **Step to be taken** |  |

1. **Customize chart specificities**

|  |  |
| --- | --- |
|  | |
| **Description** |  |
| **Activation agent** |  |
| **Pre-conditions** |  |
| **Step to be taken** |  |

1. **View stock prediction**

|  |  |
| --- | --- |
|  | |
| **Description** |  |
| **Activation agent** |  |
| **Pre-conditions** |  |
| **Step to be taken** |  |

1. **View technical indicators**

|  |  |
| --- | --- |
|  | |
| **Description** |  |
| **Activation agent** |  |
| **Pre-conditions** |  |
| **Step to be taken** |  |

1. **Download analysis chart**

|  |  |
| --- | --- |
|  | |
| **Description** |  |
| **Activation agent** |  |
| **Pre-conditions** |  |
| **Step to be taken** |  |

**Chapter 05: Interface evaluation**

## **5.1. Introduction**

## **5.2. Visual Design and Layout**

## **5.3. Key Sections and Features:**

# **Chapter 06: Data preparation**

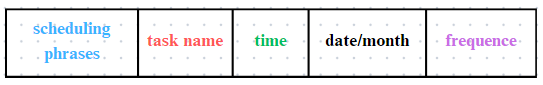
## **6.1 Data Preparation**

The data collection phase was meticulously undertaken by our team to compile a comprehensive dataset for our project. Within this dataset, we systematically cataloged various tasks that users typically engage in for their daily and routine activities. The evaluation of these tasks was performed through our own assessment, considering their regularity and significance in users' scheduling patterns.In total, our dataset encompasses around 2000 distinct tasks, each accompanied by detailed information. For each task, we not only recorded its inherent importance but also estimated the approximate time required for its completion. This dual categorization allows for a nuanced understanding of the dataset, providing insights into both the priority and temporal aspects of the tasks included.

Our team conducted a thorough analysis to ensure the relevance and representation of diverse tasks within the dataset. We aimed to capture a spectrum of activities that individuals commonly encounter in their daily lives. The tasks span various domains, reflecting the dynamic nature of users' scheduling needs.The inclusion of importance ratings provides a qualitative dimension to the dataset, enabling a nuanced exploration of task priorities. Additionally, the time estimates associated with each task offer valuable insights into the anticipated effort required for task completion. These attributes collectively contribute to the richness of our dataset, making it a robust resource for understanding user behavior and preferences in task management.

Subsequently, we leveraged the curated set of tasks to generate user-friendly sentences that individuals can employ for scheduling their activities. These sentences are carefully crafted to encompass various components, including scheduling terms, task names, time specifications, dates, and the recurrence frequency of tasks. The integration of these elements results in the creation of coherent and complete sentences that users can readily adopt for efficient task scheduling.The scheduling terms embedded in the sentences serve as cues for organizing and planning activities. These terms are strategically selected to resonate with users' scheduling preferences and habits. Additionally, task names are seamlessly integrated, ensuring clarity and specificity in the scheduling process. Users can easily identify and relate to their intended tasks through the inclusion of these names.Time specifications and date references add a temporal dimension to the sentences, enabling users to precisely allocate tasks within their schedules. Whether it's a specific time of day, a duration, or a particular date, the sentences provide flexibility to cater to diverse scheduling needs. Furthermore, the recurrence frequency of tasks is incorporated to accommodate repetitive activities, allowing users to efficiently plan and manage recurring commitments.

The culmination of these components results in well-formed sentences that encapsulate the essential details needed for effective scheduling. By providing users with ready-made sentences, our system aims to streamline the task management process, offering a convenient and user-centric approach to creating comprehensive and meaningful schedules. This approach not only enhances the user experience but also contributes to the overall efficiency and effectiveness of task scheduling within the scope of our project.

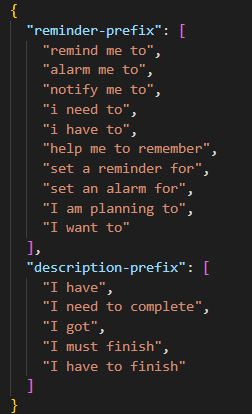


The components within the sentence can be repositioned based on the context of the utterance.

## **6.2. Data Preprocessing**

### 6.2.1 Input generation

We compiled a list of scheduling phrases, categorizing them into two parts: phrases associated with nouns and phrases associated with verbs. Alongside these lists, we also created inventories of prepositions and conjunctions to seamlessly combine them and form complete sentences.



