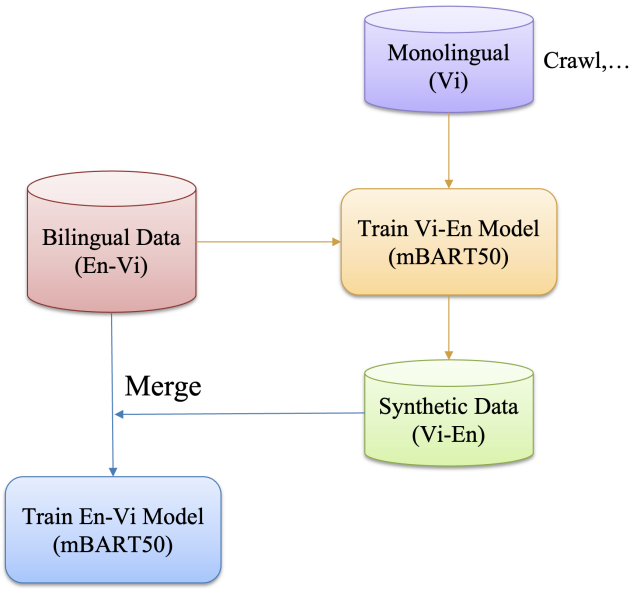
AI VIET NAM – AIO COURSE 2023

Project: Low-Resource Machine Translation

Quoc-Thai Nguyen và Quang-Vinh Dinh
PR-Team: Đăng-Nhã Nguyễn, Minh-Châu Phạm và Hoàng-Nguyên Vũ

Ngày 25 tháng 2 năm 2024

Phần I. Giới thiệu



Hình 1: Kỹ thuật dịch ngược áp dụng cho mô hình dịch Anh - Việt.

Dịch Máy (Machine Translation) với mục đích tự động dịch văn bản từ ngôn ngữ tự nhiên này sang ngôn ngữ tự nhiên khác. Một trong những thách thức lớn hiện nay của các hệ thống dịch là có ít tài nguyên để huấn luyện mô hình. Vì vậy, trong phần này, chúng ta sẽ tập trung vào cải thiện các mô hình dịch với ít tài nguyên (Low-resource machine translation), số lượng các cặp dữ liệu song ngữ hạn chế, ví dụ mô hình dịch huấn luyện với chỉ 20000 cặp câu tiếng anh và tiếng việt. Vì vậy, để cải tiến mô hình dịch trong trường hợp có ít tài nguyên, chúng ta tập trung vào 2 hướng tiếp cận như sau:

1. Hướng 1: Sử dụng mô hình pre-trained mBART50
2. Hướng 2: Sử dụng kỹ thuật dịch ngược (Back-translation) để tăng cường dữ liệu

Phần II. Pre-trained mBART50

Để xây dựng, đánh giá hiệu suất các mô hình chúng ta sử dụng bộ dữ liệu dịch **IWSLT'15 English -Vietnamese** với số lượng mẫu cho training: 133,317 cặp câu song ngữ, tập validation: 1,553 cặp câu song ngữ và tập test: 1,269 cặp câu song ngữ. Chiều dịch sẽ từ tiếng anh sang tiếng việt.

Metric để đánh giá chúng ta sử dụng: ScacreBleu (BLEU)

Trong phần này, chúng ta sẽ sử dụng kỹ thuật fine-tuning để huấn luyện mô hình dịch theo chiều EN-VI.

1. Build Dataset

```
# install libs
!pip install -q transformers sentencepiece datasets accelerate evaluate sacrebleu

# import libs
import os
import numpy as np
import torch
from torch.utils.data import Dataset
from datasets import load_dataset
import evaluate
from transformers import (
    MBart50TokenizerFast,
    AutoModelForSeq2SeqLM,
    DataCollatorForSeq2Seq,
    Seq2SeqTrainingArguments,
    Seq2SeqTrainer
)

# build dataset
class NMTDataset(Dataset):
    def __init__(self, cfg, data_type="train"):
        super().__init__()
        self.cfg = cfg

        self.src_texts, self.tgt_texts = self.read_data(data_type)

        self.src_input_ids = self.texts_to_sequences(self.src_texts)
        self.labels = self.texts_to_sequences(self.tgt_texts)

    def read_data(self, data_type):
        data = load_dataset(
            "mt_eng_vietnamese",
            "iwslt2015-en-vi",
            split=data_type
        )
        src_texts = [sample["translation"][self.cfg.src_lang] for sample in data]
        tgt_texts = [sample["translation"][self.cfg.tgt_lang] for sample in data]
        return src_texts, tgt_texts

    def texts_to_sequences(self, texts):
        data_inputs = self.cfg.tokenizer(
            texts,
            padding='max_length',
            truncation=True,
            max_length=self.cfg.max_len,
            return_tensors='pt'
        )
```

```
        return data_inputs.input_ids

    def __getitem__(self, idx):
        return {
            "input_ids": self.src_input_ids[idx],
            "labels": self.labels[idx]
        }

    def __len__(self):
        return np.shape(self.src_input_ids)[0]
```

