國立臺灣大學電機資訊學院資訊工程學系
碩士論文

Department of Computer Science & Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

從需求到任務模型：以Transformer爲基礎的機器學習方法

From Requirements to Task Model: A Transformer-based Machine Learning Approach

許　恆

Heng Hsu

指導教授：李允中 博士

Advisor: Jonathan Lee, Ph.D.

中華民國112 年7 月

July, 2023

doi:10.6342/NTU202303644

國立臺灣大學碩士學位論文
口試委員會審定書

MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

從需求到任務模型：以 Transformer 為基礎的機器學習方法

From Requirements to Task Model: A Transformer-based Machine Learning Approach

本論文係許 恆君（學號 R10922107）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 112 年 7 月 27 日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Computer Science and Information Engineering on 27 July 2023 have examined a Master's thesis entitled above presented by HSU, HENG (student ID: R10922107) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

李允中	[illegible]	鄭有進
（指導教授 Advisor）		
徐國勛	[illegible]	

系主任/所長 Director: 洪士灝

誌謝

首先我要感謝我的指導教授李允中教授，這兩年以來的教導以及做研究的方法：尋找研究的主題、相關研究的探討、問題的發掘、以及解決問題的方法。再來，我要感謝台灣大學軟體工程研究室的成員，陳力聖、林辰臻、林怡伶、劉仁軒、張馨尹、梁峻瑞、錢怡君助理以及學弟妹等，在一些共同的專案中一同努力、互相合作、相處融洽。由於許多人的幫助，讓我得以完成此篇論文。

doi:10.6342/NTU202303644

摘要

現今軟體專案的開發通常都是從需求開始，先利用需求來產生使用案例規格書以及使用案例圖，接著再使用這兩種文件作爲基礎，進行後續軟體的設計、實作以及測試等一系列開發工作，最終得以將軟體產出。

而在這個過程中，除了最一開始的需求可以由一般使用者提出以外，其餘的環節往往都需要大量專業的軟體工程師參與才得以進行，這不但耗費大量的金錢與時間，也讓一般使用者難以自行開發軟體。

我們的研究專注於網路應用程式的開發，並提出了一種方法能讓一般使用者只需提供使用案例規格書以及使用案例圖作爲輸入，即可自動產生網路應用程式。這個方法可以被分成兩個部分，前半部分是將輸入轉換成任務模型，後半部分則是利用任務模型自動產生網路應用程式，本研究主要專注於前半段，因此稱我們開發的系統爲Task Model Generator。

使用者輸入使用案例規格書以及使用案例圖後，Task Model Generator會利用機器學習模型以及特定的演算法自動產生出一系列的任務模型，後續則可以利用這些任務模型自動構建出使用者介面，並產生網路應用程式。如此一來不僅能降低網路應用程式的開發門檻，也能大幅減少開發所需的金錢與時間。

關鍵詞 — 任務模型、機器學習、設計模式、使用者介面元件合成

doi:10.6342/NTU202303644

Abstracts

A software development process usually begins with the requirements, which are used to generate use case specifications and use case diagrams. These documents then serve as the foundation for designing, implementing, and testing the software. Eventually the software is produced.

Typically, in this process, only the requirements can be provided by common users, while other parts require the participation of skilled software engineers. This not only consumes a significant amount of time and money, but also makes it hard for common users to develop software on their own.

Our research focuses specifically on the development of web applications. We propose an approach that enables users to generate web applications by only providing use case specifications and use case diagram as input. The process can be divided into two parts: task model generation and web application generation. This research mainly focuses on the first part and introduces a system called "Task Model Generator."

Task Model Generator uses machine learning models and specific algorithms to generate a series of task models, which can be used to design and build the user

doi:10.6342/NTU202303644

interface later. This research not only simplifies the web application development for common users but also reduces the amount of time and money required for the development process.

Index terms — Task Model, Machine Learning, Design Pattern, UI Component Composition

doi:10.6342/NTU202303644

Contents

doi:10.6342/NTU202303644

doi:10.6342/NTU202303644

List of Figures

doi:10.6342/NTU202303644

doi:10.6342/NTU202303644

List of Tables

doi:10.6342/NTU202303644

Chapter 1

Introduction

How to reduce the threshold and resources required for software development has always been an important issue in software engineering. In the research field of web application development, with the contributions of a series of researches [6] [7] [10], we have enabled users to create web applications without the need of writing code by using our system. In our system, they only need to focus on designing the user interface and binding relevant services, and then our system can automatically generate the corresponding web application. Building upon this foundation, we aim to further minimize the user's manual operations and design efforts, allowing them to only provide the most essential information and the corresponding web application is generated automatically.

The essential information here refers to the requirements and their derived use case specifications along with a use case diagram. These documents provide the information about the user-system interactions and the relationships between inter-

doi:10.6342/NTU202303644

actions. In a software development process, software engineers use this information to design, implement and test the software. However, in the proposed approach of our research, once obtaining the essential information, we can use Task Model Generator to generate task models, which can be used in later steps to build the user interface and generate the web application. For more details about how to use task models to build a user interface and generate a web application can be referred to the research conducted by Liou [8]. The process of Task Model Generator can be divided into four steps:

1. Task Generation
2. Task Attribute Generation
3. Use Case CTT Generation
4. Main CTT Generation

This paper will focus on discussing the design and implementation details of the aforementioned steps. Since the process of implementation involves the use of Transformer-based machine learning models, we will also discuss how to use design patterns to interpret and analyze machine learning models. Additionally, our research will cover how to make Composition Engine support the generation of frontend code that uses Angular API and incorporates authentication mechanism.

The paper is structured as follows: Chapter 2 describes related research; Chapter 3 describes the design and implementation details of the steps executed in Task Model Generator; Chapter 4 describes the process of organizing machine learning

doi:10.6342/NTU202303644

design pattern and discusses its architecture; Chapter 5 describes how Composition Engine is modified to generate frontend code that supports Angular API and incorporates authentication mechanism; Chapter 6 summarizes the contributions of our research and discusses the problems we have encountered, as well as future work.

doi:10.6342/NTU202303644

Chapter 2

Related Work

In this chapter, we present the related work and highlight the characteristics of these works:

2.1 Figma

Figma [3] is an online drawing tool used for web application design on both computer and mobile platforms. Its main features include support for dynamic interactions between web pages, such as navigation and pop-up windows, as well as real-time collaboration.

Designers can use Figma to create UI components, plan the layout of web applications for different browsers and use their own designed UI components for web page design. Additionally, it provides a mechanism for setting navigation relationships between web pages, allowing designers to design the application's workflow

doi:10.6342/NTU202303644

and relationships between different web pages in a graphical manner.

Figma only requires simple actions such as dragging, clicking and entering basic parameters, making it accessible to users without programming skill. This user-friendly aspect makes it easy for common users to get started with this tool.

2.2 FreeMarker

FreeMarker [4] is a Java-based template engine that allows the combination of data models and templates to generate corresponding textual content. Templates here refer to textual content containing data binding positions. By binding data to these positions, specific content from the data models can be displayed in the output text.

With FreeMarker, users can focus on writing templates to present different data and FreeMarker will generate the corresponding textual content. Additionally, FreeMarker supports various advanced template features, such as conditional blocks, iterations, assignments and string operations. These features enable users to display data in different ways and support binding to more complex data structures.

2.3 Task Model

Task Model [5] is primarily used to represent how activities are performed to achieve user goals. It is commonly used in user interface design, where designers use it to represent and manipulate abstracted activities, thereby designing processes

doi:10.6342/NTU202303644

that achieve user goals.

The use of Task Model is typically driven by the need for supporting the implementation of corresponding systems. If we only provide developers with user stories or simple design prototypes, developers would have to make many design decisions on their own. However, developers often lack the relevant background to construct a complete interactive system. Using Task Model enables designers to represent designs in a more systematic way, providing a basis for implementation and ensuring that design details are not lost during the implementation process.

2.4 Concur Task Tree

Concur Task Tree (CTT) [2] is a meta-model used to visually represent task models while also supporting display in XML format. Using Concur Task Tree allows the interchange of task models between different design tools.

Its main features include focus on activities, hierarchical structure, graphical syntax, rich temporal operators, task allocation, objects and task attributes, etc. Additionally, with the emergence of collaborative applications due to the development of the internet, Concur Task Tree can also be used to represent the design of such applications and support applications where users have multiple actors.

doi:10.6342/NTU202303644

2.5 BertSum

BertSum [1] is an open-source, autoencoding, Transformer-based machine learning model, specifically designed for executing summarization tasks. It is derived from the BERT model. Apart from revising the original BERT's architecture, it also modifies the training schedule by separately optimizing the encoder and decoder. The encoder is fine-tuned twice for extractive summarization tasks and abstractive summarization tasks, enabling BertSum to adapt to different types of summarization tasks.

In terms of performance, BertSum achieved nearly the best results in summarization tasks at the time of its publication. Even though other new language models have surpassed BertSum's performance later, its open-source feature and lower system requirements still make it widely used for executing summarization tasks.

doi:10.6342/NTU202303644

Chapter 3

Task Model Generator

Task Model Generator utilizes use case specifications and a use case diagram to generate task models. The generated task models include Concur Task Trees (CTTs) for each use case as well as the whole project. Through these CTTs, we can analyze the relationships between different tasks to automatically build the user interface and generate the corresponding web application.

The process of Task Model Generator can be divided into four steps: Task Generation, Task Attribute Generation, Use Case Generation, and Main CTT Generation (See Figure 3.1). In this chapter, we will first introduce the definition and the format of the Task Model, followed by introductions to the design and the implementation details of each of the four steps.

doi:10.6342/NTU202303644

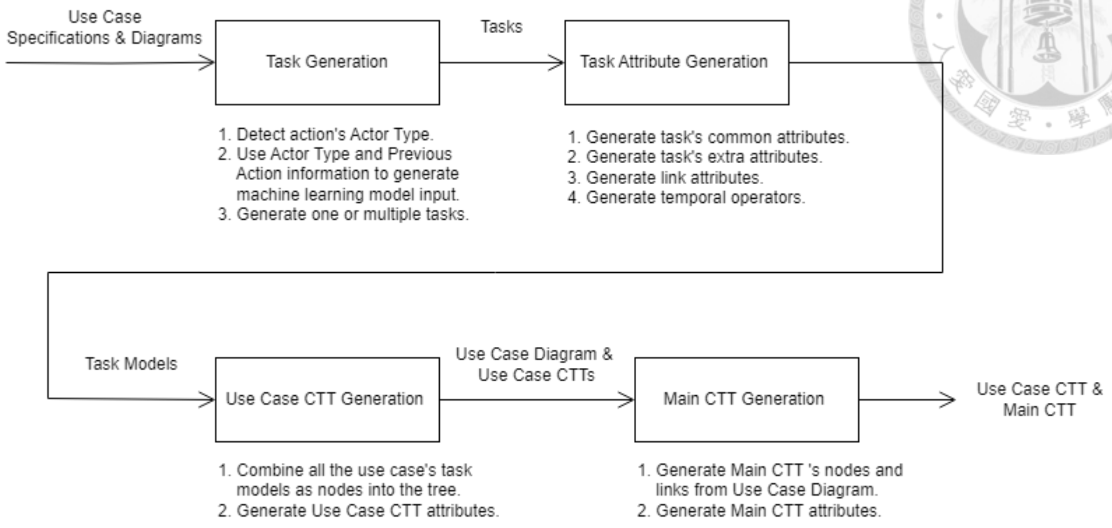

Use Case Specifications & Diagrams
Task Generation
Tasks
Task Attribute Generation
1. Detect action's Actor Type.
2. Use Actor Type and Previous Action information to generate machine learning model input.
3. Generate one or multiple tasks.
1. Generate task's common attributes.
2. Generate task's extra attributes.
3. Generate link attributes.
4. Generate temporal operators.
Task Models
Use Case CTT Generation
Use Case Diagram & Use Case CTTs
Main CTT Generation
Use Case CTT & Main CTT
1. Combine all the use case's task models as nodes into the tree.
2. Generate Use Case CTT attributes.
1. Generate Main CTT 's nodes and links from Use Case Diagram.
2. Generate Main CTT attributes.

Figure 3.1: Process of Task Model Generator

3.1 Task Model Definition

In Task Model Generator, task models are represented in the form of CTT, and the generated CTTs are represented in XML format. The XML representation of a CTT can be divided into two parts: the header and the body. The header contains the information about the whole CTT, while the body contains the information about the tasks in the CTT and the links between the tasks in the CTT.

In the header, there is only one section: description. The description section contains the following information:

id: CTT's ID.

name: CTT's name.

overview: CTT's overview, typically the introduction of corresponding use case or

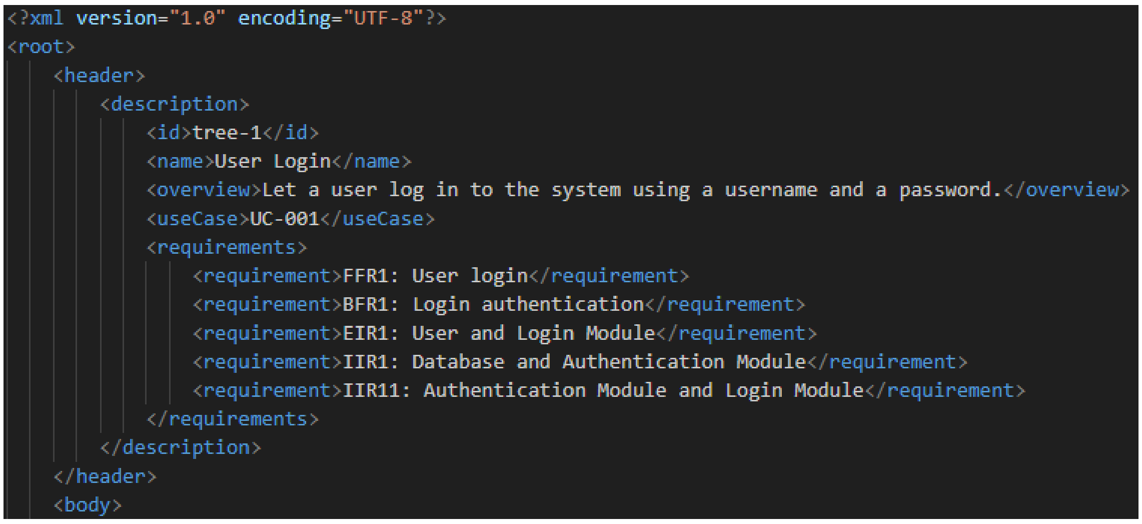
doi:10.6342/NTU202303644

project based on the type of the CTT.

useCase: CTT's corresponding use case.

requirements: CTT's corresponding requirements.

If a CTT is a use case CTT, its header will contain all the information mentioned above (See Figure 3.2). If it is a main CTT, its header will only contain id, name, and overview.



```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <header>
        <description>
            <id>tree-1</id>
            <name>User Login</name>
            <overview>Let a user log in to the system using a username and a password.</overview>
            <useCase>UC-001</useCase>
            <requirements>
                <requirement>FFR1: User login</requirement>
                <requirement>BFR1: Login authentication</requirement>
                <requirement>EIR1: User and Login Module</requirement>
                <requirement>IIR1: Database and Authentication Module</requirement>
                <requirement>IIR11: Authentication Module and Login Module</requirement>
            </requirements>
        </description>
    </header>
    <body>
```

Figure 3.2: Example of CTT's header

In the body, there are two sections: nodes and links. The nodes section contains information about all the tasks in the CTT, while the links section contains information about the relationships between tasks in the CTT. All tasks in the nodes section contain the following information:

id: task's ID.

doi:10.6342/NTU202303644

name: task's name and content.

iterative: whether this task can be executed repeatedly or not.

optional: whether this task can be executed optionally or not.

category: task's category.

task type: task's task type based on its category.

childrenIds: the IDs of this task's child tasks.

Based on our definition, task's category can be divided into the following four types:

Interaction: tasks involving user-system interactions and executed by the user.

System: tasks executed by the system.

Abstraction: tasks related to abstraction.

User: tasks executed by the user, and are unrelated to the system, with only one relevant task type: "User."

The corresponding task types for task's categories (excluding "User") are shown in Table 3.1, Table 3.2 and Table 3.3. There is also an example of the nodes section in the body of a use case CTT (See Figure 3.3).

In the body, apart from the nodes section, there's also the links section. The links section contains operators, where each operator represents a link connecting two tasks. An operator contains the following information:

doi:10.6342/NTU202303644

Category	Task Type	Description
Interaction	InteractionTaskGroup	Task that contains subtasks all from “Interaction” category.
	Input	User enters data.
	SelectFromFixedList	User chooses data from a list of predefined data.
	SelectFromDynamicallyAcquiredList	User selects data from a list of dynamic data.
	Control	User triggers activities in the system, such as clicking a button.
	SelectFromVisualizedInfo	User selects data from information displayed on the screen.
	RespAlert	User responds to system alerts.

Table 3.1: Task Types of “Interaction” Category

id: link's ID.

type: the temporal operator type of the link, representing the relationship between the source task and the target task.

sourceId: the source task's ID of the link.

targetId: the target task's ID of the link.

Temporal operator can be further divided into eight types, as shown in Table 3.4.

doi:10.6342/NTU202303644

Category	Task Type	Description
System	SystemTaskGroup	Task that contains subtasks all from “System” category.
	Checking	System displays messages for user confirmation.
	ErrorMessage	System displays error or warning messages.
	Feedback	System displays the progress of a task.
	FilterInfo	System filters data.
	InputValidation	System validates user input.
	VisualizeFixedValue	System displays fixed data.
	VisualizeDynamicallyAcquiredValue	System displays dynamic data.
	Service	System triggers backend services.

Table 3.2: Task Types of “System” Category

For use case CTTs, there's an example of the links section in the body (See Figure 3.4). There's also a visualized representation of the CTT displayed in Figure 3.4 (See Figure 3.5).

Task Model Generator has two inputs: use case specifications and a use case diagram. The reason for not directly using requirements as input is that the information contained solely in requirements is not sufficient to build a user interface. The requirements need to be further expanded into use case specifications and a use case diagram for more comprehensive amount of information. We currently have

doi:10.6342/NTU202303644

Category	Task Type	Description
Abstraction	TaskGroup	Task that contains subtasks from different categories.
	IncludeTask	Task that refers to another CTT.