To enable the model to learn diverse expressions conveying the same meaning, we created various sentence combinations with different phrasings. These sentences consist of components such as scheduling phrases, actions, prepositions, time indicators (day or hour expressions), and more. The quantity of each component in the sentence can vary to accommodate the different ways users may express scheduling for a particular event. This approach enriches the training data, allowing the model to grasp the flexibility and nuances in users' preferences when scheduling events.

Certainly, here are some examples illustrating the same meaning but expressed in various ways:

1. Scheduling a Meeting:
   * I plan to schedule a meeting for tomorrow at 2 PM.

[r-pre, action, prep, no\_day, prep, time]

* + Tomorrow at 2 PM, I'll set up a meeting.

[no\_date, prep, time, r-pre, action]

* + I'm organizing a meeting for 2 PM tomorrow.

[r-pre, action, prep, time, no\_date]

1. Setting a Reminder:
   * Set a reminder for the dentist appointment on Friday.

[r-pre, prep, action, prep, day\_of\_week]

* + On Friday, create a reminder for the dentist.

[prep, day\_of\_week, r-pre, action]

* + Schedule a reminder for my dentist appointment this Friday.

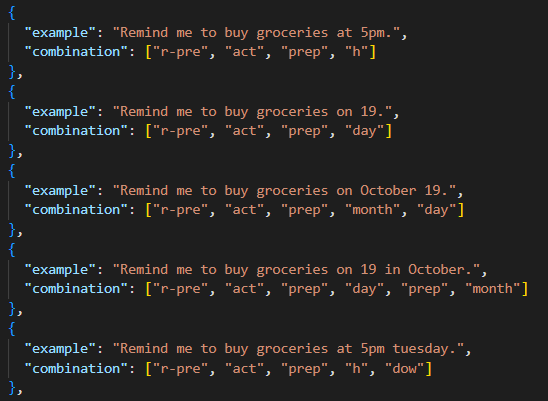
[r-pre, action, day\_of\_week]

1. Adding an Event:
   * Add an event to my calendar for the conference next week.

[r-pre, action, number\_of\_weeks]

* + I'd like to schedule the conference for next week on my calendar.

[r-pre, action, number\_of\_weeks, prep, c-nouns]



### 6.2.2 Label generation

After compiling various sentence structures, we utilized this information to combine suitable words with each action, creating multiple sentences with equivalent meanings. Upon completing the input preparation for the model, we relied on actions and time-related phrases to generate labels for each utterance. For each action, we have created 15 labels, encompassing various aspects of scheduling. These labels include:

* Task name (summarize)
* Importance level (imp)
* Frequency (single, daily, weekly)
* Approximate completion time (expected\_minutes)
* Date (day)
* Month (month)
* Time (specific\_time)
* Day of the week (dow)
* Number of weeks (no\_weeks)
* Number of months (no\_months)
* Number of days (no\_days)
* Task type (categories)
* Time of day (tod)
* Repeat frequency within a day (daily)
* Repeat frequency within a week (weekly)

These labels are designed to capture a comprehensive set of information associated with each action, providing a detailed and versatile framework for the model to understand and generate meaningful scheduling sentences.

Regarding the importance of tasks, as we don't have user-specific data, we have autonomously assessed the importance of each task based on our own criteria, assigning importance ratings on a scale of 1-5 for each task. This approach allows for a subjective yet consistent evaluation, where different tasks can be assigned varying levels of importance. Similarly, for the estimated time required to complete each task, we have undertaken a self-assessment and labeled each task with different time durations according to our judgment. This enables the model to understand and work with diverse time constraints associated with different tasks, accommodating a range of complexities in scheduling. These self-assigned importance ratings and time labels contribute to creating a foundational dataset for training the model, fostering its ability to generate accurate and contextually relevant scheduling sentences.

Depending on the context of the sentence, the remaining labels such as time, date, day, etc., will be labeled accordingly. If a particular label is not mentioned in the sentence, it will be assigned the value "None." This approach allows for flexibility in capturing the diverse information that may be present or omitted based on the user's expression in the scheduling sentence. The "None" value serves as a placeholder for labels that are not explicitly specified in a given context.