2. Tokenizer, Model, Metric

```
class BaseConfig:
    """ base Encoder Decoder config """

    def __init__(self, **kwargs):
        for k, v in kwargs.items():
            setattr(self, k, v)

class NMTConfig(BaseConfig):
    # Data
    src_lang = 'en'
    tgt_lang = 'vi'
    max_len = 75
    add_special_tokens = True

    # Model
    model_name = "facebook/mbart-large-50-many-to-many-mmt"

    # Training
    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
    learning_rate = 5e-5
    train_batch_size = 32
    eval_batch_size = 32
    num_train_epochs = 2
    save_total_limit = 1
    ckpt_dir = f'./mbart50-{src_lang}-{tgt_lang}'
    eval_steps = 1000

    # Inference
    beam_size = 5

cfg = NMTConfig()
cfg.tokenizer = MBart50TokenizerFast.from_pretrained(cfg.model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(cfg.model_name)

# metric
metric = evaluate.load("sacrebleu")

def postprocess_text(preds, labels):
    preds = [pred.strip() for pred in preds]
    labels = [[label.strip()] for label in labels]

    return preds, labels

def compute_metrics(eval_preds):
    preds, labels = eval_preds
    if isinstance(preds, tuple):
```

```
        preds = preds[0]

    preds= np.where(preds != -100, preds, cfg.tokenizer.pad_token_id)
    decoded_preds = cfg.tokenizer.batch_decode(preds, skip_special_tokens=True,
    clean_up_tokenization_spaces=True)

    labels= np.where(labels != -100, labels, cfg.tokenizer.pad_token_id)
    decoded_labels = cfg.tokenizer.batch_decode(labels, skip_special_tokens=True,
    clean_up_tokenization_spaces=True)

    decoded_preds, decoded_labels = postprocess_text(decoded_preds, decoded_labels)

    result = metric.compute(predictions=decoded_preds, references=decoded_labels)
    result = {"bleu": result["score"]}

    prediction_lens = [np.count_nonzero(pred != cfg.tokenizer.pad_token_id) for pred
    in preds]
    result["gen_len"] = np.mean(prediction_lens)
    result = {k: round(v, 4) for k, v in result.items()}

    return result
```

3. Trainer

```
train_dataset = NMTDataset(cfg, data_type="train")
valid_dataset = NMTDataset(cfg, data_type="validation")
test_dataset = NMTDataset(cfg, data_type="test")

training_args = Seq2SeqTrainingArguments(
    predict_with_generate=True,
    evaluation_strategy="steps",
    save_strategy='steps',
    save_steps=cfg.eval_steps,
    eval_steps=cfg.eval_steps,
    output_dir=cfg.ckpt_dir,
    per_device_train_batch_size=cfg.train_batch_size,
    per_device_eval_batch_size=cfg.eval_batch_size,
    learning_rate=cfg.learning_rate,
    save_total_limit=cfg.save_total_limit,
    num_train_epochs=cfg.num_train_epochs,
    load_best_model_at_end=True,
)

data_collator = DataCollatorForSeq2Seq(
    cfg.tokenizer,
    model=model
)

trainer = Seq2SeqTrainer(
    model,
    training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    data_collator=data_collator,
    tokenizer=cfg.tokenizer,
    compute_metrics=compute_metrics
)

trainer.train()
```

4. Training

Sau khi huấn luyện chúng ta thu được kết quả đánh giá như sau:

