# COSE474-2024F: Final Project Proposal “Classifying the concept of CS terminology”

**장동윤(Dongyoon Chang)**

## 1. Introduction

### 1.1. Motivation

Computer Science, which is consisted of lots of concepts, is so wide and also rapidly growing. For me, studying all of these extensive concepts and building the whole structure are really hard one. Every terminologies and ways that I’ve learned mess up with each other, which makes me feel confusing and hard to apply to proper area. So, I felt some need to clear up all these terminologies.

### 1.2. Problem Definiton

My problem is solving confused feelings for concepts of CS terminolgies. I should arrange all of them into larger categories, which represents all knowledge of CS. Searching it on Google or else AIs can be our way out. Even making up the whole structure to noting apps like Notion can be a solution. But cherry-picking the information that I really want on Google is really tired. Also, writing all of the information on Notion is really fatigue too. So, I want to make my own prompted AI that can serve my task.

### 1.3. Concise Description for Contribution

I’m gonna contribute for AI academia by analyzing and fine tuning the Language Model, then derive the result how fine tuning can improve the ability of model. We can also compare the performance between zero-shot classification and fine tuning.

## 2. Goal

The goal is to classify all CS terminology into larger categories. At first, I choose several larger categories, which can properly represent the whole vast CS_concepts. Then, I input terms which I want to find bigger concepts.

For example, when I want to find out categories for DI(Dependency Injection), which usually used in OOP languages like Java. I initially suggest some bigger categories like computer_architecture, OS, network, DB, data_structure, algorithm, software_engineering, language, AI/ML, etc. Then, DI might be clustered to language, because it is the technology for some OOP language like Java. It controls the dependency of Java’s objects, which makes developer more easier. When we use Hibernate as input term, it will be clustered to DB.

In order to improve our model’s quality, we need to do some work for it.

- **Batch Datasets**: There is no available dataset for CS terminologies to broader categories in hugging faces, so I will create my own dataset based on my knowledge.
- **Design Prompts**: I will design effective prompts for my model to solve classifying problems.
- **Evaluate model performance**: I’ll evaluate my model’s performance, by comparing with SOTA models, which will be BART(facebook) for this case.

## 3. Pre-trained Model

As I achieve this goal, I’m going to use pre-trained model which is for NLP, labeling datas for text. In my search, I will use BERT for my project, because it can easily apply prompt tuning.

## 4. State-of-the-art methods and baselines

I’m gonna compare my result with BART’s answer, which is one of the State-of-the-art for text clustering, pre-trained one using zero-shot classification.

## 5. Methods

### 5.1. Significance and Novelty

The significance and novelty of this project is that we can automatically classify the input data. Attempts for knowledge management about CS terminology by language model was just a few case.

Also, I used BERT model for inferring language without context, which didn’t attempt for this model.

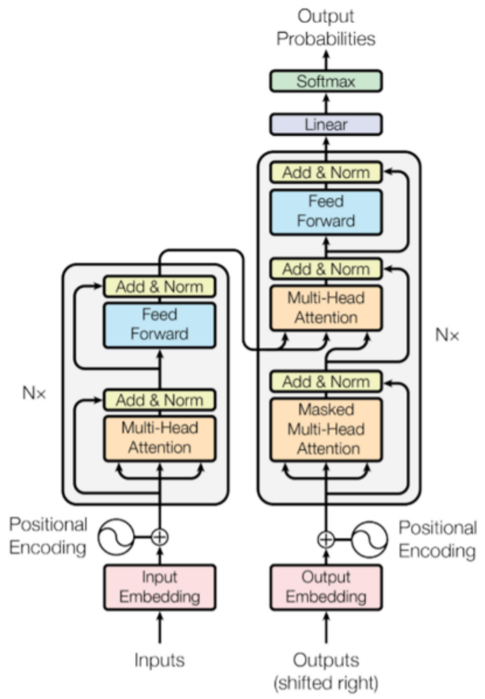


*Figure 1.* BERT

### 5.2. Main Challenges and How I address them

- **Lack of Existing Datasets**: There is no available dataset for CS terminologies to broader categories in hugging faces, so I will create my own dataset based on my knowledge.
- **Complexity of Terminologies**: CS terms may be polymorphic or not be defined easily. In order to solve this, I will define my categories more strictly, which can properly define the whole terms.
- **Effective Prompt Design**: Making really effective prompts is really hard. I'll do several experiments for different prompts to find the most effective one.