Table 3.3: Task Types of “Abstraction” Category

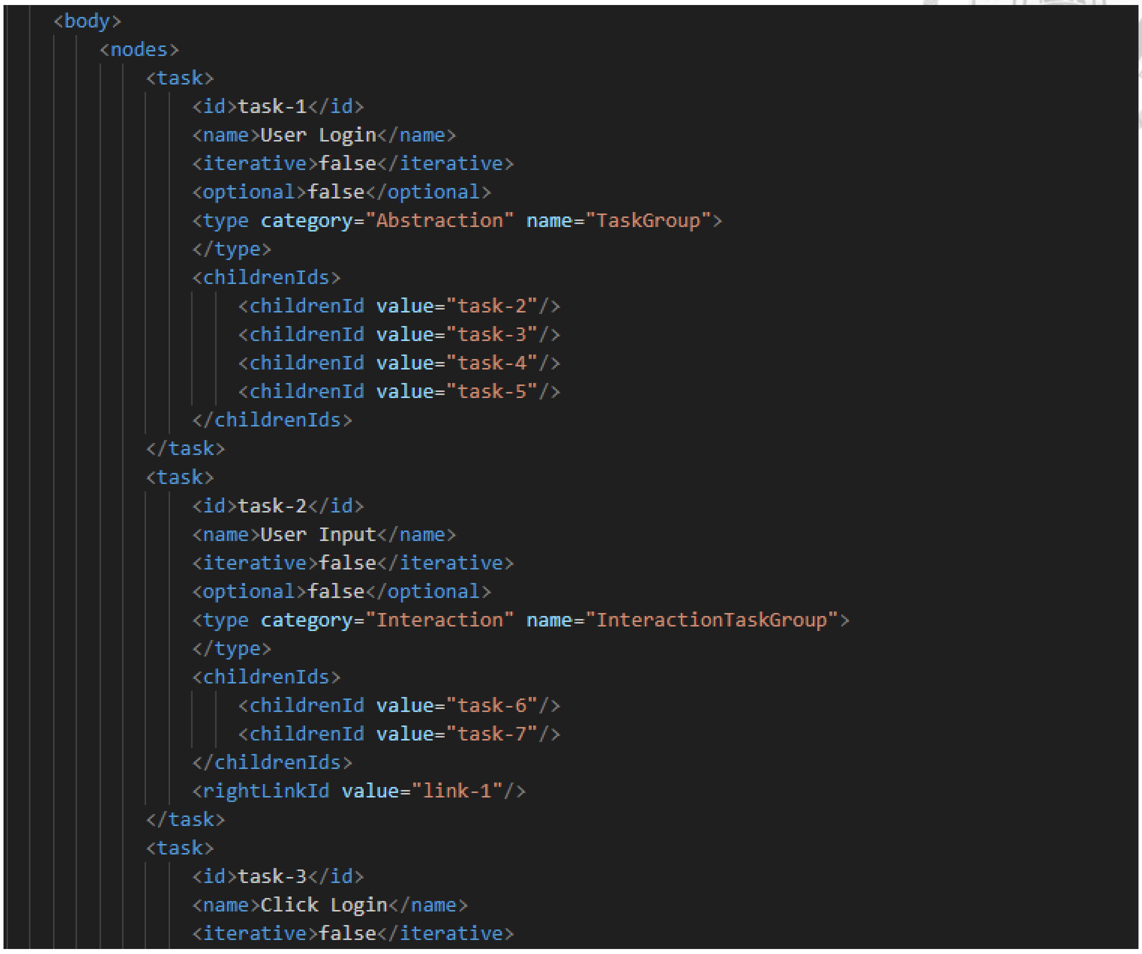
set some standards for use case specification, for example: the use case specification must include fields such as Use Case ID, Use Case Name, Action ID, Actor Type, Actor, and Action. Additionally, there are specific guidelines for describing an action, so the content of a use case specification is formalized (See Figure 3.7). There's an example input for use case diagram (See Figure 3.8). Note that when inputting a use case diagram into our system, it needs to be converted into a format that our system can interpret.

3.2 Actor Type Detection

Through the process of Task Model Generator, we first generate tasks, where tasks here refer to what the actor does to achieve the goal without including the information required to construct the task model. Following the order of actions in the use case specifications, we use machine learning models to generate tasks from the actions. The process can be divided into three steps:

1. Detect the actor type of the current action in the use case.

doi:10.6342/NTU202303644



```
<body>
    <nodes>
        <task>
            <id>task-1</id>
            <name>User Login</name>
            <iterative>false</iterative>
            <optional>false</optional>
            <type category="Abstraction" name="TaskGroup">
            </type>
            <childrenIds>
                <childrenId value="task-2"/>
                <childrenId value="task-3"/>
                <childrenId value="task-4"/>
                <childrenId value="task-5"/>
            </childrenIds>
        </task>
        <task>
            <id>task-2</id>
            <name>User Input</name>
            <iterative>false</iterative>
            <optional>false</optional>
            <type category="Interaction" name="InteractionTaskGroup">
            </type>
            <childrenIds>
                <childrenId value="task-6"/>
                <childrenId value="task-7"/>
            </childrenIds>
            <rightLinkId value="link-1"/>
        </task>
        <task>
            <id>task-3</id>
            <name>Click Login</name>
            <iterative>false</iterative>
```

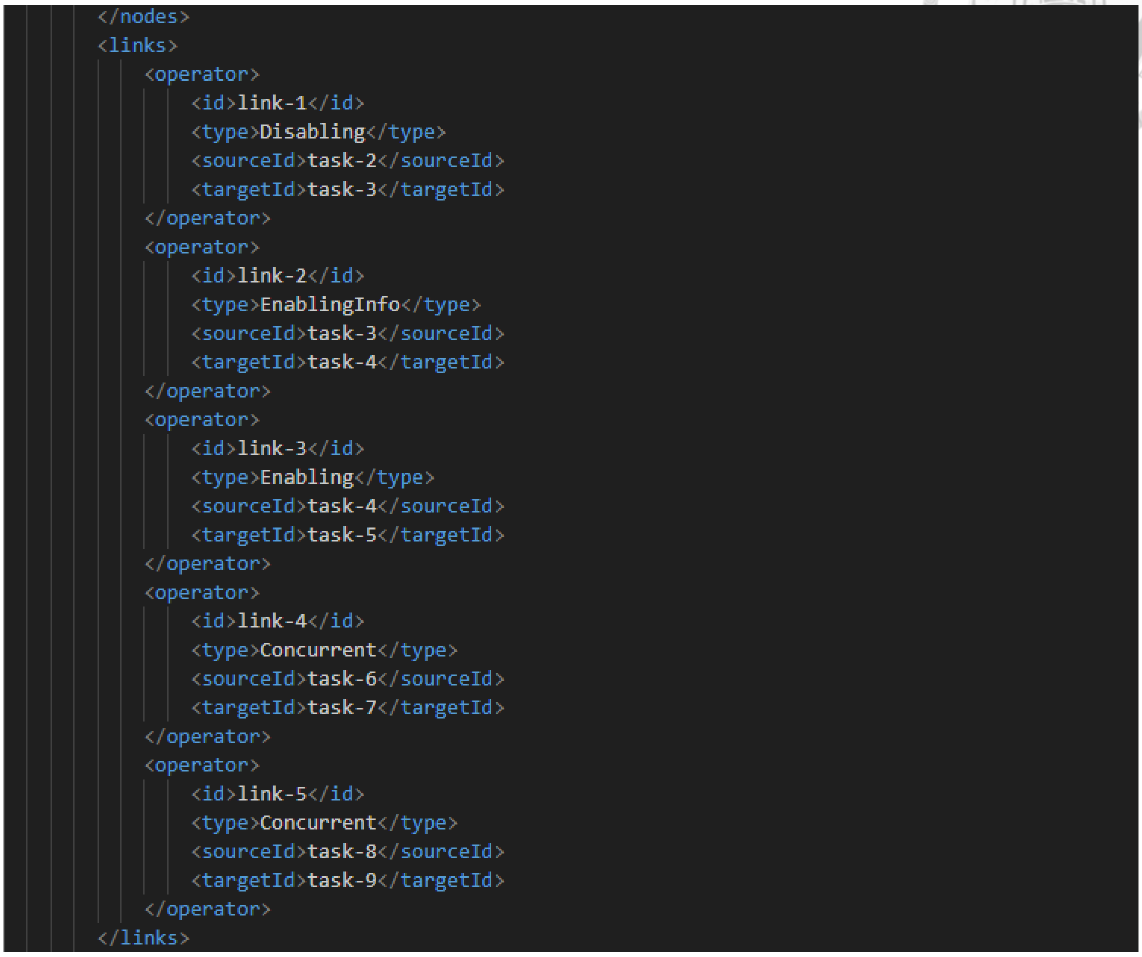
Figure 3.3: Example of CTT body's nodes section

2. Retrieve the corresponding few-shot learning examples based on the actor type, and then combine these examples with information about the actions executed prior to the current action to generate input for the machine learning model.

3. Use the machine learning model input to generate tasks for the current action. This may result in one or multiple tasks being generated.

The step of detecting the actor type of the current action in the use case primarily

doi:10.6342/NTU202303644



```
        </nodes>
        <links>
            <operator>
                <id>link-1</id>
                <type>Disabling</type>
                <sourceId>task-2</sourceId>
                <targetId>task-3</targetId>
            </operator>
            <operator>
                <id>link-2</id>
                <type>EnablingInfo</type>
                <sourceId>task-3</sourceId>
                <targetId>task-4</targetId>
            </operator>
            <operator>
                <id>link-3</id>
                <type>Enabling</type>
                <sourceId>task-4</sourceId>
                <targetId>task-5</targetId>
            </operator>
            <operator>
                <id>link-4</id>
                <type>Concurrent</type>
                <sourceId>task-6</sourceId>
                <targetId>task-7</targetId>
            </operator>
            <operator>
                <id>link-5</id>
                <type>Concurrent</type>
                <sourceId>task-8</sourceId>
                <targetId>task-9</targetId>
            </operator>
        </links>
```

Figure 3.4: Example of CTT body's links section

addresses the input length limitation of the GPT machine learning models. Since GPT models use few-shot learning technique, they only require task descriptions and a small number of examples as inputs. The model can learn to execute the task based on these inputs, without the need for extensive training data and doing gradient descent as in traditional machine learning models.

Using the few-shot learning technique makes GPT models highly dependent on

doi:10.6342/NTU202303644

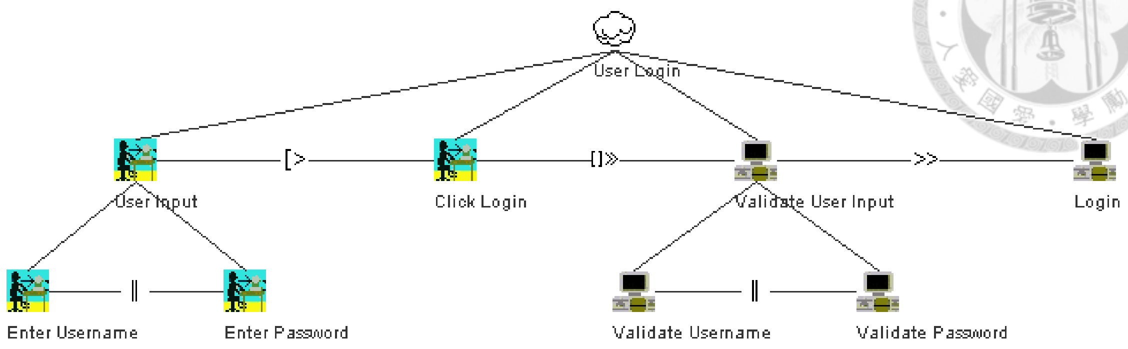
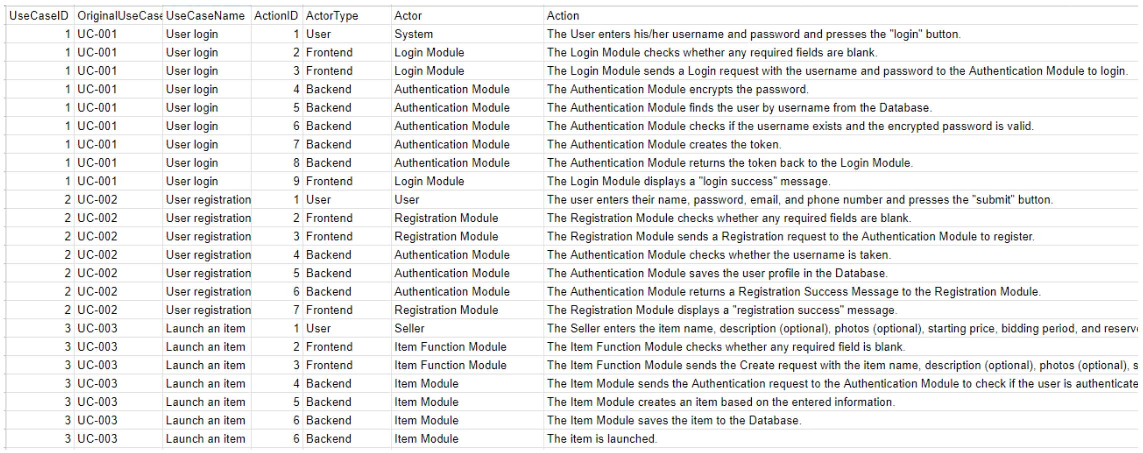


Figure 3.5: Example of Visualized CTT



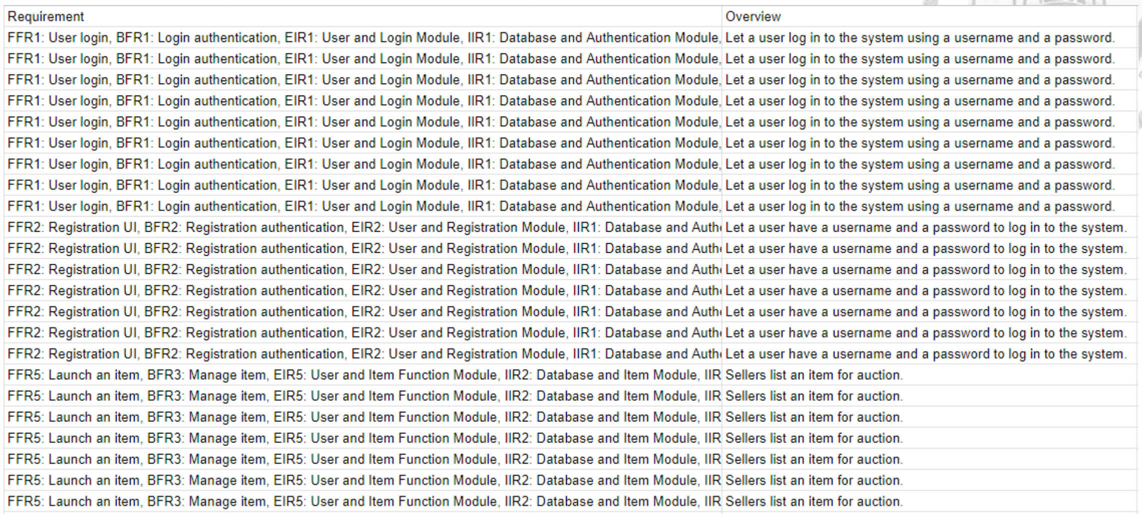
UseCaseID	OriginalUseCase	UseCaseName	ActionID	ActorType	Actor	Action
1	UC-001	User login	1	User	System	The User enters his/her username and password and presses the "login" button.
1	UC-001	User login	2	Frontend	Login Module	The Login Module checks whether any required fields are blank.
1	UC-001	User login	3	Frontend	Login Module	The Login Module sends a Login request with the username and password to the Authentication Module to login.
1	UC-001	User login	4	Backend	Authentication Module	The Authentication Module encrypts the password.
1	UC-001	User login	5	Backend	Authentication Module	The Authentication Module finds the user by username from the Database.
1	UC-001	User login	6	Backend	Authentication Module	The Authentication Module checks if the username exists and the encrypted password is valid.
1	UC-001	User login	7	Backend	Authentication Module	The Authentication Module creates the token.
1	UC-001	User login	8	Backend	Authentication Module	The Authentication Module returns the token back to the Login Module.
1	UC-001	User login	9	Frontend	Login Module	The Login Module displays a "login success" message.
2	UC-002	User registration	1	User	User	The user enters their name, password, email, and phone number and presses the "submit" button.
2	UC-002	User registration	2	Frontend	Registration Module	The Registration Module checks whether any required fields are blank.
2	UC-002	User registration	3	Frontend	Registration Module	The Registration Module sends a Registration request to the Authentication Module to register.
2	UC-002	User registration	4	Backend	Authentication Module	The Authentication Module checks whether the username is taken.
2	UC-002	User registration	5	Backend	Authentication Module	The Authentication Module saves the user profile in the Database.
2	UC-002	User registration	6	Backend	Authentication Module	The Authentication Module returns a Registration Success Message to the Registration Module.
2	UC-002	User registration	7	Frontend	Registration Module	The Registration Module displays a "registration success" message.
3	UC-003	Launch an item	1	User	Seller	The Seller enters the item name, description (optional), photos (optional), starting price, bidding period, and reserv
3	UC-003	Launch an item	2	Frontend	Item Function Module	The Item Function Module checks whether any required field is blank.
3	UC-003	Launch an item	3	Frontend	Item Function Module	The Item Function Module sends the Create request with the item name, description (optional), photos (optional), s
3	UC-003	Launch an item	4	Backend	Item Module	The Item Module sends the Authentication request to the Authentication Module to check if the user is authenticate
3	UC-003	Launch an item	5	Backend	Item Module	The Item Module creates an item based on the entered information.
3	UC-003	Launch an item	6	Backend	Item Module	The Item Module saves the item to the Database.
3	UC-003	Launch an item	6	Backend	Item Module	The item is launched.

Figure 3.6: Example of Input Use Case Specification (Left Part)

the task descriptions and the examples in the input. Due to the input length limitation (for instance, 2048 tokens in GPT-NeoX), the number of examples per input cannot exceed around twenty. Achieving good learning performance with this small number of examples is very challenging.