Step	Training Loss	Validation Loss	Bleu	Gen Len
1000	0.501900	0.599626	31.826500	32.365600
2000	0.493200	0.581667	32.680700	32.416900
3000	0.484500	0.572604	32.876400	32.588700
4000	0.484600	0.561723	33.089600	32.848700
5000	0.473300	0.555238	33.772600	32.662700
6000	0.471100	0.551521	33.693500	32.728900
7000	0.464800	0.545575	33.738100	32.953500
8000	0.453600	0.534770	34.213600	32.956700
9000	0.366800	0.546183	34.096400	32.688700
10000	0.366000	0.546823	33.752300	32.802200
11000	0.361400	0.544369	33.953200	32.760400
12000	0.359000	0.540036	34.301600	32.951100
13000	0.359900	0.535568	34.730100	32.772300
14000	0.355300	0.535669	34.714100	32.805400
15000	0.355800	0.532964	34.919300	32.687200
16000	0.352400	0.530680	34.871400	32.792000

Hình 2: Huấn luyện mô hình mBART50 trên bộ dữ liệu Anh - Việt.

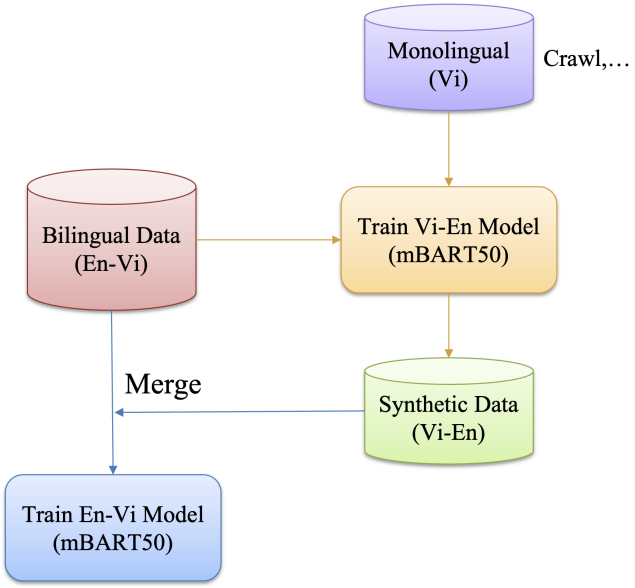
5. Evaluation

```python
def inference(
    text,
    tokenizer,
    model,
    device="cpu",
    max_length=75,
    beam_size=5
    ):
    inputs = tokenizer(
        text,
        padding="max_length",
        truncation=True,
        max_length=max_length,
        return_tensors="pt"
        )
    input_ids = inputs.input_ids.to(device)
    attention_mask = inputs.attention_mask.to(device)
```

```
    model.to(device)

    outputs = model.generate(
        input_ids,
        attention_mask=attention_mask,
        max_length=max_length,
        early_stopping=True,
        num_beams=beam_size,
        length_penalty=2.0
    )

    output_str = tokenizer.batch_decode(outputs, skip_special_tokens=True)

    return output_str

sentence = 'i go to school'
inference(sentence, cfg.tokenizer, model)
```

Phần III. Back-Translation

Trong phần này chúng ta sẽ cải tiến mô hình dịch với kỹ thuật dịch ngược (Back-Translation)



Hình 3: Kỹ thuật dịch ngược áp dụng cho mô hình dịchp.

Kỹ thuật dịch ngược bao gồm các bước sau:

- Bước 1: Xây dựng bộ dữ liệu đa ngữ (Monolingual data) dành cho tiếng việt.
- Bước 2: Huấn luyện mô hình mBART50 với chiều dịch là Việt - Anh.
- Bước 3: Tạo bộ dữ liệu tổng hợp sử dụng mô hình mBART50 Vi-En trên bộ dữ liệu đơn ngữ
- Bước 4: Kết hợp bộ dữ liệu tổng hợp với bộ dữ liệu gốc để huấn luyện mô hình En-Vi.

1. Vietnamese Monolingual Data Preparing

Bộ dữ liệu đơn ngữ dành cho tiếng việt có thể tạo ra bằng cách thu thập từ các trang tin tức,... Ở đây để đơn giản, chúng ta sử dụng tập test của bộ dữ liệu PhoMT lấy phần text của ngôn ngữ tiếng việt (Có thể tải về tại đây).

2. Train Vi-En Model

Chúng ta sử dụng bộ dữ liệu gốc huấn luyện mô hình mBART50 theo chiều Vi-En. Các bước huấn luyện tương tự với phần 2.