Below is a complete input and label set that we feed into the model:

{

    "input": "i need to Go to school at 19 o'clock for everyday",

    "target": "<task><sum>Go to school <cate>edu-activities<imp>3<freq>daily<exp\_min>160<totd>night<spec\_time><dow><day><month><no\_date><no\_week><no\_month><daily>19:00:00<weekly></task>"

  }

After completing the dataset, we have over 18,000 input-output pairs. Among these, we allocated 90% for training and the remaining 10% for testing purposes.



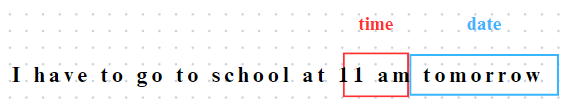
# **Chapter 07:** **Methodology**

## **7.1. Parse human-readable date/time text**

Parse-datetime determines date and time types within a sentence through its sophisticated parsing mechanism. The library employs a combination of natural language processing and rule-based algorithms to identify and extract temporal information from user-provided input. It starts by breaking down the input sentence into tokens, which are individual units of meaning. This process involves identifying words or phrases that could potentially represent date or time-related information. After tokenization, the library utilizes part-of-speech tagging to analyze the grammatical roles of words within the sentence. This step helps identify words that function as nouns, verbs, adjectives, or other parts of speech, aiding in the recognition of date and time components. parsedatetime employs pattern matching algorithms to recognize common date and time patterns in the tokenized and tagged sentence. This includes identifying expressions like "tomorrow," "next Monday," or "in two weeks.".

The model considers the context of the entire sentence to resolve ambiguities and refine its understanding of date and time information. Contextual clues, such as words like "before" or "after," help determine the relationships between different temporal elements. It also supports localization, meaning it can adapt to different languages and cultural conventions. This localization feature enhances the library's ability to understand date and time expressions in a diverse range of linguistic contexts. In cases where the parsing engine encounters uncertainty or ambiguity, parsedatetime incorporates fallback mechanisms to make educated guesses based on common usage patterns. This enhances the library's robustness in handling various user inputs.

By combining these techniques, parsedatetime can effectively identify and extract date and time information from user-provided sentences, contributing to its versatility in applications that involve dynamic scheduling and temporal comprehension.



**7.2. Natural Language Processing**

**7.2.1. Word Embedding**

Word embedding is a powerful technique in natural language processing (NLP) that transforms words or phrases into dense vector representations in a continuous vector space. It overcomes the limitations of traditional sparse and high-dimensional representations by capturing semantic and contextual relationships between words. This report provides an overview of word embedding, its significance in NLP, and its applications in various tasks.

Word embedding techniques, such as Word2Vec, have revolutionized NLP by enabling machines to understand and process textual data more effectively. Unlike traditional methods that represent words as sparse and high-dimensional vectors, word embeddings map words to dense vectors, where similar words are represented by vectors that are closer in the vector space. This dense representation captures semantic relationships, allowing algorithms to understand the meaning of words and infer relationships. For example, words like "king" and "queen" or "man" and "woman" have similar vector representations, enabling algorithms to perform word analogy tasks. Furthermore, word embeddings capture contextual similarities by assigning similar vector representations to words that appear in similar contexts. This contextual understanding enhances the performance of algorithms in various NLP tasks.

Word embeddings have found extensive applications in NLP tasks such as sentiment analysis, machine translation, text classification, and information retrieval. By utilizing word embeddings, algorithms can leverage the semantic and contextual relationships between words to improve accuracy and performance. Pre-trained word embeddings like GloVe and FastText are available and provide a solid starting point for NLP tasks. These embeddings are trained on large corpora and capture general language semantics. However, it is also possible to train domain-specific word embeddings using specific datasets to capture domain-specific semantics and contextual information. This flexibility allows NLP practitioners to tailor word embeddings to the specific requirements of their tasks and achieve better results.

In summary, word embedding is a fundamental technique in NLP that captures semantic and contextual relationships between words by representing them as dense vectors in a continuous vector space. The ability to encode semantic and contextual information within these vector representations has transformed the field of NLP, enabling algorithms to understand and process textual data more effectively. By capturing word relationships and context, word embeddings have proven invaluable in a wide range of NLP applications, contributing to improved accuracy and performance. As the field of NLP continues to advance, word embedding techniques will play a crucial role in further enhancing the capabilities of natural language understanding and processing systems.