Hence, we have generated dedicated few-shot learning examples for different actor types and the execution order of the current action in the use case (See Table 3.5). This approach indirectly increases the total number of examples we can input,

doi:10.6342/NTU202303644



Requirement	Overview
FFR1: User login, BFR1: Login authentication, EIR1: User and Login Module, IIR1: Database and Authentication Module,	Let a user log in to the system using a username and a password.
FFR1: User login, BFR1: Login authentication, EIR1: User and Login Module, IIR1: Database and Authentication Module,	Let a user log in to the system using a username and a password.
FFR1: User login, BFR1: Login authentication, EIR1: User and Login Module, IIR1: Database and Authentication Module,	Let a user log in to the system using a username and a password.
FFR1: User login, BFR1: Login authentication, EIR1: User and Login Module, IIR1: Database and Authentication Module,	Let a user log in to the system using a username and a password.
FFR1: User login, BFR1: Login authentication, EIR1: User and Login Module, IIR1: Database and Authentication Module,	Let a user log in to the system using a username and a password.
FFR1: User login, BFR1: Login authentication, EIR1: User and Login Module, IIR1: Database and Authentication Module,	Let a user log in to the system using a username and a password.
FFR1: User login, BFR1: Login authentication, EIR1: User and Login Module, IIR1: Database and Authentication Module,	Let a user log in to the system using a username and a password.
FFR1: User login, BFR1: Login authentication, EIR1: User and Login Module, IIR1: Database and Authentication Module,	Let a user log in to the system using a username and a password.
FFR1: User login, BFR1: Login authentication, EIR1: User and Login Module, IIR1: Database and Authentication Module,	Let a user log in to the system using a username and a password.
FFR2: Registration UI, BFR2: Registration authentication, EIR2: User and Registration Module, IIR1: Database and Auth	Let a user have a username and a password to log in to the system.
FFR2: Registration UI, BFR2: Registration authentication, EIR2: User and Registration Module, IIR1: Database and Auth	Let a user have a username and a password to log in to the system.
FFR2: Registration UI, BFR2: Registration authentication, EIR2: User and Registration Module, IIR1: Database and Auth	Let a user have a username and a password to log in to the system.
FFR2: Registration UI, BFR2: Registration authentication, EIR2: User and Registration Module, IIR1: Database and Auth	Let a user have a username and a password to log in to the system.
FFR2: Registration UI, BFR2: Registration authentication, EIR2: User and Registration Module, IIR1: Database and Auth	Let a user have a username and a password to log in to the system.
FFR2: Registration UI, BFR2: Registration authentication, EIR2: User and Registration Module, IIR1: Database and Auth	Let a user have a username and a password to log in to the system.
FFR2: Registration UI, BFR2: Registration authentication, EIR2: User and Registration Module, IIR1: Database and Auth	Let a user have a username and a password to log in to the system.
FFR5: Launch an item, BFR3: Manage item, EIR5: User and Item Function Module, IIR2: Database and Item Module, IIR	Sellers list an item for auction.
FFR5: Launch an item, BFR3: Manage item, EIR5: User and Item Function Module, IIR2: Database and Item Module, IIR	Sellers list an item for auction.
FFR5: Launch an item, BFR3: Manage item, EIR5: User and Item Function Module, IIR2: Database and Item Module, IIR	Sellers list an item for auction.
FFR5: Launch an item, BFR3: Manage item, EIR5: User and Item Function Module, IIR2: Database and Item Module, IIR	Sellers list an item for auction.
FFR5: Launch an item, BFR3: Manage item, EIR5: User and Item Function Module, IIR2: Database and Item Module, IIR	Sellers list an item for auction.
FFR5: Launch an item, BFR3: Manage item, EIR5: User and Item Function Module, IIR2: Database and Item Module, IIR	Sellers list an item for auction.
FFR5: Launch an item, BFR3: Manage item, EIR5: User and Item Function Module, IIR2: Database and Item Module, IIR	Sellers list an item for auction.

Figure 3.7: Example of Input Use Case Specification (Right Part)

enhancing the model's performance. The example input and output for this step are shown as follows:

Input: ActorType: User
ActionID: 1

Output: few-shot learning examples corresponding to ActorType: User and ActionID: 1

3.3 Task Generation

After detecting the actor type and the execution order of the current action, the next step is to generate corresponding machine learning model input and use the machine learning model to generate tasks. In this step, we have experimented with

doi:10.6342/NTU202303644

two different machine learning models: GPT-NeoX and BertSum. The inputs for these two models are essentially the same, with the difference being that GPT-NeoX requires additional few-shot learning examples. The information contained in the model input is as follows:

Few-shot Learning Examples: the task examples provided to the model for few-shot learning.

Actor Type: the actor's type, which can be "User" or "Frontend." As the actor type determines the task types that the actor can execute, providing this information enhances the model's accuracy.

Previous Action: the actions executed before the current action. Since the content of later actions may simplify details already mentioned in previous actions, incorporating information from previous actions helps the model capture the precise semantics of the current action.

Current Action: the action for which tasks are being generated.

When generating tasks, we first used GPT-NeoX. GPT-NeoX is a Transformer-based GPT model, where the Transformer architecture is designed for NLP tasks. The GPT models are good at text generation tasks such as generating articles or answering questions. Popular models like Chat-GPT and GPT-4 also belong to GPT models. A distinguishing feature of GPT-NeoX is that both its model and training data are open-source, and it has performance comparable to GPT-3, with a massive parameter count of up to 20 billion, making it one of the most parameter-rich open-source GPT models available.

doi:10.6342/NTU202303644

Before using GPT-NeoX for task generation, we execute the actor type detection as mentioned in the previous section to retrieve corresponding few-shot learning examples, which are necessary to generate a complete model input.

The model's hyper parameter setting is as follows:

- temperature: 0.95
- max length: 2048
- do sample: true

We used a test dataset containing 467 actions for our testing. Based on the test results (See Table 3.6) and the analysis of the generated tasks, the performance is not bad; however, we have encountered some issues. The most significant concern is the model's lack of strong generalization ability, causing it to heavily rely on the few-shot examples we provided, resulting in overfitting. When the tasks differ significantly from the provided examples, the accuracy of the output drops dramatically. One technically feasible approach to improve this situation is by adjusting the few-shot learning examples. However, due to the limitation of a maximum input length of 2048 tokens, adding new examples would require the removal of old ones. Nevertheless, this approach does not really solve this issue. Since we were unable to find a more powerful open-source GPT model, we attempted to use BertSum, a model specifically designed for summarization tasks.

Although our goal is to generate tasks, from the perspective of task types in the field of NLP, this is a kind of summarization task. We summarize the task from

doi:10.6342/NTU202303644

the current action and a series of information. BertSum is a model designed for summarization tasks. It is a Transformer-based BERT model. However, unlike the original BERT model, its training schedule and architecture are adjusted to optimize the model for executing summarization tasks.

Among the actions in our dataset, 70% are used for training, totaling 3259 actions; 20% are used for validation, totaling 926 actions; and 10% are used for testing, totaling 467 actions. This testing subset is also used as the test dataset for evaluating GPT-NeoX. The model's hyper parameter setting is configured according to the optimal configuration mentioned in the original paper [9], as shown as follows:

- batch size: 300
- accum count: 5
- dec dropout: 0.1
- learning rate: 0.05
- seed: 777
- sep optim: false
- train steps: 200000
- use bert emb: true
- use interval: true
- warnup steps: 8000

doi:10.6342/NTU202303644

- max pos: 512
- enc hidden size: 512
- enc layers: 6
- enc ff size: 2048
- enc dropout: 0.1
- dec layer: 6
- dec hidden size: 512
- dec ff size: 2048
- encoder: baseline
- task: abs

The test results are presented below (See Table 3.6). Considering both the test results and the analysis of the generated tasks, BertSum's performance is worse than that of GPT-NeoX. The proportion of incomplete sentence outputs is higher in BertSum than in GPT-NeoX. We believe there are three potential causes for this phenomenon: first, the training dataset is insufficient; second, the hyper parameter setting could be incorrect; and third, the model may have limited abilities.

After comparing with the original paper, we believe that the dataset size is the most significant factor. Because the dataset size used in the original paper is around 200,000, whereas our dataset size is only around 5,000, which is only 1/40 of the

doi:10.6342/NTU202303644

original paper's dataset size. This results in insufficient model training. Our hyper parameter setting is based on the optimal configuration in the original paper, but the models trained in the original paper performed much better in summarization tasks compared to ours. This indicates that we have not reached the model's upper bound of performance. If we can continue to increase the dataset size in the future, the model's performance can be improved more. The example input and output for this step are shown as follows:

Input: ActorType: User

ActionID: 1

Actions with an ActionID less than 1: None

Action: The User enters his/her username and password and presses the "Login" button.

Output: 1. Enter username.

2. Enter password.

3. Press Login.

3.4 Task Attribute Generation

After generating the tasks, we proceed to generate task attributes to construct complete task models that can be combined into a CTT. The attributes to be generated can be categorized into two types: first, the common attributes that are shared by every type of task; and second, extra attributes that are specific to certain types

doi:10.6342/NTU202303644

of tasks. After generating these two types of attributes, we will also generate link attributes. Among the link attributes, the most crucial one is the temporal operator, which represents the relationship between two tasks.

The following are the common attributes and the order in which they are generated:

1. name
2. id
3. childrenIds
4. category
5. task type
6. iterative
7. optional

These attributes have already been explained in 3.1. Here, we are focusing on how to generate them:

name: has already been generated in task generation step using a machine learning model.

id: can be generated using a simple algorithm.

childrenIds: can be generated using a simple algorithm.

doi:10.6342/NTU202303644

category: can be generated using a machine learning model.

task type: can be generated using a machine learning model.

iterative: can be generated using a machine learning model.

optional: can be generated using a machine learning model.

The example input and output for each of these attributes, which use a machine learning model to generate, are shown as follows:

Generate "category":

Input: Task: Enter username.
Actor Type: User

Output: Category: Interaction

Generate "task type":

Input: Task: Enter username.
Actor Type: User
Category: Interaction

Output: Task Type: Input

Generate "optional":

Input: Task: Enter username.

Output: Optional: False

doi:10.6342/NTU202303644

Input: Task: Enter description (optional).

Output: Optional: True

Generate "iterative" :

Input: Task: Enter username.

Output: Iterative: False

Input: Task: Click Update (iterative).

Output: Iterative: True

After generating common attributes, the next step is to generate extra attributes. Not all task types require the generation of extra attributes. The task types that require extra attributes can be referred to Table 3.7. These extra attributes are also generated using different machine learning models. The example input and output for each of these extra attributes, which use a machine learning model to generate, are shown as follows:
Generate "inputType" :

Input: Task: Enter username.

Output: Input Type: text

Generate "options" :

doi:10.6342/NTU202303644

Input: Task: Select gender.

Action: Select a gender from male and female.

Output: Options: male, female

Generate “visualizedTaskID”:

Input: Previous Task:

1. Get item list.
2. Display item list.

Current Task: Select item.

Output: Visualized Task ID: 2

Generate “alertTaskID”:

Input: Previous Task:

1. Click Update.
2. Display Are you sure to update? message.

Current Task: Click Confirm.

Output: Alert Task ID: 2

Generate “message”:

Input: Previous Task:

1. Click Login.
2. Display Login success message.

Current Task: Confirm Login info.

doi:10.6342/NTU202303644

Output: Message: Login success.

Input: Previous Task:

1. Click Login.
2. Display Username or password is wrong message.

Current Task: Show Login error.

Output: Message: Username or password is wrong.

Generate “serviceTaskID”:

Input: Previous Task:

1. Get launched item list.
2. Display launched item list.
3. Click Filter.

Current Task: Filter out Ongoing item from get launched item list.

Output: Service Task ID: 1

Generate “validatorType”:

Input: Task: Validate username is blank.

Output: Validator Type: Required

Generate “value”:

Input: Task: Display lab’s name Selab.

doi:10.6342/NTU202303644

Output: Value: Selab

After generating both common attributes and extra attributes, the next step is to generate link attributes. The following are the link attributes and the order in which they are generated:

1. id
2. sourceId
3. targetId
4. type (temporal operator)

These attributes have already been explained in 3.1. In this section, we will focus on the process of generating them. Besides the "type" (temporal operator), all the other attributes can be generated using algorithms. However, the temporal operator requires a machine learning model to generate, as it is a kind of classification task. The types of temporal operators can be found in 3.1. Once we have the task name, the actor type, and the task type information of the linked tasks, we can input this information into the model to generate corresponding temporal operator. The example input and output for temporal operator are shown as follows:

Input: Left Task: Click Login.
Left Actor Type: User
Left Task Type: Control
Right Task: Login.

doi:10.6342/NTU202303644

Right Actor Type: Frontend

Right Task Type: Service

Output: Temporal Operator: Enabling

After generating the link attributes, all the required task attributes have been generated. The next step is to generate use case CTTs.

3.5 Use Case CTT Generation

In this step, we combine the tasks and task attributes generated in the previous steps, which belong to the same use case, to generate the corresponding use case CTT. The steps for generating use case CTT are as follows:

1. Follow the order of actions in the use case specification to sequentially read and process the task models (including tasks and task attributes) generated from each action.
2. Generate tasks of “InteractionTaskGroup” task type and “SystemTaskGroup” task type, and then assign their respective child tasks.
3. If there are task models included from other use cases, generate corresponding abstraction category tasks and create the links between the original task and the newly generated abstraction category tasks, based on the “include” relationship.

doi:10.6342/NTU202303644

4. After processing all actions' task models, generate the header's information for the CTT based on the use case specification.

5. Output the use case CTT in XML format.

In the second step, there are two scenarios where the generation of tasks of "InteractionTaskGroup" task type or "SystemTaskGroup" task type is needed. The first scenario is when multiple task models are from the same action and share the same task type. In this case, a task of either "InteractionTaskGroup" task type or "SystemTaskGroup" task type is generated, and the task models are assigned as child tasks of this task.

The second scenario is when all actions' task models are processed for a use case, a task of "TaskGroup" task type representing the entire use case is created, and all actions' task models are assigned as child tasks of this task.

The example input and output for this step are shown as follows. There's also an example of generated use case CTT in XML format (See Figure 3.9).

Input: use case specifications, previously generated task models

Output: use case CTTs in XML format

3.6 Main CTT Generation

After generating the use case CTTs, the final step is to generate the main CTT. The data source for the main CTT is the use case diagram, which provides the

doi:10.6342/NTU202303644

relationships between use cases that are used to generate the corresponding main CTT. The steps for generating the main CTT are as follows:

1. Based on the use case diagram, identify a use case that does not extend from another use case.

2. Collect use cases that satisfy one of the following conditions: use cases that extend from the use case identified in step 1, or use cases whose pre-conditions are fulfilled after the completion of the use case identified in step 1.

3. Establish the corresponding relationships of the use cases collected in step 2 in the main CTT based on their connections to other use cases.

4. Repeat step 2 and step 3 until all use cases have been processed.

5. Output the main CTT in XML format.

The example input and output for this step are shown as follows. There's also an example of generated main CTT in XML format (See Figure 3.10).

Input: use case diagram

Output: main CTT in XML format

After completing this step, we have successfully generated all the CTTs. We can now continue to use these CTTs to generate corresponding web application. Detailed information about how to use CTTs to build user interface and generate web application automatically is provided in Liou's research [8].

doi:10.6342/NTU202303644

Temporal Operator	Description
Enabling	The task on the right (target) needs the task on the left (source) to complete before it can start.
Choice	Either task on both sides can start, but not both.
Enabling with information passing	The task on the right (target) needs the task on the left (source) to complete before it can start, and the output of the left (source) task is used as input for the right (target) task.
Concurrent tasks	Tasks on both sides can be executed in any order.
Concurrent communicating tasks	Tasks on both sides can exchange messages while executing simultaneously.
Task independence	Tasks on both sides can be executed in any order, but once one starts, the other must wait for it to complete.
Disabling	The task on the right (target) disables the task on the left (source) when it starts.
Suspend-Resume	The task on the left (source) is suspended by the task on the right (target). After the right task finishes, the left (source) task can resume.

Table 3.4: Types of Temporal Operator

doi:10.6342/NTU202303644

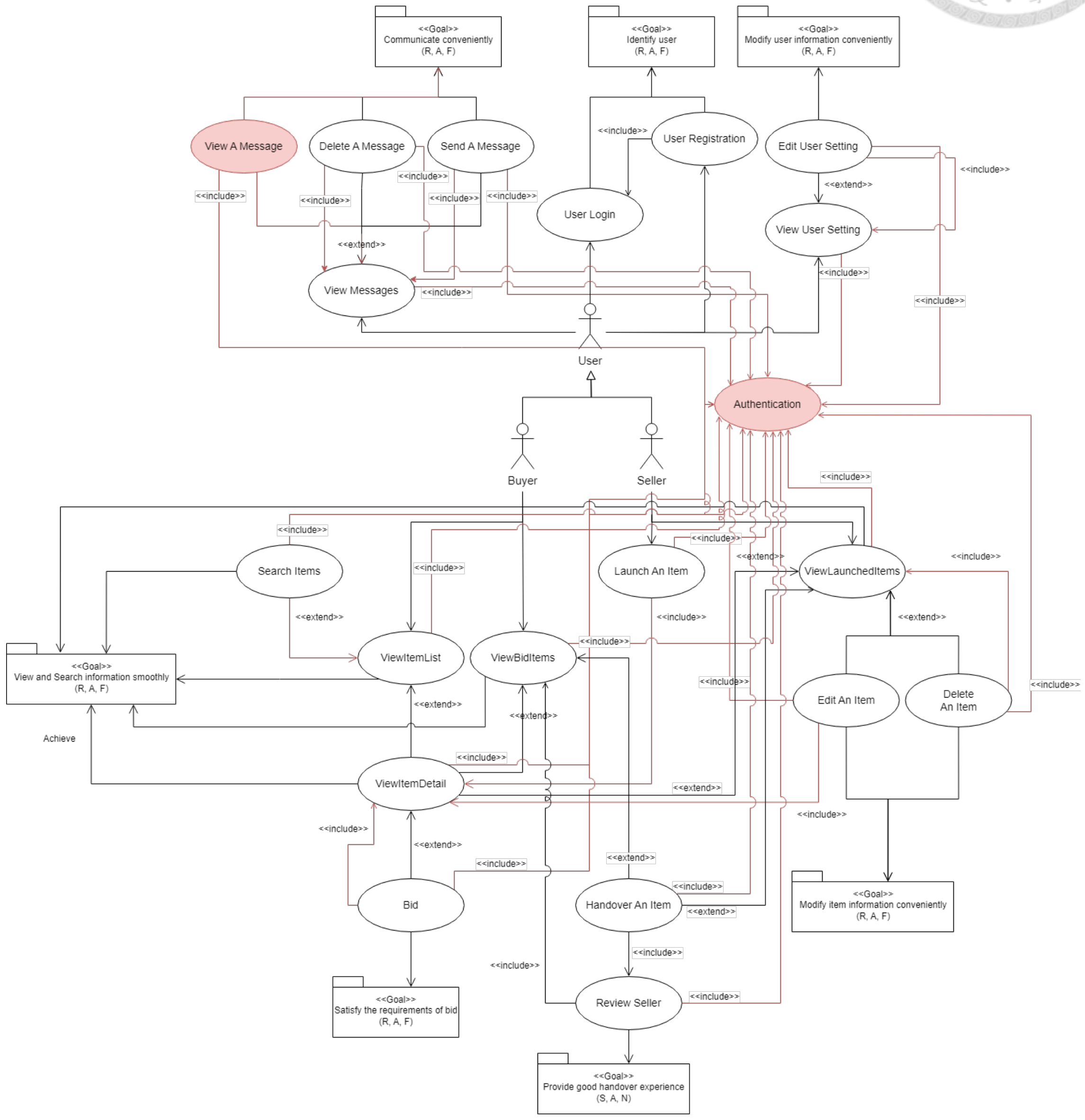

<<Goal>> Communicate conveniently (R, A, F)
<<Goal>> Identify user (R, A, F)
<<Goal>> Modify user information conveniently (R, A, F)
View A Message
Delete A Message
Send A Message
User Registration
Edit User Setting
User Login
View User Setting
View Messages
Authentication
User
Buyer
Seller
Search Items
Launch An Item
ViewLaunchedItems
ViewItemList
ViewBidItems
<<Goal>> View and Search information smoothly (R, A, F)
Achieve
Edit An Item
Delete An Item
ViewItemDetail
Bid
Handover An Item
<<Goal>> Modify item information conveniently (R, A, F)
<<Goal>> Satisfy the requirements of bid (R, A, F)
Review Seller
<<Goal>> Provide good handover experience (S, A, N)
<<include>>
<<extend>>