2.1. Build Dataset

```python
# build dataset
class NMTDataset(Dataset):
    def __init__(self, cfg, data_type="train"):
        super().__init__()
        self.cfg = cfg

        self.src_texts, self.tgt_texts = self.read_data(data_type)
```

```
        self.src_input_ids = self.texts_to_sequences(self.src_texts)
        self.labels = self.texts_to_sequences(self.tgt_texts)

    def read_data(self, data_type):
        data = load_dataset(
            "mt_eng_vietnamese",
            "iwslt2015-en-vi",
            split=data_type
        )
        src_texts = [sample["translation"][self.cfg.src_lang] for sample in data]
        tgt_texts = [sample["translation"][self.cfg.tgt_lang] for sample in data]
        return src_texts, tgt_texts

    def texts_to_sequences(self, texts):
        data_inputs = self.cfg.tokenizer(
            texts,
            padding='max_length',
            truncation=True,
            max_length=self.cfg.max_len,
            return_tensors='pt'
        )
        return data_inputs.input_ids

    def __getitem__(self, idx):
        return {
            "input_ids": self.src_input_ids[idx],
            "labels": self.labels[idx]
        }

    def __len__(self):
        return np.shape(self.src_input_ids)[0]
```

2.2. Tokenizer, Model, Metric

```
class BaseConfig:
    """ base Encoder Decoder config """

    def __init__(self, **kwargs):
        for k, v in kwargs.items():
            setattr(self, k, v)

class NMTConfig(BaseConfig):
    # Data
    src_lang = 'vi'
    tgt_lang = 'en'
    max_len = 75
    add_special_tokens = True

    # Model
    model_name = "facebook/mbart-large-50-many-to-many-mmt"

    # Training
    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
    learning_rate = 5e-5
    train_batch_size = 32
    eval_batch_size = 32
    num_train_epochs = 2
    save_total_limit = 1
```

```
    ckpt_dir = f'./mbart50-{src_lang}-{tgt_lang}'
    eval_steps = 1000

    # Inference
    beam_size = 5

cfg = NMTConfig()
cfg.tokenizer = MBart50TokenizerFast.from_pretrained(cfg.model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(cfg.model_name)

# metric
metric = evaluate.load("sacrebleu")

def postprocess_text(preds, labels):
    preds = [pred.strip() for pred in preds]
    labels = [[label.strip()] for label in labels]

    return preds, labels

def compute_metrics(eval_preds):
    preds, labels = eval_preds
    if isinstance(preds, tuple):
        preds = preds[0]

    preds= np.where(preds != -100, preds, cfg.tokenizer.pad_token_id)
    decoded_preds = cfg.tokenizer.batch_decode(preds, skip_special_tokens=True,
    clean_up_tokenization_spaces=True)

    labels= np.where(labels != -100, labels, cfg.tokenizer.pad_token_id)
    decoded_labels = cfg.tokenizer.batch_decode(labels, skip_special_tokens=True,
    clean_up_tokenization_spaces=True)

    decoded_preds, decoded_labels = postprocess_text(decoded_preds, decoded_labels)

    result = metric.compute(predictions=decoded_preds, references=decoded_labels)
    result = {"bleu": result["score"]}

    prediction_lens = [np.count_nonzero(pred != cfg.tokenizer.pad_token_id) for pred
    in preds]
    result["gen_len"] = np.mean(prediction_lens)
    result = {k: round(v, 4) for k, v in result.items()}

    return result
```

2.3. Trainer

```
#training
train_dataset = NMTDataset(cfg, data_type="train")
valid_dataset = NMTDataset(cfg, data_type="validation")
test_dataset = NMTDataset(cfg, data_type="test")

training_args = Seq2SeqTrainingArguments(
    predict_with_generate=True,
    evaluation_strategy="steps",
    save_strategy='steps',
    save_steps=cfg.eval_steps,
    eval_steps=cfg.eval_steps,
    output_dir=cfg.ckpt_dir,
    per_device_train_batch_size=cfg.train_batch_size,
    per_device_eval_batch_size=cfg.eval_batch_size,
```

```
    learning_rate=cfg.learning_rate,
    save_total_limit=cfg.save_total_limit,
    num_train_epochs=cfg.num_train_epochs,
    load_best_model_at_end=True,
)
data_collator = DataCollatorForSeq2Seq(
    cfg.tokenizer,
    model=model
)
trainer = Seq2SeqTrainer(
    model,
    training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    data_collator=data_collator,
    tokenizer=cfg.tokenizer,
    compute_metrics=compute_metrics
)
trainer.train()
```