A picture containing text, screenshot, businesscard, font

Description automatically generated

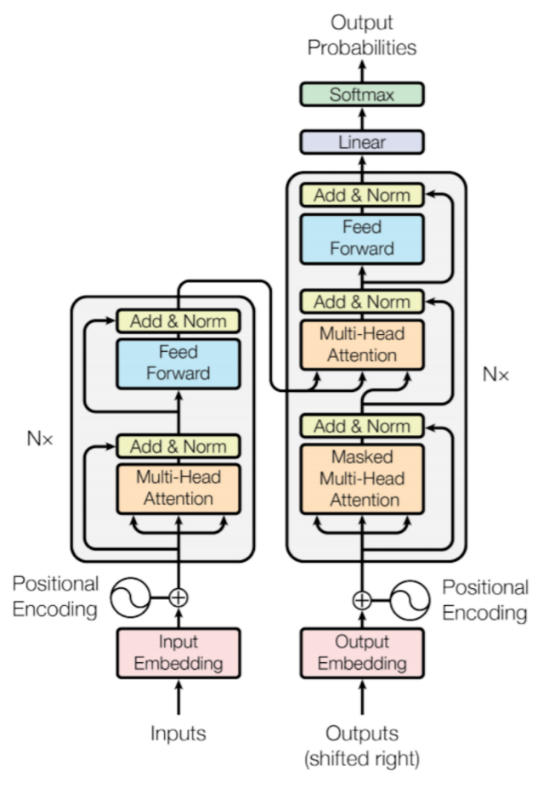
**7.2.2. Transformer Architecture**

The Transformer architecture has emerged as a significant breakthrough in natural language processing (NLP), revolutionizing the field by introducing a self-attention mechanism that captures word dependencies without traditional recurrent or convolutional structures. This groundbreaking approach allows the model to attend to all positions in the input sequence simultaneously, enabling efficient parallelization and effective handling of long-range dependencies. As a result, the Transformer has achieved remarkable success in various NLP tasks, including machine translation, text generation, and language understanding, surpassing previous state-of-the-art results.

At the core of the Transformer architecture is the self-attention mechanism, which fundamentally changes the way models process sequential data. By employing an encoder-decoder structure with multiple layers, each comprising a self-attention module and a position-wise feed-forward neural network, the Transformer enables the model to understand both global and local dependencies within the input sequence. This comprehensive understanding, combined with the ability to perform non-linear transformations between positions, empowers the Transformer to capture intricate linguistic patterns and relationships.

The parallelization-friendly design of the Transformer architecture has further contributed to its success. Unlike traditional sequential models, such as recurrent neural networks (RNNs), the Transformer can process the entire input sequence in parallel. This characteristic leverages the computational power of modern hardware, such as GPUs, leading to faster training and inference times, particularly for longer sequences. Moreover, the Transformer's capacity to learn from vast amounts of data has made it a preferred choice for NLP tasks, where large-scale datasets are often available.

In summary, the Transformer architecture has reshaped the NLP landscape by offering a powerful and efficient alternative to traditional sequence models. Its ability to capture word dependencies through self-attention, along with its parallelization-friendly design, has propelled it to achieve state-of-the-art results in various NLP tasks. With its exceptional performance, the Transformer continues to drive advancements in machine translation, text generation, and language understanding, and its impact on the field is likely to endure.