Figure 3.8: Example of Input Use Case Diagram

doi:10.6342/NTU202303644

Actor Type	Action Number (ID)	First Action	Number of Examples
User	1	No	15
User	2 or more	No	3
User	Don't care	Yes	14
Frontend	1	No	16
Frontend	Don't care	Yes	4

Table 3.5: Types of Few-Shot Learning Example

Test Type	GPT-NeoX	BertSum
ROUGE-1 Average R	0.44833	0.83267
ROUGE-1 Average P	0.39268	0.19385
ROUGE-1 Average F	0.40270	0.30395
ROUGE-2 Average R	0.15443	0.64208
ROUGE-2 Average P	0.14296	0.11508
ROUGE-2 Average F	0.14202	0.18578
ROUGE-L Average R	0.15923	0.81963
ROUGE-L Average P	0.386918	0.18981
ROUGE-L Average F	0.39813	0.29805
ROUGE-F(1/2/3/l)	41.87/14.85/22.56	30.39/18.58/29.80

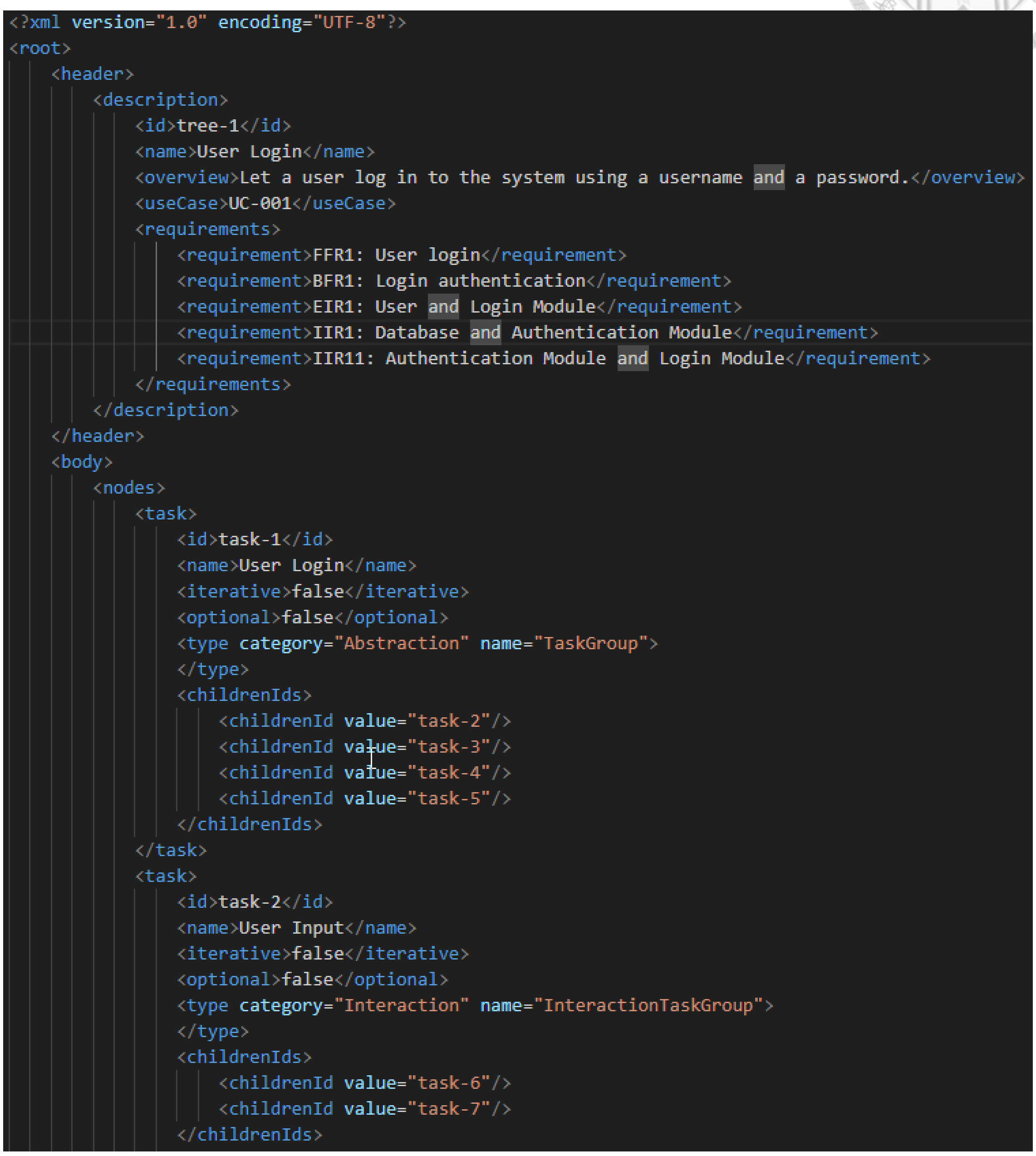
Table 3.6: Test Results of GPT-NeoX and BertSum for Task Generation

doi:10.6342/NTU202303644

Category	Task Type	Extra Attributes
Interaction	InteractionTaskGroup	
	Input	inputType
	SelectFromFixedList	options
	SelectFromDynamicallyAcquiredList	
	Control	
	SelectFromVisualizedInfo	visualizedTaskID
	RespAlert	alertTaskID
System	SystemTaskGroup	
	Checking	message
	ErrorMessage	message
	Feedback	
	FilterInfo	serviceTaskID
	InputValidation	validatorType
	VisualizeFixedValue	value
	VisualizeDynamicallyAcquiredValue	
	Service	

Table 3.7: Extra Attributes of Task

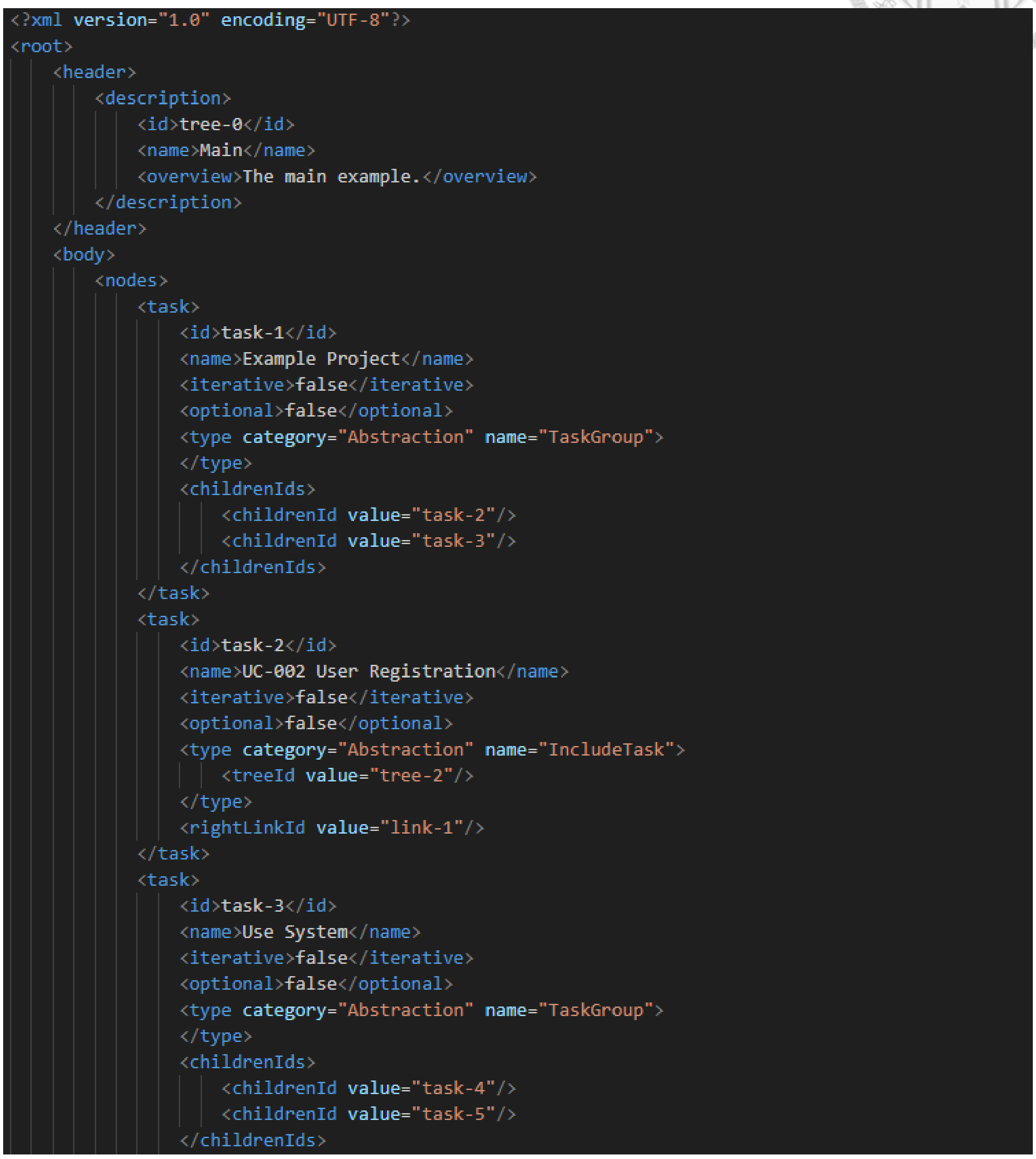
doi:10.6342/NTU202303644



```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <header>
        <description>
            <id>tree-1</id>
            <name>User Login</name>
            <overview>Let a user log in to the system using a username and a password.</overview>
            <useCase>UC-001</useCase>
            <requirements>
                <requirement>FFR1: User login</requirement>
                <requirement>BFR1: Login authentication</requirement>
                <requirement>EIR1: User and Login Module</requirement>
                <requirement>IIR1: Database and Authentication Module</requirement>
                <requirement>IIR11: Authentication Module and Login Module</requirement>
            </requirements>
        </description>
    </header>
    <body>
        <nodes>
            <task>
                <id>task-1</id>
                <name>User Login</name>
                <iterative>false</iterative>
                <optional>false</optional>
                <type category="Abstraction" name="TaskGroup">
                </type>
                <childrenIds>
                    <childrenId value="task-2"/>
                    <childrenId value="task-3"/>
                    <childrenId value="task-4"/>
                    <childrenId value="task-5"/>
                </childrenIds>
            </task>
            <task>
                <id>task-2</id>
                <name>User Input</name>
                <iterative>false</iterative>
                <optional>false</optional>
                <type category="Interaction" name="InteractionTaskGroup">
                </type>
                <childrenIds>
                    <childrenId value="task-6"/>
                    <childrenId value="task-7"/>
                </childrenIds>
```

Figure 3.9: Example of Use Case CTT in XML format

doi:10.6342/NTU202303644



```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <header>
        <description>
            <id>tree-0</id>
            <name>Main</name>
            <overview>The main example.</overview>
        </description>
    </header>
    <body>
        <nodes>
            <task>
                <id>task-1</id>
                <name>Example Project</name>
                <iterative>false</iterative>
                <optional>false</optional>
                <type category="Abstraction" name="TaskGroup">
                </type>
                <childrenIds>
                    <childrenId value="task-2"/>
                    <childrenId value="task-3"/>
                </childrenIds>
            </task>
            <task>
                <id>task-2</id>
                <name>UC-002 User Registration</name>
                <iterative>false</iterative>
                <optional>false</optional>
                <type category="Abstraction" name="IncludeTask">
                    <treeId value="tree-2"/>
                </type>
                <rightLinkId value="link-1"/>
            </task>
            <task>
                <id>task-3</id>
                <name>Use System</name>
                <iterative>false</iterative>
                <optional>false</optional>
                <type category="Abstraction" name="TaskGroup">
                </type>
                <childrenIds>
                    <childrenId value="task-4"/>
                    <childrenId value="task-5"/>
                </childrenIds>
```

Figure 3.10: Example of Main CTT in XML format

doi:10.6342/NTU202303644

Chapter 4

Machine Learning Design Pattern

The purpose of Machine Learning Design Patterns is to offer a new perspective for understanding and analyzing machine learning models, enabling us to enhance and develop these models in the future. This perspective is from the concept of design patterns in software engineering, which involves proposing solutions to recurring problems. By employing these solutions, we can solve similar problems without needing to reinvent the wheel.

We propose the solutions for Transformer-based machine learning models, which we refer to as Machine Learning Design Patterns. The process contains the following steps:

1. Analyze the functionalities of the modules in the machine learning models.
2. Group modules with similar functionalities.
3. Abstract each group for generalization.

doi:10.6342/NTU202303644

The details of each step are explained in 4.1, 4.2, and 4.3. The abstracted groups form what we refer to as Machine Learning Design Patterns.

4.1 Module Analysis

We focus on analyzing two types of machine learning models: GPT-3 and BERT. We chose these models because they are currently among the most widely used Transformer-based machine learning models. Their architectures are similar, but with slight differences, making them suitable for us to derive design patterns for this type of machine learning model.

First, we analyze the architectures of these two models. For BERT, its architecture can be divided into the following modules based on their order of execution:

1. Tokenizer
2. Encoder
3. Fully Connected Layer
4. Activation Function

For GPT-3, its architecture can also be divided into the following modules based on their order of execution:

1. Tokenizer
2. Decoder

doi:10.6342/NTU202303644

3. Layer Normalization
4. Linear Layer
5. Activation Function

Tokenizer, in both BERT and GPT-3, is responsible for converting the model's input into a format that the model can understand. Thus, we categorize the tokenizer's function as data preprocessing.

The Encoder in BERT and the Decoder in GPT-3 share similar architectures. The only difference lies in how they generate attention. Regardless of the method used to generate attention, their function is to capture the relationships between tokens in a sentence and the contextual meaning of tokens in different contexts. The multi-layer architecture of Encoders and Decoders generates token meanings in different ways for each layer, enhancing the model's language capabilities. Thus, we categorize the function of multi-layer Encoders and Decoders as language representation learning.

The Fully Connected Layer and Activation Function in BERT are responsible for converting the output of BERT into results relevant to our target task. Therefore, we categorize the functions of these two modules as fine-tuning.

In GPT-3, Layer Normalization, Linear Layer, and Activation Function are responsible for using the model's output to predict the next token to be generated. Thus, we categorize the functions of these three modules as prediction making. We present the final result of our analysis in Figure 4.1.

After analyzing the architectures of both models, we continue to analyze the key

doi:10.6342/NTU202303644

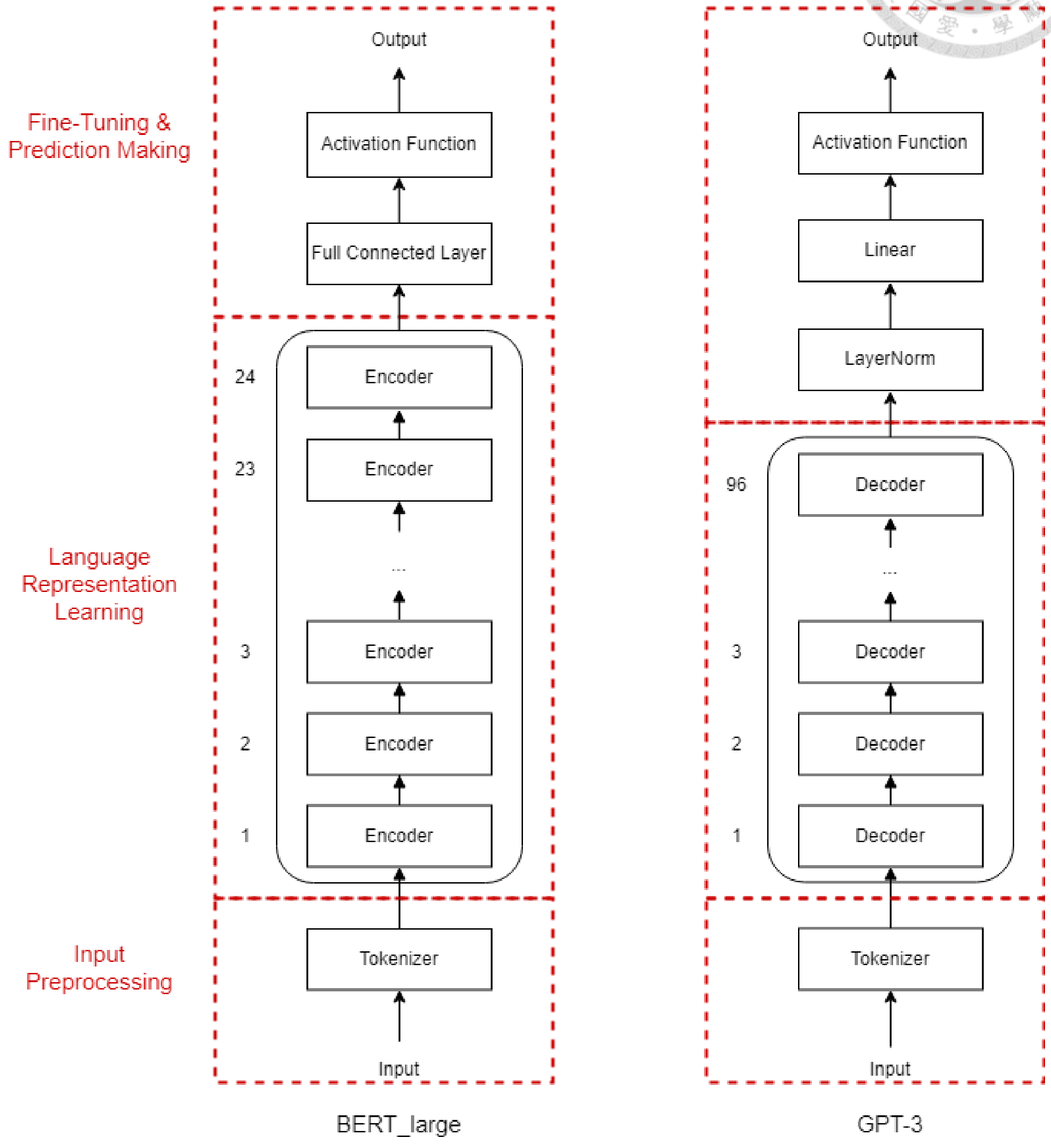


Figure 4.1: Analysis of GPT-3's Architecture and BERT's Architecture

doi:10.6342/NTU202303644

modules of the two models: the Encoder and the Decoder. The architectures of the Decoder and BERT's Encoder are very similar, differing mainly in the method used to generating Self-Attention. The term "Attention" refers to the degree of correlation. The term "Self-Attention" refers a special case of Attention where all the necessary data for calculating attention comes from the same input sequence. Similar to the language usage in real life, a word's meaning can vary in different contexts, and its correlation with other words in the same sentence is not fixed.