3. Synthetic Data Generation

Chúng ta sử dụng mô hình đã huấn luyện ở mục 2 để dịch các bản dịch ở dữ liệu đơn ngữ.

```
# load vi data from phomt dataset
with open('./test.vi', 'r') as f:
    vi_phomt = f.readlines()
# inference a sentence or bacth
def inference(
    text,
    tokenizer,
    model,
    device="cpu",
    max_length=75,
    beam_size=5
    ):
    inputs = tokenizer(
        text,
        padding="max_length",
        truncation=True,
        max_length=max_length,
        return_tensors="pt"
        )
    input_ids = inputs.input_ids.to(device)
    attention_mask = inputs.attention_mask.to(device)
    model.to(device)
    outputs = model.generate(
        input_ids,
        attention_mask=attention_mask,
        max_length=max_length,
        early_stopping=True,
        num_beams=beam_size,
        length_penalty=2.0
    )
    output_str = tokenizer.batch_decode(outputs, skip_special_tokens=True)
    return output_str[0]

en_phomt_preds = []
for sent in vi_phomt:
    en_sent = inference(sent, cfg.tokenizer, model)
    en_phomt_preds.append(en_sent)
```

4. Train En-VI model

Trong phần này chúng ta kết hợp bộ dữ liệu tổng hợp và bộ dữ liệu song ngữ ban đầu để huấn luyện mô hình.

4.1. Build Dataset

```python
# build dataset
class NMTDataset(Dataset):
    def __init__(self, cfg, src_augs=[], tgt_augs=[], data_type="train"):
        super().__init__()
        self.cfg = cfg

        self.src_texts, self.tgt_texts = self.read_data(data_type)
        if data_type == 'train':
            self.src_texts.extend(src_augs)
            self.tgt_texts.extend(tgt_augs)

        self.src_input_ids = self.texts_to_sequences(self.src_texts)
        self.labels = self.texts_to_sequences(self.tgt_texts)

    def read_data(self, data_type):
        data = load_dataset(
            "mt_eng_vietnamese",
            "iwslt2015-en-vi",
            split=data_type
        )
        src_texts = [sample["translation"][self.cfg.src_lang] for sample in data]
        tgt_texts = [sample["translation"][self.cfg.tgt_lang] for sample in data]
        return src_texts, tgt_texts

    def texts_to_sequences(self, texts):
        data_inputs = self.cfg.tokenizer(
            texts,
            padding='max_length',
            truncation=True,
            max_length=self.cfg.max_len,
            return_tensors='pt'
        )
        return data_inputs.input_ids

    def __getitem__(self, idx):
        return {
            "input_ids": self.src_input_ids[idx],
            "labels": self.labels[idx]
        }

    def __len__(self):
        return np.shape(self.src_input_ids)[0]
```

4.2. Tokenizer, Model, Metric

```python
# config
class BaseConfig:
    """ base Encoder Decoder config """

    def __init__(self, **kwargs):
        for k, v in kwargs.items():
            setattr(self, k, v)
```

```
class NMTConfig(BaseConfig):
    # Data
    src_lang = 'en'
    tgt_lang = 'vi'
    max_len = 75
    add_special_tokens = True

    # Model
    model_name = "facebook/mbart-large-50-many-to-many-mmt"

    # Training
    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
    learning_rate = 5e-5
    train_batch_size = 32
    eval_batch_size = 32
    num_train_epochs = 2
    save_total_limit = 1
    ckpt_dir = f'./mbart50-{src_lang}-{tgt_lang}-backtranslation-2'
    eval_steps = 1000

    # Inference
    beam_size = 5

cfg = NMTConfig()

cfg.tokenizer = MBart50TokenizerFast.from_pretrained(cfg.model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(cfg.model_name)

train_dataset = NMTDataset(cfg, en_phomt_preds, vi_phomt, data_type="train")
valid_dataset = NMTDataset(cfg, data_type="validation")
test_dataset = NMTDataset(cfg, data_type="test")

# metric
metric = evaluate.load("sacrebleu")

def postprocess_text(preds, labels):
    preds = [pred.strip() for pred in preds]
    labels = [[label.strip()] for label in labels]
    return preds, labels

def compute_metrics(eval_preds):
    preds, labels = eval_preds
    if isinstance(preds, tuple):
        preds = preds[0]

    preds= np.where(preds != -100, preds, cfg.tokenizer.pad_token_id)
    decoded_preds = cfg.tokenizer.batch_decode(preds, skip_special_tokens=True, clean_up_tokenization_spaces=True)

    labels= np.where(labels != -100, labels, cfg.tokenizer.pad_token_id)
    decoded_labels = cfg.tokenizer.batch_decode(labels, skip_special_tokens=True, clean_up_tokenization_spaces=True)

    decoded_preds, decoded_labels = postprocess_text(decoded_preds, decoded_labels)

    result = metric.compute(predictions=decoded_preds, references=decoded_labels)
    result = {"bleu": result["score"]}