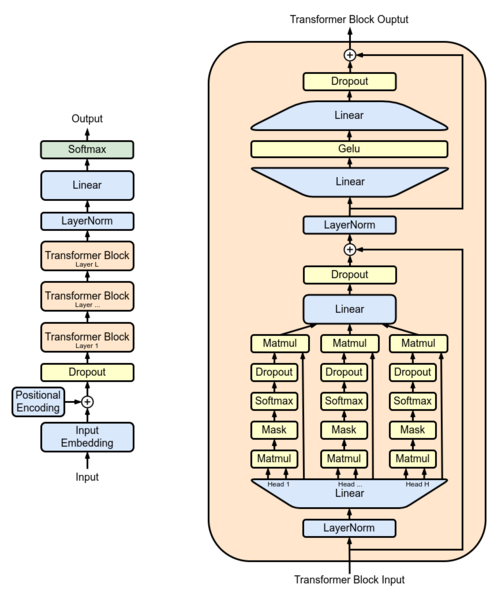


**7.2.3. Generative Pre-trained Transformer**

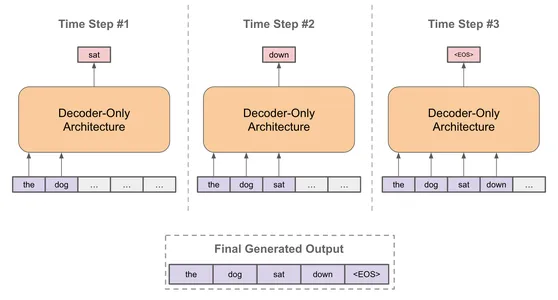
GPT is a new approach to pre-training language representations that achieves state-of-the-art results on eleven natural language processing tasks. GPT is characterized by its remarkable ability to generate coherent and contextually relevant text across a wide range of topics. With a transformer architecture, GPT-2 excels in capturing intricate patterns and relationships within data, making it a powerful tool for natural language processing tasks. One of its notable features is its extensive pre-training on diverse datasets, allowing it to grasp the nuances of language and context effectively. GPT has demonstrated impressive language generation capabilities, producing human-like text that spans multiple paragraphs. Its versatility extends to applications such as content creation, text completion, and language understanding. However, the power of GPT also comes with challenges related to ethical considerations, particularly regarding the potential misuse of its text generation capabilities. As researchers continue to explore and improve upon transformer-based models like GPT, they contribute significantly to the ongoing advancements in artificial intelligence and natural language understanding.

GPT-2 is a Transformer architecture that was notable for its size (1.5 billion parameters) on its release. The model is pretrained on a WebText dataset - text from 45 million website links. It largely follows the previous GPT architecture with some modifications:

* Layer normalization is moved to the input of each sub-block, similar to a pre-activation residual network and an additional layer normalization was added after the final self-attention block.
* A modified initialization which accounts for the accumulation on the residual path with model depth is used. Weights of residual layers are scaled at initialization by a factor of where is the number of residual layers.
* The vocabulary is expanded to 50,257. The context size is expanded from 512 to 1024 tokens and a larger batch size of 512 is used.



At its core, GPT-2 is an autoregressive model. It predicts the next word in a sequence based on the preceding words. This prediction process continues iteratively until the desired length of text is generated. GPT-2 uses a softmax function to estimate the probability distribution over the vocabulary for each word in the sequence.



**7.4.4. Apply NLP to project**

The simplest way to run a trained GPT-2 is to allow it to ramble on its own (which is technically called generating unconditional samples) – alternatively, we can give it a prompt to have it speak about a certain topicIn the rambling case, we can simply hand it the start token and have it start generating words (the trained model uses as its start token. Let’s call it instead). Essential parameters such as and can be adjusted to customize the generated output.

For prompted responses, modify the input text to include a specific prompt relevant to the project's context. This prompt guides the model's response, and parameters can be fine-tuned accordingly. Following this, generate the model's response to the given prompt and handle the output appropriately. This might involve displaying the generated text to users, saving it to a file, or integrating it into the project in a manner aligned with project objectives. It is crucial to ensure efficient handling of model loading and tokenization, especially in real-time scenarios, to optimize the application of GPT-2 for the project's specific needs.

# **Chapter 08: Evaluation**

## **8.1. Model Evaluation**

After training the model with the created dataset for 100 epochs and with the following set parameters: , , and we obtained the following results:

The overall loss of the model on the entire dataset is **0.252**, but during practical experiments, the observed loss is higher at **0.39**. According to the evaluation, the predicted time required for tasks in sentences generated by the model tends to be consistently overestimated, exceeding the actual time by more than 15 minutes. However, the model performs well in accurately determining the date and month of the tasks, achieving high precision for other attributes compatible with the dataset. Moreover, when introducing new tasks not present in the training dataset, there is a notable likelihood that the model struggles to predict the categories of those tasks. This suggests a potential limitation in the model's ability to generalize to novel tasks or categories not encountered during training. Fine-tuning on additional diverse data or adjusting the model architecture may be explored to enhance its adaptability to unseen tasks. Overall, the evaluation highlights areas of strength and potential improvement for the GPT-2 model in the context of task prediction and scheduling.

After incorporating the parsedatetime library, the program's capabilities have significantly improved, enhancing its ability to accurately extract the time details of tasks. The integration of this library has proven to be instrumental in refining the program's time extraction functionality, contributing to more precise identification and handling of task schedules.

## **8.2. Application Evaluation**

# **Chapter 09: Conclusion**

# **References**