The purpose of generating Self-Attention is to enable the model to comprehend the overall meaning of a sentence based on the correlation between tokens. The Multi-Head Self-Attention mechanism calculates Self-Attention in multiple ways simultaneously, synthesizing the results of these different approaches. In difficult NLP tasks, this can yield better results compared to using only one way (head) to calculate Self-Attention. The difference between Masked and Non-Masked Self-Attention lies in the mechanism. Masked Self-Attention generates Attention only from words in the previous positions, whereas Non-Masked Self-Attention has no such limitation. This difference causes BERT and GPT-3 to excel at different tasks. The analysis results are shown below (See Figure 4.2).

After categorizing the modules of both models based on their functionalities, we continue to create class diagrams for these two models and mapped our analysis results onto these diagrams. First, each module of the models are represented as distinct classes. Then, we further decomposed the key modules, i.e., the Encoder and the Decoder, and represent the mechanism of generating Self-Attention using the pipeline design pattern.

doi:10.6342/NTU202303644

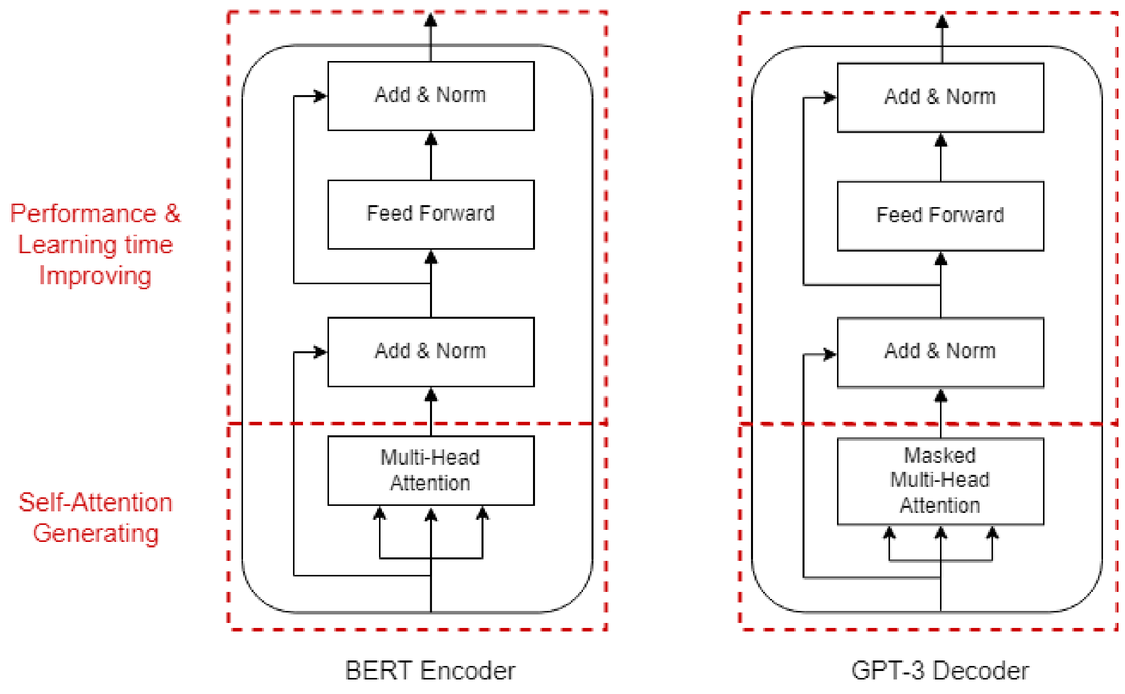


Figure 4.2: Analysis of GPT-3 Decoder and BERT Encoder

doi:10.6342/NTU202303644

Figure 4.3 illustrates the mapping of our analysis results onto the Class Diagram for BERT, while Figure 4.4 does the same for GPT-3.

Figure 4.5 and Figure 4.6 represent the Sequence Diagrams for the BERT Pipeline and Encoder Pipeline, respectively in the BERT's Class Diagram.

Similarly, Figure 4.7 and Figure 4.8 represent the Sequence Diagrams for the GPT-3 Pipeline and Decoder Pipeline, respectively in the GPT-3's Class Diagram.

4.2 Grouping

After analyzing the functionalities of the modules in both BERT and GPT-3, the results can be summarized as follows:

Input Preprocessing of BERT：

Purpose: process the input text so the next stages can use it properly.

Mechanism: use a tokenizer.

Output: a sequence of integer values representing the indices of the corresponding tokens in the vocabulary.

Language Representaion Learning of BERT：

Purpose: learn to capture each token's meaning and relations to all the tokens in the input sequence.

Mechanism: use Multi-Head Self-Attention mechanism.

doi:10.6342/NTU202303644

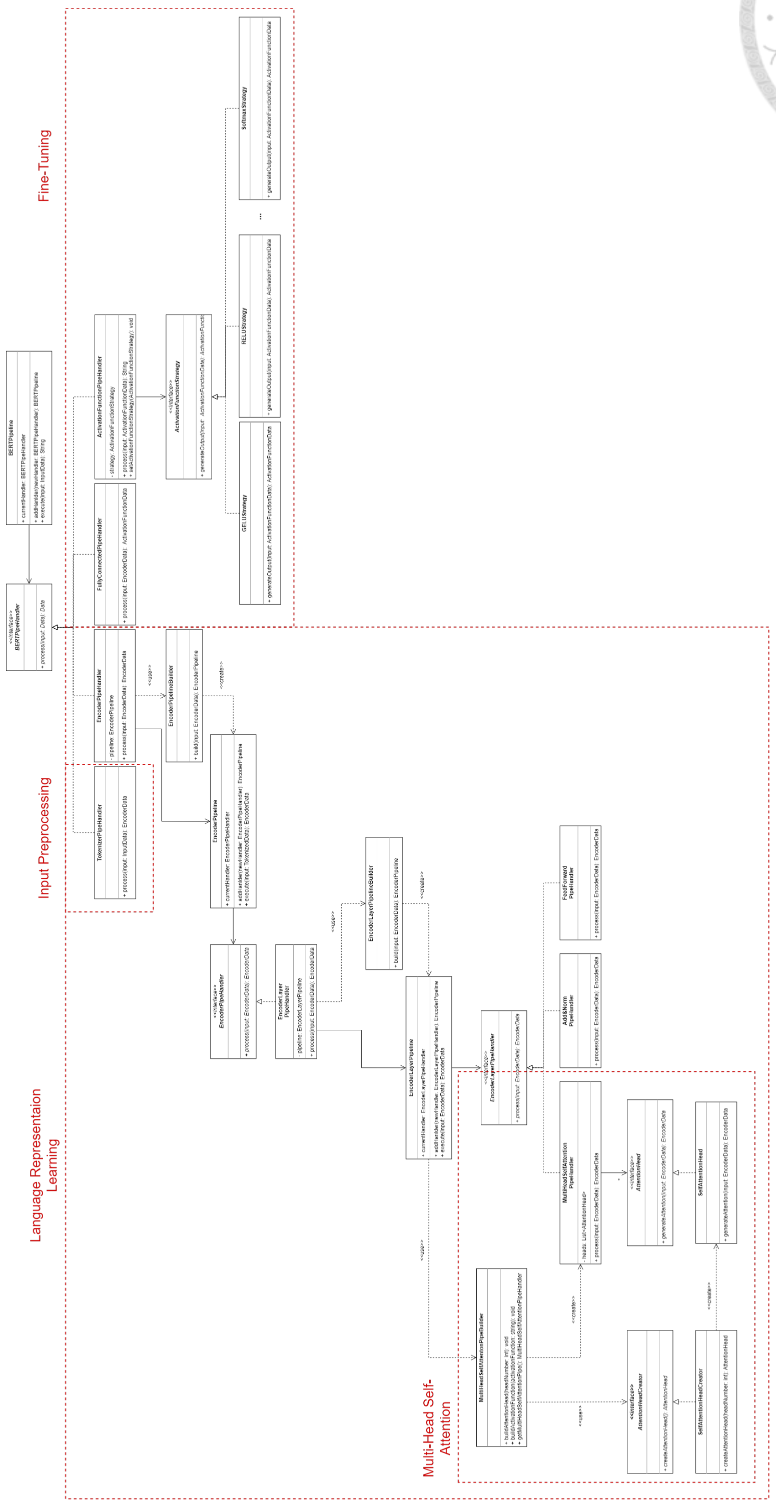

Fine-Tuning
Input Preprocessing
Language Representaion Learning
Multi-Head Self-Attention
BERTPipeline
<<interface>> BERTPipeHandler
FullyConnectedPipeHandler
ActivationFunctionPipeHandler
<<interface>> ActivationFunctionStrategy
GELUStrategy
RELUStrategy
SoftmaxStrategy
TokenizerPipeHandler
EncoderPipeHandler
EncoderPipelineBuilder
EncoderPipeline
<<interface>> EncoderPipeHandler
EncoderLayer PipeHandler
EncoderLayerPipelineBuilder
EncoderLayerPipeline
<<interface>> EncoderLayerPipeHandler
AddNorm PipeHandler
FeedForward PipeHandler
MultiHeadSelfAttention PipeHandler
<<interface>> AttentionHead
SelfAttentionHead
MultiHeadSelfAttentionPipeBuilder
<<interface>> AttentionHeadCreator
SelfAttentionHeadCreator
<<use>>
<<create>>

Figure 4.3: Analysis of BERT's Class Diagram

doi:10.6342/NTU202303644

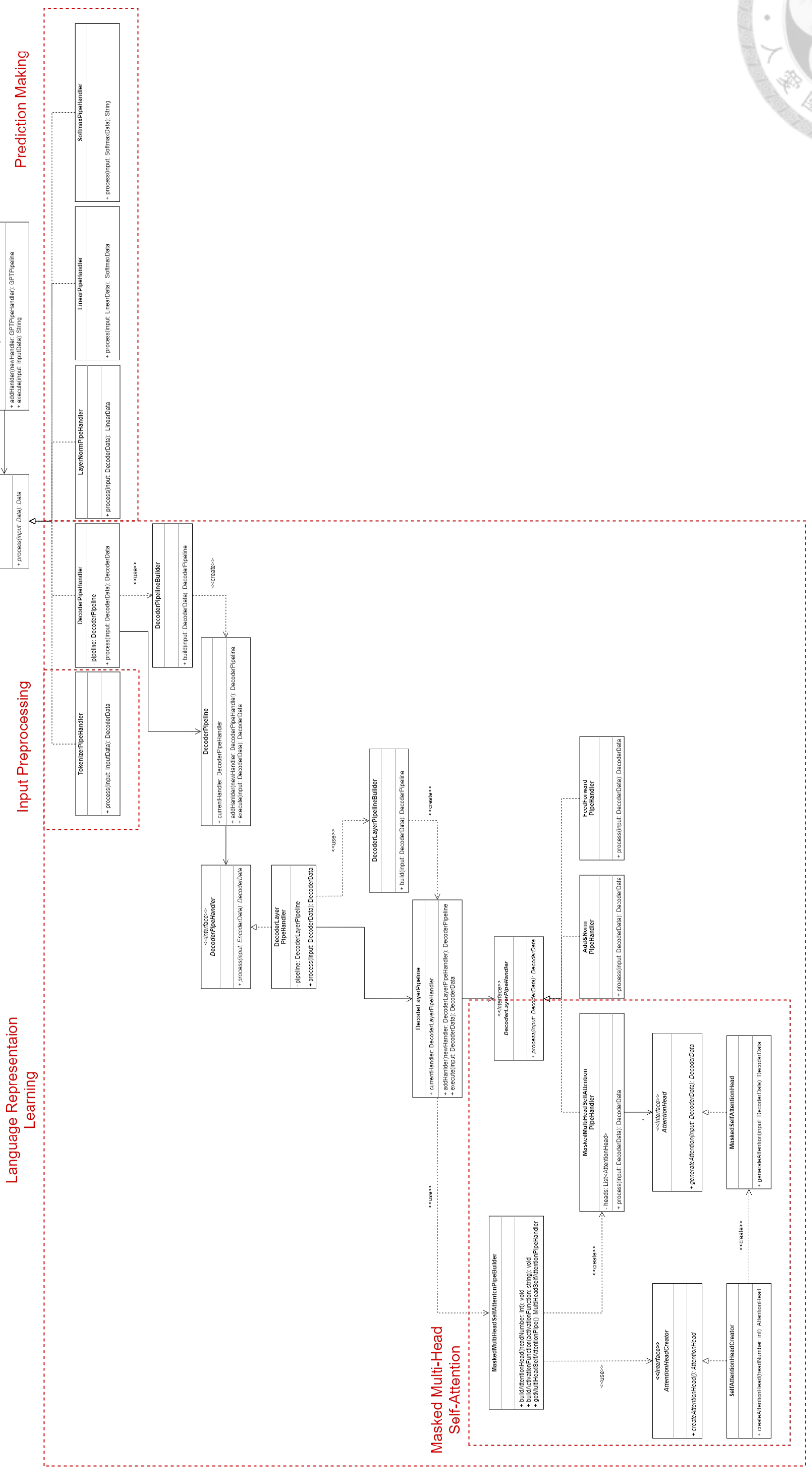

Prediction Making
Input Preprocessing
Language Representaion Learning
Masked Multi-Head Self-Attention
GPTPipeline
+ currentHandler: GPTPipeHandler
+ addHanlder(newHandler: GPTPipeHandler): GPTPipeline
+ execute(input: InputData): String
<<Interface>> GPTPipeHandler
+ process(input: Data): Data
SoftmaxPipeHandler
+ process(input: SoftmaxData): String
LinearPipeHandler
+ process(input: LinearData): SoftmaxData
LayerNormPipeHandler
+ process(input: DecoderData): LinearData
DecoderPipeHandler
- pipeline: DecoderPipeline
+ process(input: DecoderData): DecoderData
TokenizerPipeHandler
+ process(input: InputData): DecoderData
<<use>>
DecoderPipelineBuilder
+ build(input: DecoderData): DecoderPipeline
<<create>>
DecoderPipeline
+ currentHandler: DecoderPipeHandler
+ addHanlder(newHandler: DecoderPipeHandler): DecoderPipeline
+ execute(input: DecoderData): DecoderData
<<Interface>> DecoderPipeHandler
+ process(input: EncoderData): DecoderData
DecoderLayer PipeHandler
- pipeline: DecoderLayerPipeline
+ process(input: DecoderData): DecoderData
<<use>>
DecoderLayerPipelineBuilder
+ build(input: DecoderData): DecoderPipeline
<<create>>
DecoderLayerPipeline
+ currentHandler: DecoderLayerPipeHandler
+ addHanlder(newHandler: DecoderLayerPipeHandler): DecoderPipeline
+ execute(input: DecoderData): DecoderData
<<Interface>> DecoderLayerPipeHandler
+ process(input: DecoderData): DecoderData
FeedForward PipeHandler
+ process(input: DecoderData): DecoderData
Add&Norm PipeHandler
+ process(input: DecoderData): DecoderData
MaskedMultiHeadSelfAttention PipeHandler
- heads: List<AttentionHead>
+ process(input: DecoderData): DecoderData
*
<<Interface>> AttentionHead
+ generateAttention(input: DecoderData): DecoderData
MaskedSelfAttentionHead
+ generateAttention(input: DecoderData): DecoderData
<<use>>
MaskedMultiHeadSelfAttentonPipeBuilder
+ buildAttentionHead(headNumber: int): void
+ buildActivationFunction(activationFunction: string): void
+ getMultiHeadSelfAttentionPipe(): MultiHeadSelfAttentionPipeHandler
<<create>>
<<use>>
<<Interface>> AttentionHeadCreator
+ createAttentionHead(): AttentionHead
<<create>>
SelfAttentionHeadCreator
+ createAttentionHead(headNumber: int): AttentionHead