### 5.3. Reproducibility— Algorithm Pseudocode

Pseudocode for CS Terminology Classification using BERT

```
Install and Import Lib
Add category, training dataset's term and category (ex: HTTP-Networks), test_terms
Encode category and Map with label
Split and Tokenize the dataset
Load model
Create Train, trainer, Train It
Predict test terms, Print It
```

## 6. Experiments

### 6.1. Dataset

The dataset is composed of CS terms. For fine tuning, I'll add some original training set for fine tuning, for example ”Networks” for ”HTTP”, or ”Databases” for ”MySQL”.

I manually wrote all of the dataset by searching online resources, or based on my knowledge. It ensures a diverse representation of concepts across different CS domains.

Datasets that I chose as follows ”HTTP”, ”TCP/IP”, ”MySQL”, ”Binary Search”, ”QuickSort”, ”Agile Methodology”, ”Python”, ”Linked List”, ”REST API”, ”NoSQL”, ”Linux”, ”Dynamic Programming”, ”PostgreSQL”, ”DNS”, ”Merge Sort”, ”Scrum”, ”JavaScript”, ”Trie”, ”HTTPS”, ”Docker”. I used some datasets on test like HTTP or Docker, in order to keep track of whether the fine tuning precisely applied.

### 6.2. Computing Resources

Google Colab: T4 GPU engine, Ubuntu 22.04.3 LTS, PyTorch 2.5.1+cu121 for model testing. Hugging Face for finding transformer models.

### 6.3. Experimental Design/Setup

- **Training Procedure**:
  - ∗ I applied fine-tuning by adding training dataset and its category too.
- **Baseline for Comparison**:
  - ∗ I compare the performance of my model with BART, which uses zero-shot classification.
- **Evaluation Metrics**:
  - ∗ Evaluate the performance according to the accuracy of the answer.

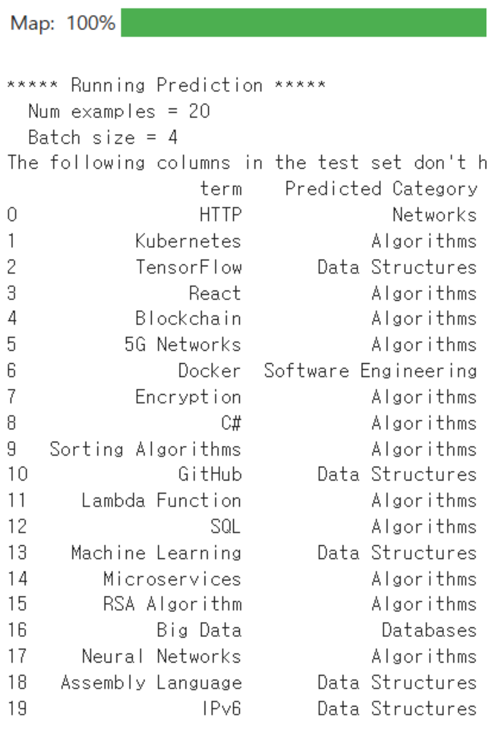
### 6.4. Quantitative Results & Analysis

| | BERT | BART(SOTA) |
|---|---|---|
| Correct count | 7 | 16 |

BERT showed the less performance than SOTA model, even though I attempted to fix the epoch, learning_rate, batch_size and anything else.