```

```
    prediction_lens = [np.count_nonzero(pred != cfg.tokenizer.pad_token_id) for pred
    in preds]
    result["gen_len"] = np.mean(prediction_lens)
    result = {k: round(v, 4) for k, v in result.items()}

    return result
```

4.3. Trainer

```
#training
training_args = Seq2SeqTrainingArguments(
    predict_with_generate=True,
    evaluation_strategy="steps",
    save_strategy='steps',
    save_steps=cfg.eval_steps,
    eval_steps=cfg.eval_steps,
    output_dir=cfg.ckpt_dir,
    per_device_train_batch_size=cfg.train_batch_size,
    per_device_eval_batch_size=cfg.eval_batch_size,
    learning_rate=cfg.learning_rate,
    save_total_limit=cfg.save_total_limit,
    num_train_epochs=cfg.num_train_epochs,
    load_best_model_at_end=True,
)
data_collator = DataCollatorForSeq2Seq(
    cfg.tokenizer,
    model=model
)
trainer = Seq2SeqTrainer(
    model,
    training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    data_collator=data_collator,
    tokenizer=cfg.tokenizer,
    compute_metrics=compute_metrics
)
trainer.train()
```

5. Experiment

Kết quả đánh giá trên tập test của các thực nghiệm được mô tả như bảng sau:

Experiment	Model	ScareBLEU
#1	Standard Transformer (Greedy Search)	24.66
#2	BERT-to-BERT (Greedy Search)	25.41
#3	BERT-to-GPT2 (Greedy Search)	23.56
#4	mBART50	34.87
#5	Back-Translation (Monolingual)	35.22

Hình 4: So sánh kết quả thực nghiệm.

Phần 4. Câu hỏi trắc nghiệm

Câu hỏi 1 Mô hình BART có kiến trúc là gì?
a) Transformer-Encoder
b) Transformer-Decoder
c) Transformer-Encoder-Decoder
d) BiLSTM

Câu hỏi 2 Hàm mục tiêu nào sau đây không phải của BART?
a) Token Masking
b) Token Deletion
c) Sentence Permutation
d) Next Sentence Prediction

Câu hỏi 3 Mô hình BART-Base có bao nhiêu lớp encoder?
a) 6
b) 12
c) 24
d) 48

Câu hỏi 4 Mô hình BART-Large có bao nhiêu lớp encoder?
a) 6
b) 12
c) 24
d) 48

Câu hỏi 5 mBART50 được huấn luyện trên bao nhiêu ngôn ngữ?
a) 5
b) 25
c) 50
d) 100

Câu hỏi 6 mBART50 là mô hình tiền huấn luyện đa ngữ cho bài toán nào?
a) Text Summarization
b) Text Classification
c) Question-Answering
d) Machine Translarion

Câu hỏi 7 Tập dữ liệu đơn ngữ tiếng việt được lấy từ bộ dữ liệu nào sau đây?
a) PhoMT
b) IMDB-Review

c) C4
d) ROOT

Câu hỏi 8 Số lượng sample trong bộ dữ liệu đơn ngữ tiếp việt là?
a) 19150
b) 19151
c) 19152
d) 19153

Câu hỏi 9 Mô hình nào sau đây có kiến trúc tương tự mô hình BART?
a) BERT
b) GPT
c) RoBERTa
d) T5

Câu hỏi 10 Mô hình T5 sử dụng tiền tố nào cho bài toán dịch?
a) translate English to Vietnamese
b) translate to Vietnamses from English
c) Cả 2 đáp án đều đúng
d) Cả 2 đáp án đều sai

Câu hỏi 11 Độ đo đánh giá được sử dụng trong phần thực nghiêmh là?
a) F1
b) ScareBLEU
c) Accuracy
d) ROUGE

- Hết -