Figure 4.4: Analysis of GPT-3's Class Diagram

doi:10.6342/NTU202303644

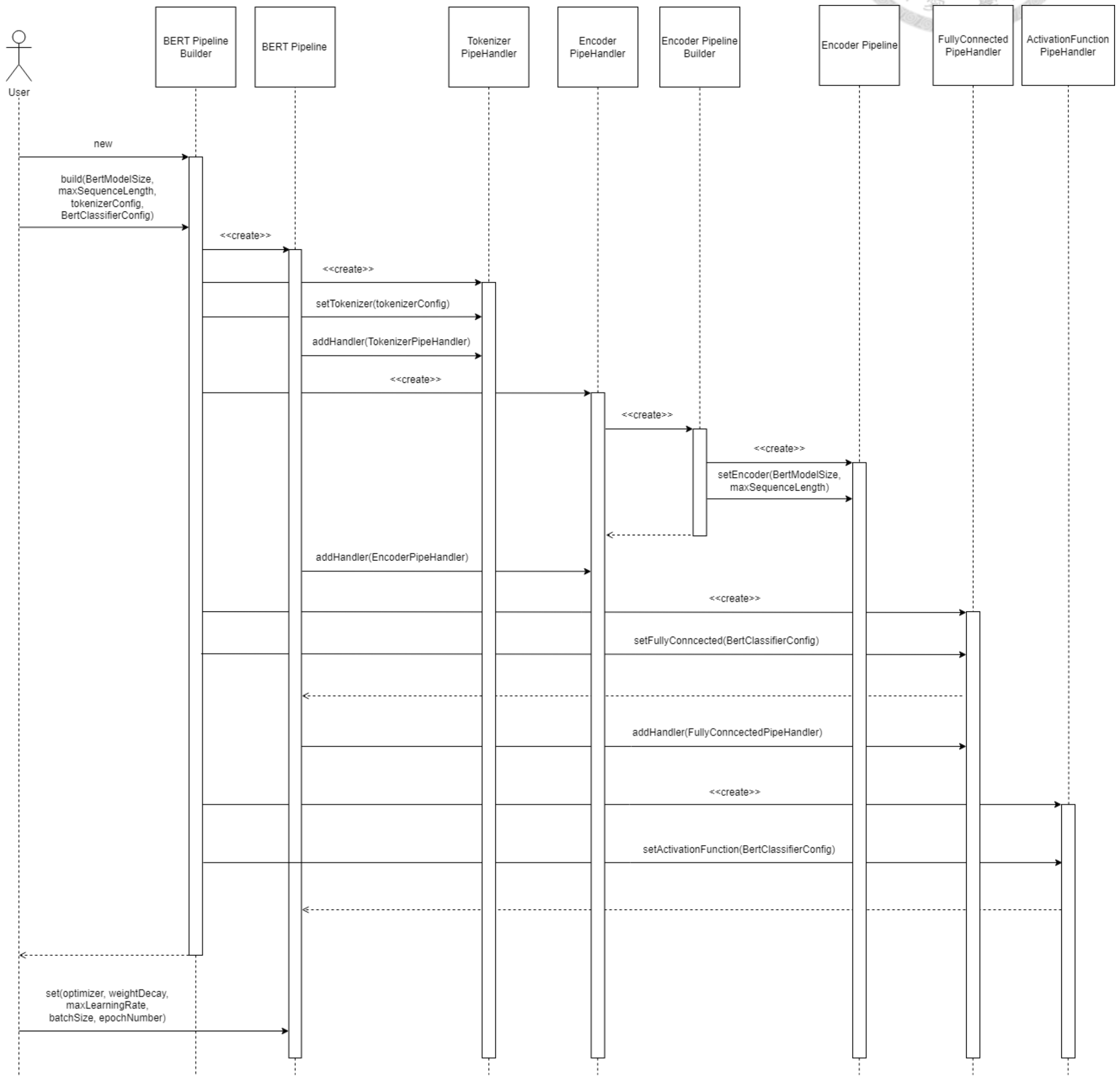

User
BERT Pipeline Builder
BERT Pipeline
Tokenizer PipeHandler
Encoder PipeHandler
Encoder Pipeline Builder
Encoder Pipeline
FullyConnected PipeHandler
ActivationFunction PipeHandler
new
build(BertModelSize, maxSequenceLength, tokenizerConfig, BertClassifierConfig)
<<create>>
<<create>>
setTokenizer(tokenizerConfig)
addHandler(TokenizerPipeHandler)
<<create>>
<<create>>
<<create>>
setEncoder(BertModelSize, maxSequenceLength)
addHandler(EncoderPipeHandler)
<<create>>
setFullyConncected(BertClassifierConfig)
addHandler(FullyConncectedPipeHandler)
<<create>>
setActivationFunction(BertClassifierConfig)
set(optimizer, weightDecay, maxLearningRate, batchSize, epochNumber)

Figure 4.5: Sequence Diagram of BERT Pipeline

doi:10.6342/NTU202303644

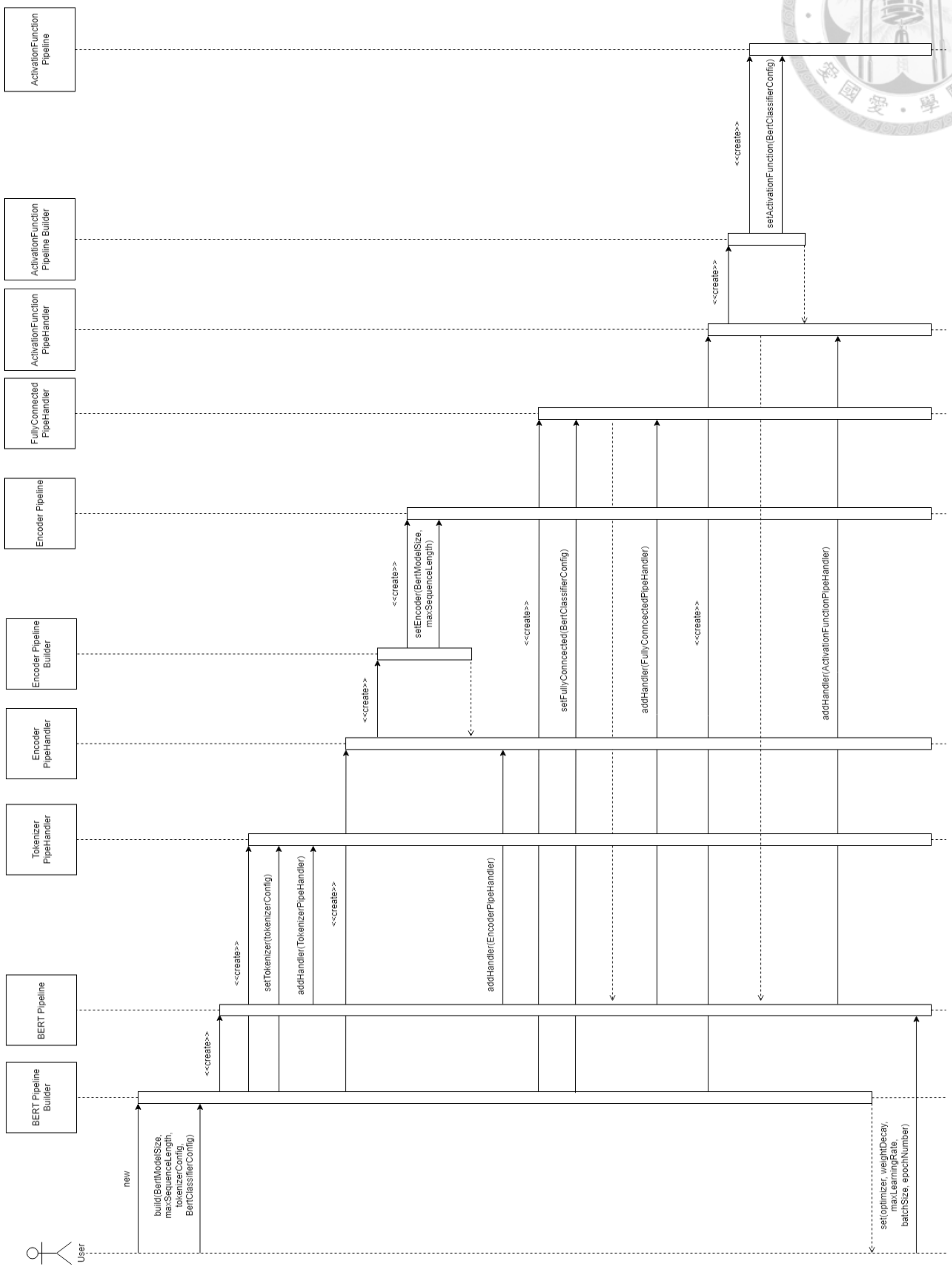

ActivationFunction Pipeline
ActivationFunction Pipeline Builder
ActivationFunction PipeHandler
FullyConnected PipeHandler
Encoder Pipeline
Encoder Pipeline Builder
Encoder PipeHandler
Tokenizer PipeHandler
BERT Pipeline
BERT Pipeline Builder
User
new
build(BertModelSize, maxSequenceLength, tokenizerConfig, BertClassifierConfig)
<<create>>
<<create>>
setTokenizer(tokenizerConfig)
addHandler(TokenizerPipeHandler)
<<create>>
<<create>>
<<create>>
setEncoder(BertModelSize, maxSequenceLength)
addHandler(EncoderPipeHandler)
<<create>>
setFullyConnected(BertClassifierConfig)
addHandler(FullyConncectedPipeHandler)
<<create>>
<<create>>
<<create>>
setActivationFunction(BertClassifierConfig)
addHandler(ActivationFunctionPipeHandler)
set(optimizer, weightDecay, maxLearningRate, batchSize, epochNumber)

Figure 4.6: Sequence Diagram of Encoder Pipeline

doi:10.6342/NTU202303644

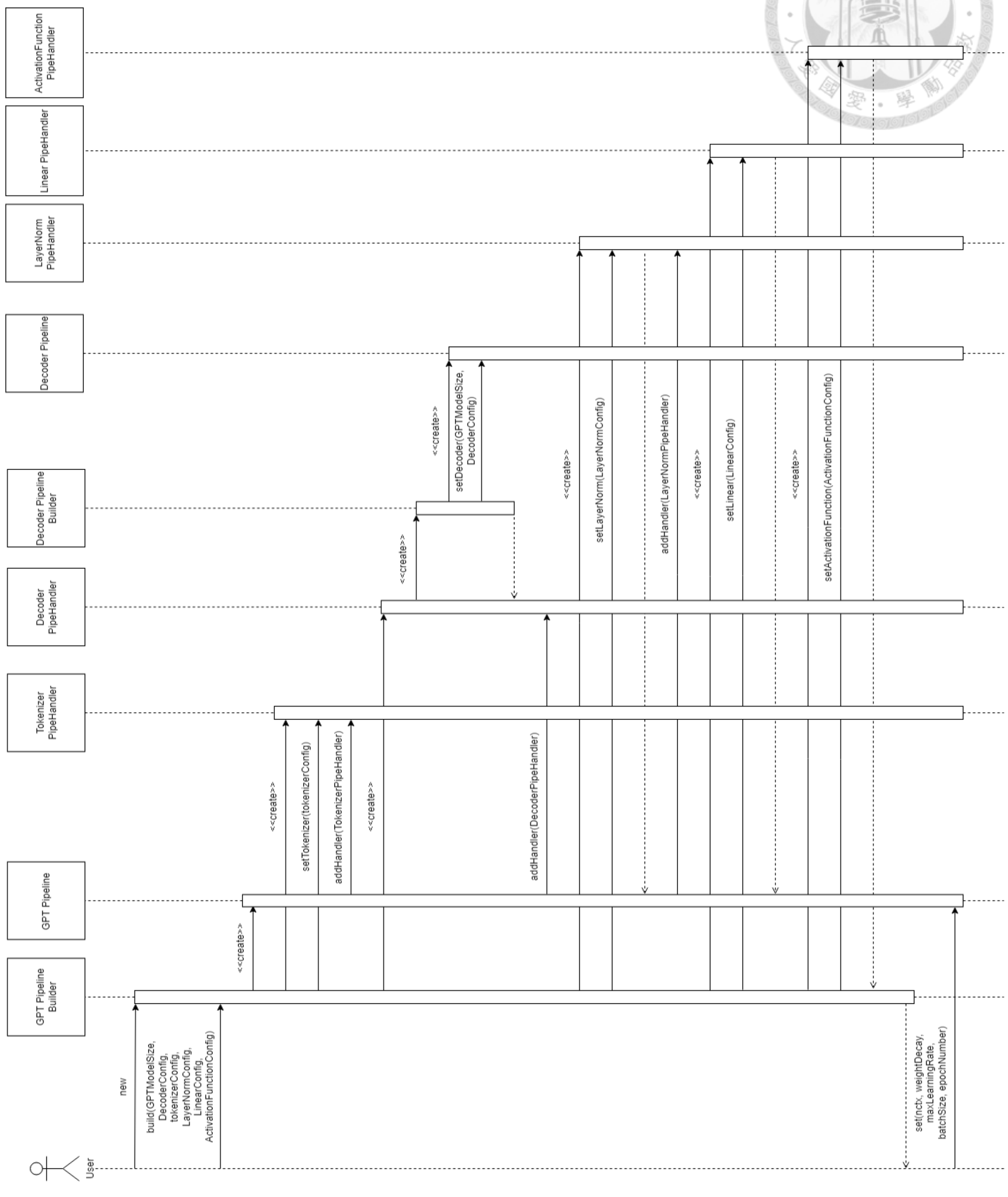

ActivationFunction PipeHandler
Linear PipeHandler
LayerNorm PipeHandler
Decoder Pipeline
Decoder Pipeline Builder
Decoder PipeHandler
Tokenizer PipeHandler
GPT Pipeline
GPT Pipeline Builder
User
new
build(GPTModelSize, DecoderConfig, tokenizerConfig, LayerNormConfig, LinearConfig, ActivationFunctionConfig)
<<create>>
<<create>>
setTokenizer(tokenizerConfig)
addHandler(TokenizerPipeHandler)
<<create>>
addHandler(DecoderPipeHandler)
<<create>>
<<create>>
setDecoder(GPTModelSize, DecoderConfig)
<<create>>
setLayerNorm(LayerNormConfig)
addHandler(LayerNormPipeHandler)
<<create>>
setLinear(LinearConfig)
<<create>>
setActivationFunction(ActivationFunctionConfig)
set(nctx, weightDecay, maxLearningRate, batchSize, epochNumber)

Figure 4.7: Sequence Diagram of GPT-3 Pipeline

doi:10.6342/NTU202303644

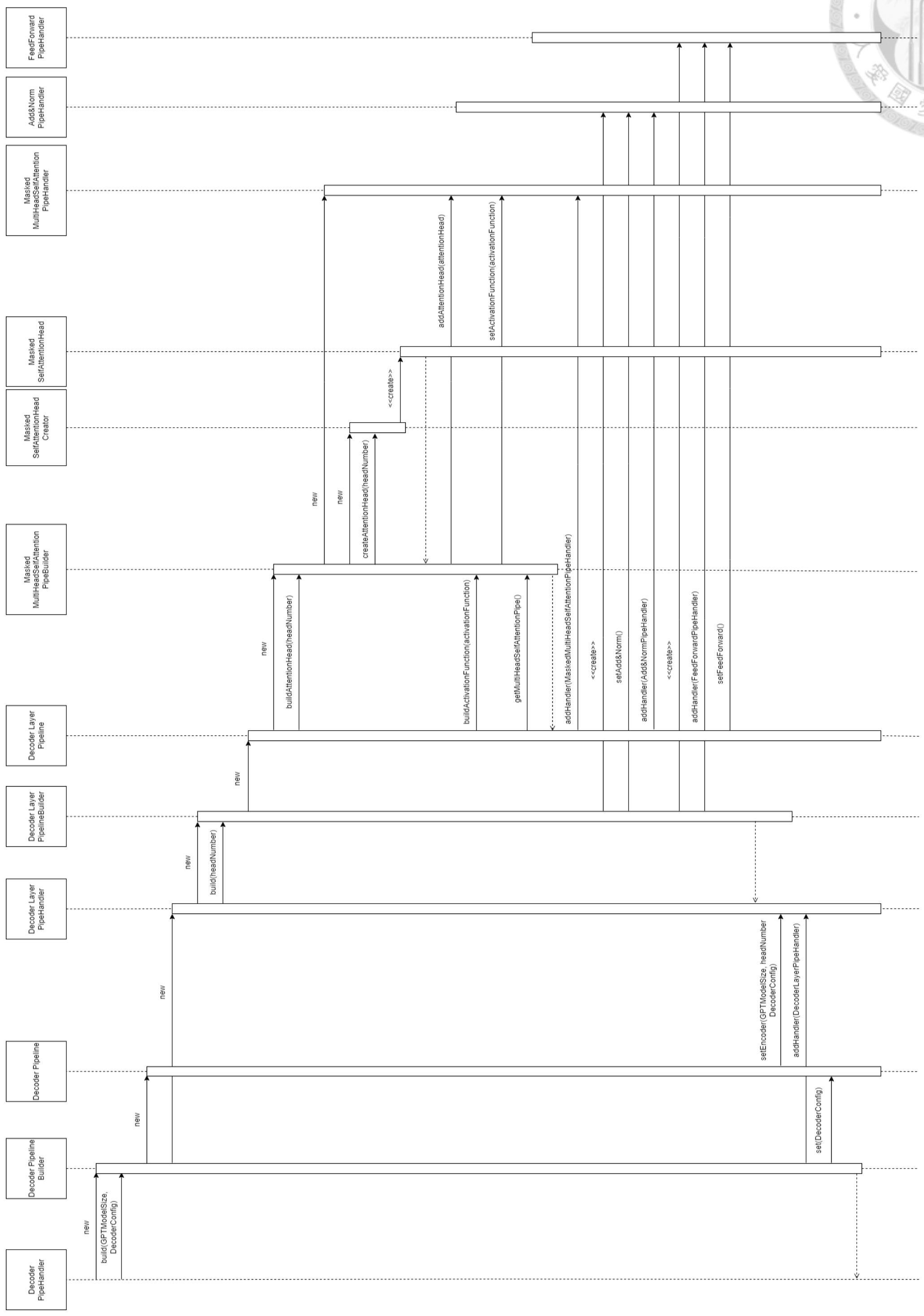

FeedForward PipeHandler
Add&Norm PipeHandler
Masked MultiHeadSelfAttention PipeHandler
Masked SelfAttentionHead
Masked SelfAttentionHead Creator
Masked MultiHeadSelfAttention PipeBuilder
Decoder Layer Pipeline
Decoder Layer PipelineBuilder
Decoder Layer PipeHandler
Decoder Pipeline
Decoder Pipeline Builder
Decoder PipeHandler
new
build(GPTModelSize, DecoderConfig)
new
new
set(DecoderConfig)
setEncoder(GPTModelSize, headNumber DecoderConfig)
addHandler(DecoderLayerPipeHandler)
new
build(headNumber)
new
new
buildAttentionHead(headNumber)
buildActivationFunction(activationFunction)
getMultiHeadSelfAttentionPipe()
addHandler(MaskedMultiHeadSelfAttentionPipeHandler)
<<create>>
setAdd&Norm()
addHandler(Add&NormPipeHandler)
<<create>>
addHandler(FeedForwardPipeHandler)
setFeedForward()
new
new
createAttentionHead(headNumber)
<<create>>
addAttentionHead(attentionHead)
setActivationFunction(activationFunction)

Figure 4.8: Sequence Diagram of Decoder Pipeline

doi:10.6342/NTU202303644

Output: a sequence of contextualized representations for each token in the input sequence.