### 6.5. Qualitative Results & Analysis

BERT 7

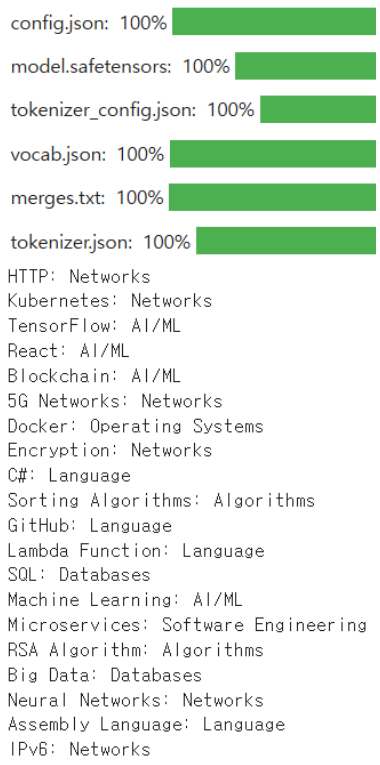


```
Map:  100%

***** Running Prediction *****
  Num examples = 20
  Batch size = 4
The following columns in the test set don't h
                   term    Predicted Category
0                  HTTP              Networks
1            Kubernetes            Algorithms
2            TensorFlow       Data Structures
3                 React            Algorithms
4            Blockchain            Algorithms
5           5G Networks            Algorithms
6                Docker  Software Engineering
7            Encryption            Algorithms
8                    C#            Algorithms
9    Sorting Algorithms            Algorithms
10               GitHub       Data Structures
11      Lambda Function            Algorithms
12                  SQL            Algorithms
13     Machine Learning       Data Structures
14        Microservices            Algorithms
15        RSA Algorithm            Algorithms
16             Big Data             Databases
17      Neural Networks            Algorithms
18    Assembly Language       Data Structures
19                 IPv6       Data Structures
```

*Figure 2.* BERT

BART(SOTA) 16



```
config.json:  100%
model.safetensors:  100%
tokenizer_config.json:  100%
vocab.json:  100%
merges.txt:  100%
tokenizer.json:  100%
HTTP: Networks
Kubernetes: Networks
TensorFlow: AI/ML
React: AI/ML
Blockchain: AI/ML
5G Networks: Networks
Docker: Operating Systems
Encryption: Networks
C#: Language
Sorting Algorithms: Algorithms
GitHub: Language
Lambda Function: Language
SQL: Databases
Machine Learning: AI/ML
Microservices: Software Engineering
RSA Algorithm: Algorithms
Big Data: Databases
Neural Networks: Networks
Assembly Language: Language
IPv6: Networks
```

*Figure 3.* BART

| Model | Correct Terms |
|---|---|
| **BERT** | HTTP, 5G Networks, Docker, Encryption, Sorting Algorithms, RSA Algorithm, Big Data |
| **BART** | HTTP, TensorFlow, 5G Networks, Docker, Encryption, C#, Sorting Algorithms, Lambda Functions, SQL, Machine Learning, Microservices, RSA Algorithm, Big Data, Neural Networks, Assembly Languages, IPv6 |

*Table 1.* Correct Terms for BERT and BART Models

As you can see the results, BERT absolutely failed his attempt to classify terminologies to category. Cases that I attempted each time derived different results for each other, which means that BERT actually didn't train well.

### 6.6. Discussion why the method is unsuccessful

First, my experiment setting might be wrong. Even though I fixed epoch, learning_rate, batch_size for BERT model, it might not be proper value to derive the right result.

Second, the whole structure of experiment had a big problem. There are many CS terms which can not be bound on one category. For example, BlockChain is a data storage technology which stores data in a p2p chain environment. It can be bound on database, network, algorithm categories simultaneously.

Last, BERT isn't actually a fitting model for classifying terms. BERT was originally designed for MLM. It is an abbreviation for "Masked Token Prediction", which predicts masked word for the default context. In this case, we don't give any context for BERT, which makes hard to show it's own performance.

## 7. Future Direction

In order to enhance the model's performance, there are some directions to be proposed:

– **Add and Improve the Dataset (Fine-Tuning)**: More data makes better output. We can improve our model's understanding by adding appropriate datasets.

– **More categories (Hierarchy)**: There are many terms which can be bound with multiple categories, so we may change category structure to other way, like building hierarchy.

– **Apply Prompt Engineering**: Applying prompt engineering in BERT actually doesn't improve performance dramatically, but it can be another solution too.

## References

1. Vaswani, Ashish et al. (2017). *Attention Is All You Need*.
2. TensorFlow.org (2022) *https://www.tensorflow.org/text/tutorials/classify_text_with_bert?utm_source=chatgpt.comhl=ko*.
3. Hugging Face *https://huggingface.co/docs/transformers/model_doc/bert*.