Fine-Tuning of BERT：

Purpose: make the BERT model generate the specific output based on the target task we give.

Mechanism: adding extra task-specific output layers like a fully connected layer followed by an activation function layer and using specific labeled data for training.

Output: the target task's output.

Input Preprocessing of GPT-3：

Purpose: process the input text so the next stages can use it properly.

Mechanism: use a tokenizer.

Output: a sequence of integer values representing the indices of the corresponding tokens in the vocabulary.

Language Representaion Learning of GPT-3：

Purpose: learn to capture each token's meaning and relations to previous token in the input sequence.

Mechanism: use Masked Multi-Head Self-Attention mechanism.

doi:10.6342/NTU202303644

Output: a sequence of contextualized representations for each token in the input sequence.

Prediction Making of GPT-3：

Purpose: predict the next token to be generated.

Mechanism: use a linear layer followed by a activation function layer.

Output: next token to be generated.

Fine-Tuning of GPT-3 (Optional)：

Purpose: make the GPT model work better on target tasks.

Mechanism: add extra task-specific output layers like a fully connected layer followed by an activation function layer and use specific labeled data for training.

Output: the target task's output.

The correspondence between the results of Grouping and the modules in the original model architectures can be referred to in Table 4.1.

4.3 Abstraction

Lastly, based on the results of Grouping, we can perform Abstraction. “Input Preprocessing” can be abstracted as “Preprocessing,” indicating that in this stage,

doi:10.6342/NTU202303644

BERT	GPT-3	Grouping of BERT	Grouping of GPT-3
Tokenization		Input Preprocessing	Input Preprocessing
Multi-Head Self-Attention	Masked Multi-Head Self-Attention	Language Representation Learning	Language Representation Learning
Add and Norm			
Feed Forward			
	Linear Layer		Prediction Making
	Activation Function		
Fully Connected Layer	Fully Connected Layer (Optional)	Fine-Tuning	Fine-Tuning (Optional)
Activation Function	Activation Function (Optional)		

Table 4.1: Grouping of BERT and GPT-3 by Functionality

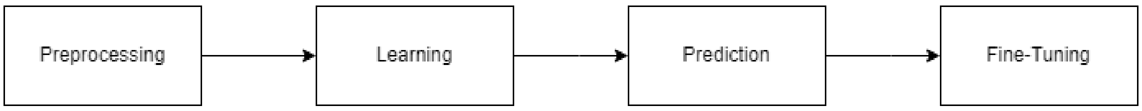


Figure 4.9: Process of Machine Learning Design Pattern

the model focuses on preprocessing tasks; "Language Representation Learning" can be abstracted as "Learning," indicating that in this stage, the model focuses on learning various types of self-attention; "Prediction Making" can be abstracted as "Prediction," indicating that in this stage, the model focuses on predicting the next output token; "Fine-Tuning" remains unchanged as it cannot be further abstracted, the model is trained to focus on generating the target task's output in this stage. The relationship between Grouping and Abstraction is shown in Table 4.2. Thus, we can have the process of the resulting Machine Learning Design Pattern (See Figure 4.9).

doi:10.6342/NTU202303644

Grouping of BERT	Grouping of GPT-3	Abstraction
Input Preprocessing	Input Preprocessing	Preprocessing
Language Representaion Learning	Language Representaion Learning	Learning
	Prediction Making	Prediction
Fine-Tuning	Fine-Tuning (Optional)	Fine-Tuning

Table 4.2: Abstraction for BERT and GPT-3

doi:10.6342/NTU202303644

Chapter 5

Composition Engine

Composition Engine is responsible for generating frontend code. After previous works (See [7] and [10]), Composition Engine can now use templates and a template engine to integrate the information from three types of description languages: PDL, SUMDL, and NDL of a web page to generate corresponding Angular code. These three description languages are all in JSON format, and their descriptions are as follows:

PDL (PageUICDL): describes the composition of UI components and their information in a web page, including the properties and functionalities of the UI components.

SUMDL: describes the service information in a web page, including the service's content, the binding relationships of services and UI components and the service return information.

doi:10.6342/NTU202303644

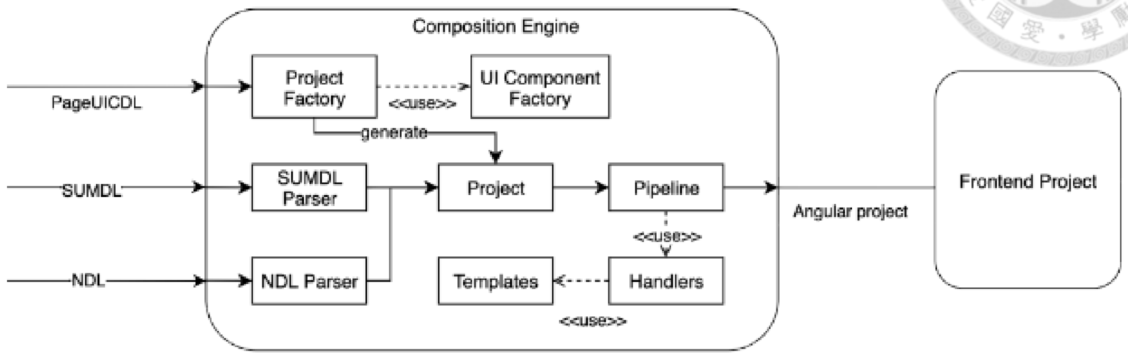


Figure 5.1: Process of Composition Engine

NDL: describes the navigation information in a web page, including the navigation destinations, navigation types and parameters received from other web pages by navigation.

5.1 Angular API

"Angular API" refers to the functions provided by Angular which enable Angular users to construct or organize the structure and functionalities of web applications. In order to enable our system's users to construct more complex features using these functions, it is necessary to implement the Angular APIs in both UI Design Client and Composition Engine.

We chose to implement the most frequently used Angular APIs, which can be categorized into three types:

doi:10.6342/NTU202303644

Transform value: allows data or values to be transformed into different formats or units, such as changing the format of dates and times.

Change behavior and appearance based on certain variables: allows UI components on the web page to have different behaviors under specific conditions. For example, displaying a success or failure message based on the success of triggering login service.

Change certain variables based on user's action: allows user actions to change the value of specific variable on the web page, influencing the behavior of the web application. For example, the web page can check if the user input for the username field is empty, determining whether the user can click the "Login" button.

The process of implementing Angular APIs in Composition Engine can be divided into three steps:

Add decorator: we use a decorator design pattern to extend the functionalities of UI components. Therefore, we need to add corresponding decorators first.

Add pipeline handler: we use a pipeline design pattern to handle decorators. Therefore, we need to add corresponding handlers to the pipeline after adding decorators. Moreover, we use a chain of responsibility design pattern to optimize the number of handlers in the pipeline.

Add render strategy: UI components are rendered to corresponding frontend code in the pipeline. Each decorator's render strategy determines how to map the

doi:10.6342/NTU202303644

decorator's information to corresponding frontend code and is different for each UI component. Hence, we add the new added decorator's corresponding render strategies to each UI Component's render strategies. This process enables the decorators of Angular APIs to be converted into frontend code.

The implemented Angular API types are shown in Table 5.1. The corresponding class diagram is shown below (See Figure 5.2).

Category	Angular API Type	Angular APIs
Transform value	Pipe	CurrencyPipe, DatePipe, DecimalPipe, JsonPipe, KeyValuePipe, LowerCasePipe, UpperCasePipe, PercentPipe, SlicePipe, TitleCasePipe
	Format	The functionality is similar to that of pipes, so there is no need for implementation.
Change behavior and appearance based on certain variables	Structural Directive	NgFor, NgIf, NgSwitch
	Attribute Directive	NgStyle
Change certain variables based on user's action	Validator	EmailValidator, MaxLengthValidator, MinLengthValidator, RequiredValidator, PatternValidator

Table 5.1: Implementation Status of Angular API

doi:10.6342/NTU202303644

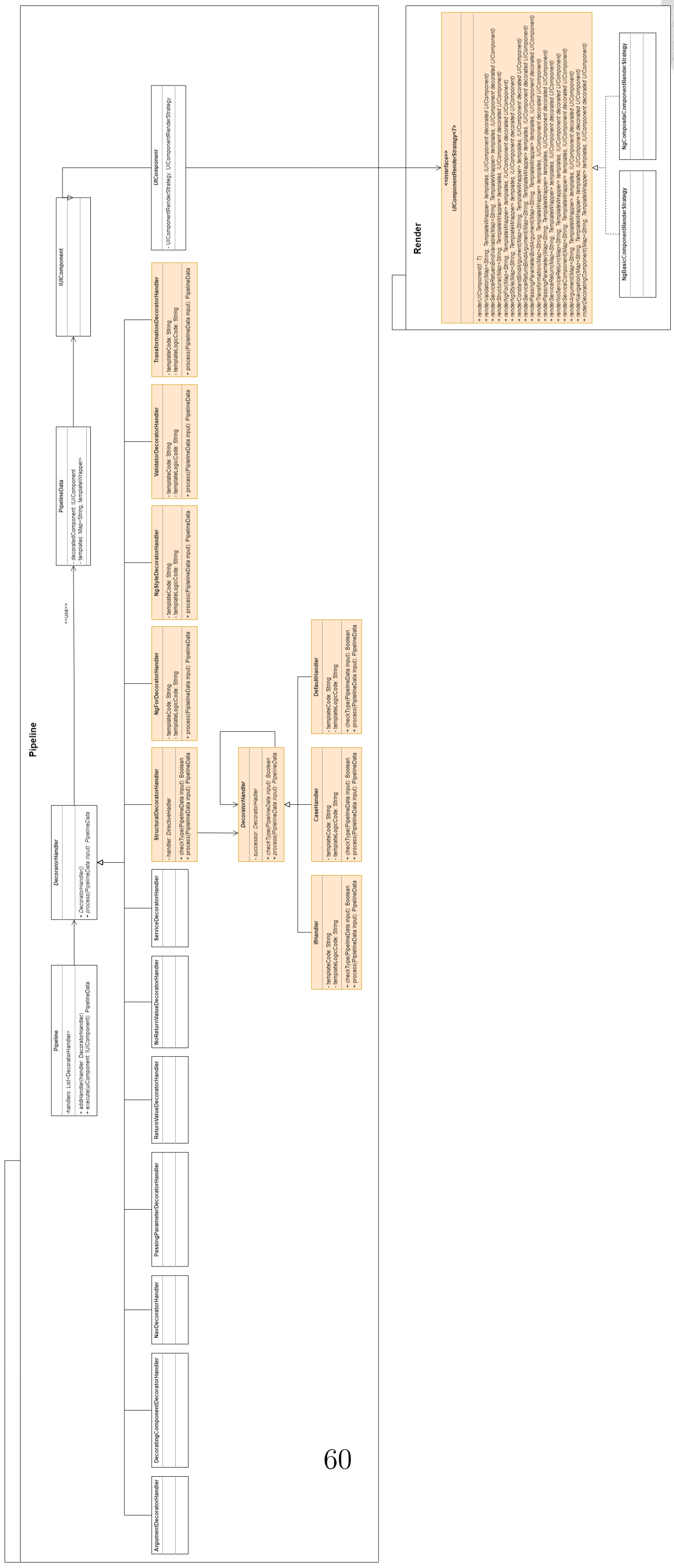

Pipeline
Render
Pipeline
DecoratorHandler
PipelineData
IUIComponent
UIComponent
<<use>>
StructuralDecoratorHandler
NgForDecoratorHandler
NgStyleDecoratorHandler
ValidatorDecoratorHandler
TransformationDecoratorHandler
ServiceDecoratorHandler
NoReturnValueDecoratorHandler
ReturnValueDecoratorHandler
PassingParameterDecoratorHandler
NavDecoratorHandler
DecoratingComponentDecoratorHandler
ArgumentDecoratorHandler
DecoratorHandler
IfHandler
CaseHandler
DefaultHandler
<<interface>>
UIComponentRenderStrategy<T>
NgBasicComponentRenderStrategy
NgCompositeComponentRenderStrategy
60

Figure 5.2: Class Diagram of Angular API in Composition Engine

doi:10.6342/NTU202303644

5.2 Authentication

The authentication mechanism is responsible for enabling different actors in a CTT to execute different use cases. It allows the UI components on our web page to exhibit different behaviors based on the current user's actor. For example, a "Buy" button is displayed only when the current user's actor is "Buyer," otherwise it remains hidden. This mechanism is implemented using the Angular APIs introduced in 5.1.

To implement this mechanism, we revise Composition Engine to make the generated frontend code have the following functionalities:

Get actor information: we use a singleton design pattern to make UI components are able to fetch the current actor's information and update corresponding permissions.

Update actor information: actor information is updated only when the user navigates to other web pages.

Change UI component's behavior: once the current actor's information is updated, the UI components on the web page will change their behavior based on the new actor's permissions, determining whether to show or not.

The corresponding class diagram is shown below (See Figure 5.3).

Next, we illustrate how the authentication mechanism operates in a real-world web application. First, after entering the web application's homepage, which is the

doi:10.6342/NTU202303644

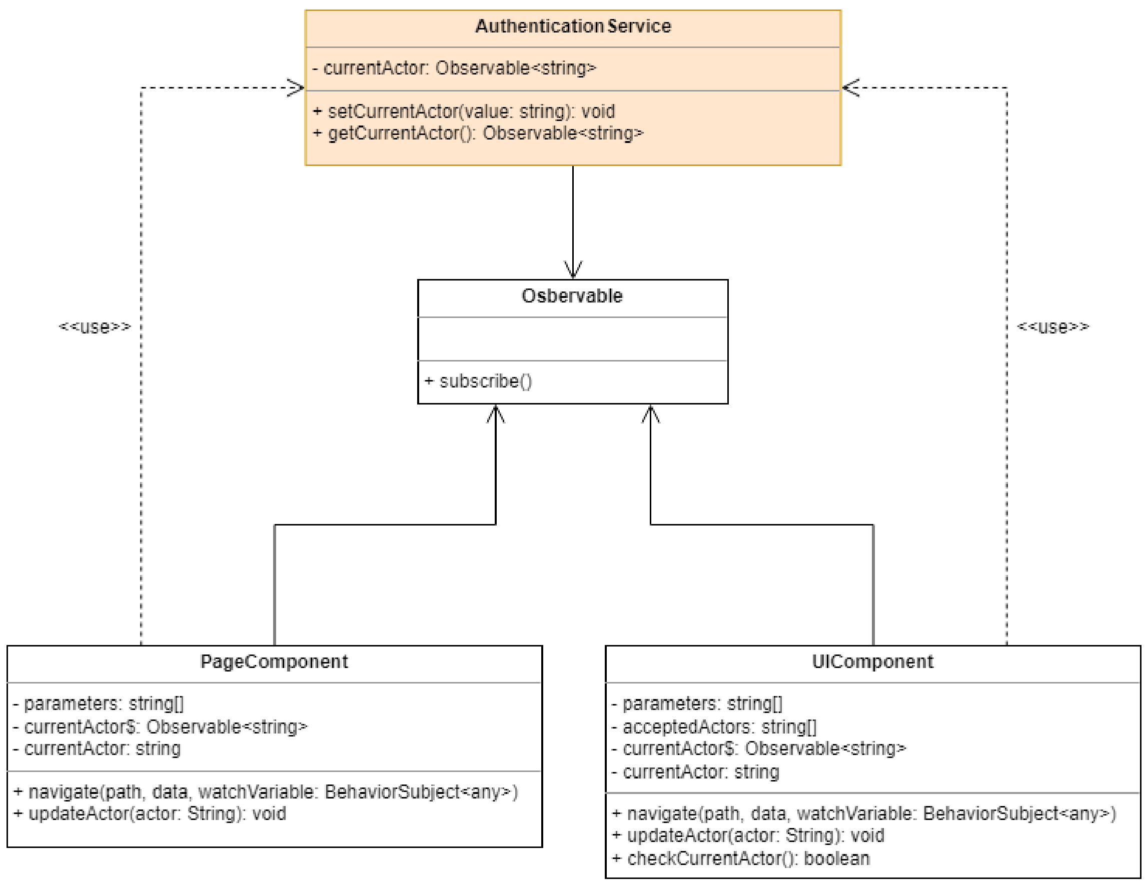

Authentication Service
- currentActor: Observable<string>
+ setCurrentActor(value: string): void
+ getCurrentActor(): Observable<string>
Osbervable
+ subscribe()
<<use>>
<<use>>
PageComponent
- parameters: string[]
- currentActor$: Observable<string>
- currentActor: string
+ navigate(path, data, watchVariable: BehaviorSubject<any>)
+ updateActor(actor: String): void
UIComponent
- parameters: string[]
- acceptedActors: string[]
- currentActor$: Observable<string>
- currentActor: string
+ navigate(path, data, watchVariable: BehaviorSubject<any>)
+ updateActor(actor: String): void
+ checkCurrentActor(): boolean

Figure 5.3: Class Diagram of Authentication Mechanism in Web Application

doi:10.6342/NTU202303644

"Login" page, the current actor is automatically updated to the lowest permission level of "User," as we have not yet navigated to any web page. (See Figure 5.4).

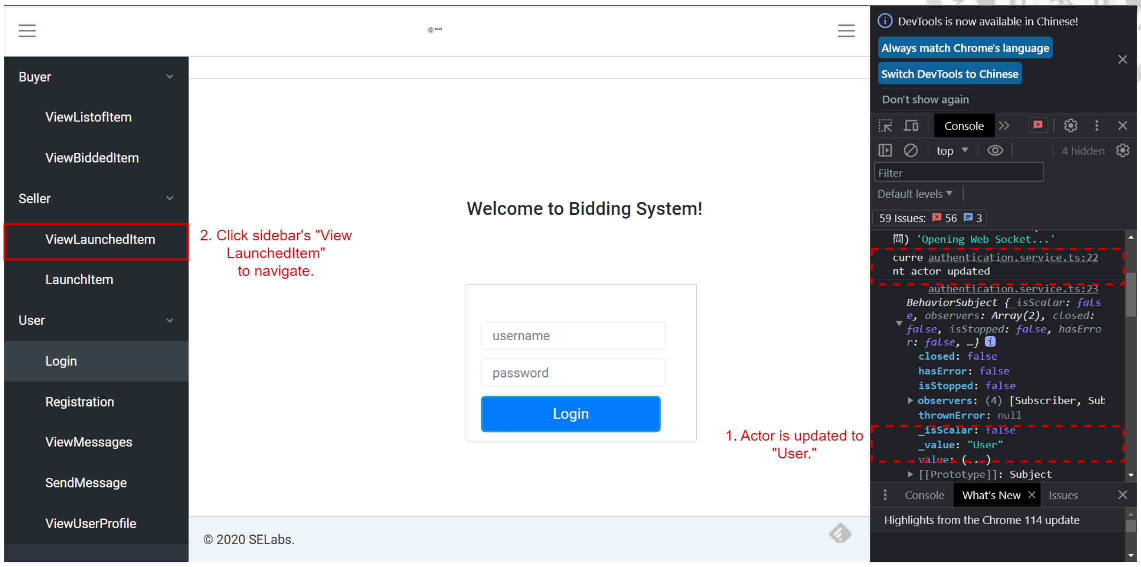
At this point, if we navigate to another web page, such as the "ViewLaunchedItem" page, we will see nothing because of the limited permissions of "User" (See Figure 5.5).

We return to the "Login" page, then enter the user name and password to login. After successfully logging in by clicking the "Login" button, the system navigates us to the "ViewLaunchedItem" page displaying the user's listed launched items (See Figure 5.6).

Simultaneously, our actor is updated to "Seller," allowing us to view the listed items. We click the "Go" button under the "Fulfill Order" column of one of the items to check the current order status. Later, the system navigates us to the "OrderStatus" page (See Figure 5.7).

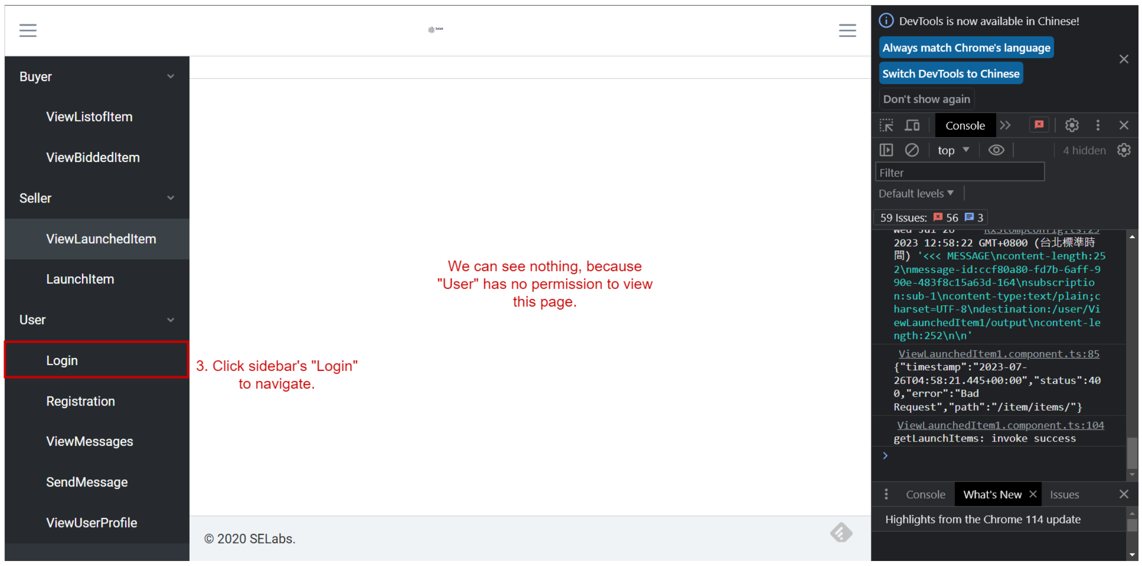
Because our permissions are updated to that of "Seller," we can see the "Payment Received" button, enabling us to confirm receipt of payment (See Figure 5.8). Thus, we have shown how the authentication mechanism works and its features in the web application.

doi:10.6342/NTU202303644



```
間) 'Opening Web Socket...'
curre authentication.service.ts:22
nt actor updated
          authentication.service.ts:23
  BehaviorSubject {_isScalar: fals
  e, observers: Array(2), closed:
  false, isStopped: false, hasErro
  r: false, …}
    closed: false
    hasError: false
    isStopped: false
  ▸ observers: (4) [Subscriber, Sub
    thrownError: null
    _isScalar: false
    _value: "User"
    value: (...)
  ▸ [[Prototype]]: Subject
```

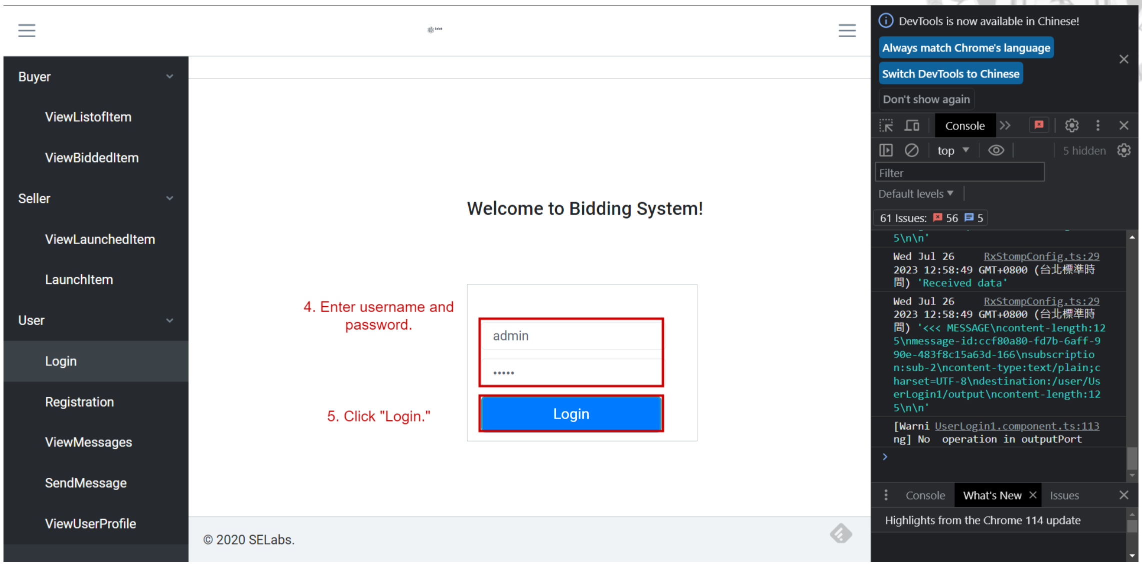
Figure 5.4: Scenario of Authentication, Step 1


Buyer
ViewListofItem
ViewBiddedItem
Seller
ViewLaunchedItem
LaunchItem
User
Login
Registration
ViewMessages
SendMessage
ViewUserProfile
© 2020 SELabs.
We can see nothing, because "User" has no permission to view this page.
3. Click sidebar's "Login" to navigate.
DevTools is now available in Chinese!
Always match Chrome's language
Switch DevTools to Chinese
Don't show again
Console
top
4 hidden
Filter
Default levels
59 Issues: 56 3
Console
What's New
Issues
Highlights from the Chrome 114 update

```
Wed Jul 26      RxStompConfig.ts:25
2023 12:58:22 GMT+0800 (台北標準時
間) '<<< MESSAGE\ncontent-length:25
2\nmessage-id:ccf80a80-fd7b-6aff-9
90e-483f8c15a63d-164\nsubscriptio
n:sub-1\ncontent-type:text/plain;c
harset=UTF-8\ndestination:/user/Vi
ewLaunchedItem1/output\ncontent-le
ngth:252\n\n'
 ViewLaunchedItem1.component.ts:85
{"timestamp":"2023-07-
26T04:58:21.445+00:00","status":40
0,"error":"Bad
Request","path":"/item/items/"}
 ViewLaunchedItem1.component.ts:104
getLaunchItems: invoke success
>
```

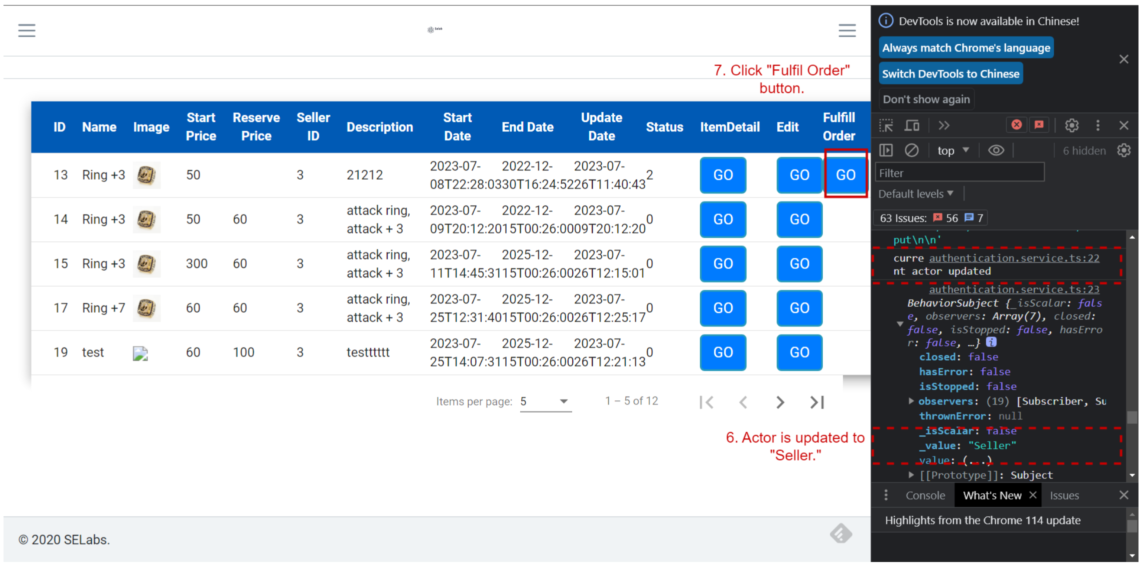
Figure 5.5: Scenario of Authentication, Step 2

doi:10.6342/NTU202303644



```
5\n\n'
Wed Jul 26      RxStompConfig.ts:29
2023 12:58:49 GMT+0800 (台北標準時
間) 'Received data'
Wed Jul 26      RxStompConfig.ts:29
2023 12:58:49 GMT+0800 (台北標準時
間) '<<< MESSAGE\ncontent-length:12
5\nmessage-id:ccf80a80-fd7b-6aff-9
90e-483f8c15a63d-166\nsubscriptio
n:sub-2\ncontent-type:text/plain;c
harset=UTF-8\ndestination:/user/Us
erLogin1/output\ncontent-length:12
5\n\n'
[Warni UserLogin1.component.ts:113
ng] No  operation in outputPort
>
```

Figure 5.6: Scenario of Authentication, Step 3

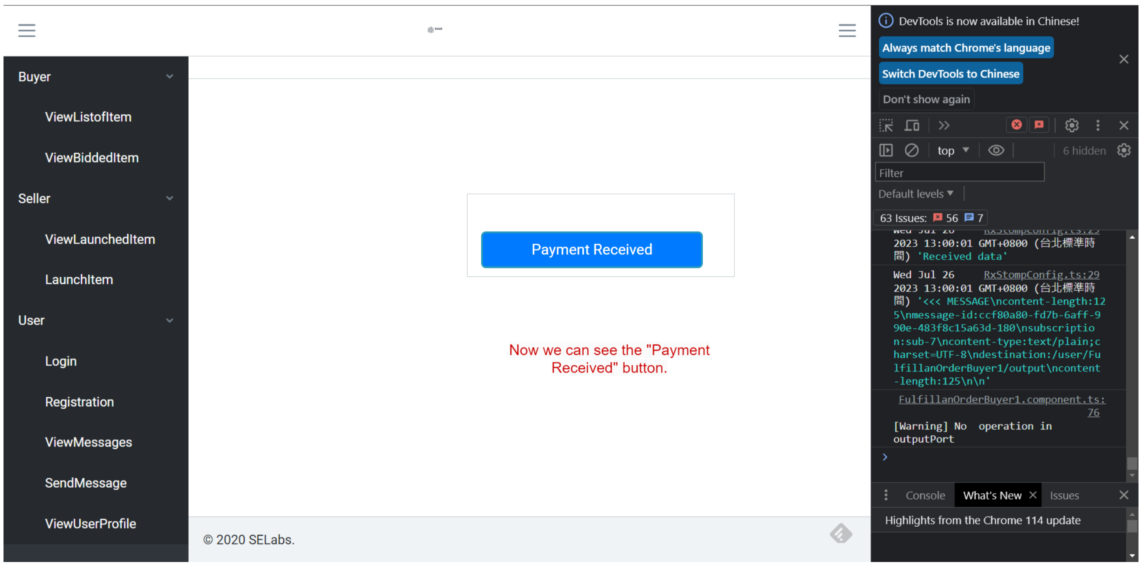

7. Click "Fulfil Order" button.
6. Actor is updated to "Seller."
Items per page: 5
1 – 5 of 12
© 2020 SELabs.
DevTools is now available in Chinese!
Always match Chrome's language
Switch DevTools to Chinese
Don't show again
top
6 hidden
Filter
Default levels
63 Issues: 56 7
Console
What's New
Issues
Highlights from the Chrome 114 update

ID	Name	Image	Start Price	Reserve Price	Seller ID	Description	Start Date	End Date	Update Date	Status	ItemDetail	Edit	Fulfill Order
13	Ring +3		50		3	21212	2023-07-08T22:28:03	2022-12-30T16:24:52	2023-07-26T11:40:43	2	GO	GO	GO
14	Ring +3		50	60	3	attack ring, attack + 3	2023-07-09T20:12:20	2022-12-15T00:26:00	2023-07-09T20:12:20	0	GO	GO	
15	Ring +3		300	60	3	attack ring, attack + 3	2023-07-11T14:45:31	2025-12-15T00:26:00	2023-07-26T12:15:01	0	GO	GO	
17	Ring +7		60	60	3	attack ring, attack + 3	2023-07-25T12:31:40	2025-12-15T00:26:00	2023-07-26T12:25:17	0	GO	GO	
19	test		60	100	3	testttttt	2023-07-25T14:07:31	2025-12-15T00:26:00	2023-07-26T12:21:13	0	GO	GO	

```
put\n\n'
curre authentication.service.ts:22
nt actor updated
      authentication.service.ts:23
  BehaviorSubject {_isScalar: fals
  e, observers: Array(7), closed:
▼ false, isStopped: false, hasErro
  r: false, …}
    closed: false
    hasError: false
    isStopped: false
  ▶ observers: (19) [Subscriber, Su
    thrownError: null
    _isScalar: false
    _value: "Seller"
    value: (...)
  ▶ [[Prototype]]: Subject
```

Figure 5.7: Scenario of Authentication, Step 4

doi:10.6342/NTU202303644



Now we can see the "Payment Received" button.

```
2023 13:00:01 GMT+0800 (台北標準時間) 'Received data'
Wed Jul 26      RxStompConfig.ts:29
2023 13:00:01 GMT+0800 (台北標準時間) '<<< MESSAGE\ncontent-length:125\nmessage-id:ccf80a80-fd7b-6aff-990e-483f8c15a63d-180\nsubscription:sub-7\ncontent-type:text/plain;charset=UTF-8\ndestination:/user/FulfillanOrderBuyer1/output\ncontent-length:125\n\n'
                FulfillanOrderBuyer1.component.ts:76
[Warning] No  operation in outputPort
```

Figure 5.8: Scenario of Authentication, Step 5

doi:10.6342/NTU202303644

Chapter 6

Conclusion and Future Work

The contribution of our research is the proposal of an approach to generate task models represented in CTT form from use case specifications and a use case diagram. Then we can use these CTTs to automatically generate a user interface and a corresponding web application.

In the process of CTT generation, almost every step involves using machine learning models. This drives us to explore new methods to analyze machine learning models in a more intuitive and comprehensible way. So that we can enhance model performance more easily in the future.

Although the current outputs of our trained machine learning models do not meet our expectations and have various issues, the overall process design has been completed. Once we enhance our models' performance, the task model generation process will continue smoothly. Here, we list the problems encountered during the research process:

doi:10.6342/NTU202303644

Lack of use case data: our model's inputs are use case specifications and a use case diagram, which require our model to have a large amount of such data for training. However, much of this kind of data is often restricted to clients or staff, making it hard to access and resulting in the lack of training data.

Inconsistent data formats for use cases: use case specifications and use case diagrams lack a rigorous standard of formats. This causes missing data fields, making it hard to preprocess data.

Various ways to narrate use cases: use cases are usually represented in textual form, so different author will have different narrative structure, level of detail and completeness of the use cases, making it more likely to have misinterpretations.

High system requirements for large language models: with the emergence of new large language models, the demand on hardware resources has grown significantly. For example, GPT-NeoX requires 30GB of VRAM just for execution. Finding suitable machines for running these models has become a challenge.

Limitations of machine learning models: GPT-NeoX has input length limitation of 2048 tokens, making it hard to learn from more examples. Additionally, BertSum requires a large number of training data to achieve better results than GPT-NeoX.

doi:10.6342/NTU202303644

Future Works

- Find more training data for use case specifications and use case diagrams to train our models for better performance.
- Automatically adjust the content and format of the data to match the standards we have set.
- Experiment with different hyper parameter settings for the models we are using to enhance the models' performance.
- Use other machine learning models to avoid the limitations of the models we are using.
- Complete the remaining parts of Task Model Generator to enable generating task models automatically.

doi:10.6342/NTU202303644

Bibliography

[1] Bertsum. https://github.com/nlpyang/PreSumm.

[2] Concur task tree. https://www.w3.org/2012/02/ctt/.

[3] Figma. https://www.figma.com/.

[4] Freemarker. https://freemarker.apache.org/index.html.

[5] Task model. https://www.w3.org/2005/Incubator/model-based-ui/wiki/Task_Model.

[6] M.-H. Hsieh. Construct and bind user interface components. Master's thesis, National Taiwan University, 2021.

[7] K.-C. Kuo. Generate web frontend servers with ui components composition and navigation. Master's thesis, National Taiwan University, 2022.

[8] R.-S. Liou. Auto build user interface from task model. Master's thesis, National Taiwan University, 2023.

[9] Y. Liu and M. Lapata. Text summarization with pretrained encoders. In *Proc. 2019 Conference on Empirical Methods in Natural Language Processing*, 2019.

[10] J.-T. Zeng. Integrate user interface components with themes. Master's thesis, National Taiwan University, 2021.

doi:10.6342/NTU202303644