4. IMPLEMENTASI SISTEM

Pada bab ini akan dibahas tentang implementasi dari desain sistem yang sudah dijelaskan dalam Bab 3 Analisis dan Desain Sistem. Pembahasan akan meliputi implementasi sistem, konfigurasi *hyperparameter*, implementasi *Bidirectional Encoder Representations from Transformers*, serta aplikasi berupa website yang telah dibuat. Tabel 4.1 merupakan hubungan daftar segmen program dan desain sistem.

Tabel 4.1 Hubungan Segmen Program dan Desain Sistem

Segmen	Flowchart	Keterangan
Segmen Program 4.1	3.2	Melakukan pengolahan data yang akan digunakan dalam model untuk mempermudah pembacaan data ke dalam sistem
Segmen Program 4.2	3.3	Melakukan *preprocessing* sederhana terhadap dataset dengan mengubah tiap kata menjadi huruf kecil serta menghapus spasi, tab, dan *newline* yang berlebihan
Segmen Program 4.3	-	Menghubungkan *google colaboratory* dengan *google drive*
Segmen Program 4.4	-	Melakukan inisialisasi *hyperparameter* pada *argument parser extractive summarizer*
Segmen Program 4.5	-	Inisialisasi *main argument parser*
Segmen Program 4.6	-	Membuat *class* BERT data yang berguna untuk menampung *tokenizer*, model, dan *special token* dari BERT
Segmen Program 4.7	-	Fungsi *pad batch collate* pada *dataloder*
Segmen Program 4.8	3.4	Melakukan pembuatan *dataloader* untuk proses pengambilan data yang diperlukan untuk proses *training*, evaluasi, dan *testing*
Segmen Program 4.9	3.4, 3.5	Implementasi proses *training*, evaluasi, dan *testing*

Segmen Program 4.10	3.6	Proses implementasi susunan *transformer encoder* untuk melakukan klasifikasi pada layer *summarization*
Segmen Program 4.11	3.1	Evaluasi dengan menggunakan ROUGE score
Segmen Program 4.12	-	Melakukan prediksi ringkasan berita

4.1 Pengolahan dataset

Pada bagian ini dataset akan dilakukan pengolahan terlebih dahulu dengan mengambil bagian ‘paragraphs’, ‘gold_labels’, dan ‘summary’. Tujuan dari proses ini adalah untuk mempermudah dalam proses training ke dalam model yang dibuat. Tahapan ini akan dilakukan pada setiap data *train*, *dev*, dan *test*. Kemudian, pada bagian ini akan digunakan delimiter ‘<q>’ untuk pemisah antar kalimat pada bagian ‘paragraphs’ dan ‘summary’. Sama halnya dengan bagian ‘gold_labels’ yang dipakai sebagai pemisah antar label ringkasan.

Segmen Program 4.1 Proses pengolahan data

```
import os
import shutil
import json
from time import time
import torch
from datetime import timedelta
from argparse import ArgumentParser

parser = ArgumentParser(description='Data Preparation for Indonesian News
Summarization')
parser.add_argument('--original_data_dir',type=str,default='./indosum_original', help='path
to original dataset')
parser.add_argument('--save_dir', type=str, default='./prepared_data/indo_news_',
help='path to save prepared data')
args = parser.parse_args()

def read(fname):
   data = []
   for line in open(fname, 'r').readlines():
      datum = json.loads(line)
      label=''
      source=''
      target= ''

      # Source document and gold labels
      for idx in range(len(datum['paragraphs'])):
```

```
            for idy in range(len(datum['paragraphs'][idx])):
                for idz in range(len(datum['paragraphs'][idx][idy])):
                    source+=datum['paragraphs'][idx][idy][idz]+' '
                source+='<q>'
                label+=str(int(datum['gold_labels'][idx][idy])) + '<q>'

        # Gold summaries
        for idx in range(len(datum['summary'])):
            for idy in range(len(datum['summary'][idx])):
                target+=datum['summary'][idx][idy] + ' '
            target+='<q>'

        source = source[:-3]
        target = target[:-3]
        label = label[:-3]
        data.append((source, target, label))
    return data

def process(path, save_dir):
    dataset = []
    data = read(path)
    corpus_type = path.split('/')[-1].split('.')[0]
    fold = path.split('/')[-1].split('.')[1]
    start_time = time()
    for idx, datum in enumerate(data):
        source, target, sent_labels = datum
        source = preprocessing(source)
        target = '<q>'.join([' '.join(sent.split()) for sent in target.split('<q>')])  # Untuk
menghilangkan spasi yang berlebihan
        b_data_dict = {"source": source,"labels": sent_labels, "target": target}
        dataset.append(b_data_dict)
        if (idx+1) % 500 == 0:
            end_time = time()
            print(f'{(idx+1)} data processed | {corpus_type}-{fold} | runtime: {end_time-
start_time} seconds')
            start_time = end_time

    if len(dataset) > 0:
        pt_file = save_dir + "{:s}.indonews.bert.pt".format(corpus_type)
        torch.save(dataset, pt_file)

# To speed up the processing data, data will be saved as .pt format
fold = [1,2,3,4,5]
start = time()
for i in fold:
    save_dir = args.save_dir + str(i) + '/' #path to processed data
    print('Create ', save_dir)
    if os.path.exists(save_dir): # check if path exist
        shutil.rmtree(save_dir)  # remove file inside directory recursively
```

```
    os.makedirs(save_dir) # create directory
    process(args.original_data_dir + '/train.0'+str(i)+'.jsonl', save_dir)
    process(args.original_data_dir + '/dev.0'+str(i)+'.jsonl', save_dir)
    process(args.original_data_dir + '/test.0'+str(i)+'.jsonl', save_dir)
end = time()
elapsed_time = timedelta(seconds=end-start)
print(f'Elapsed time: {elapsed_time}')
```

4.2 Preprocessing

Preprocessing yang akan diterapkan pada dataset adalah mengubah tiap kata dalam kalimat menjadi huruf kecil, menghilangkan karakter spasi, tab, dan *newline* yang berlebihan. Pada *preprocessing* ini, delimiter untuk pemisah antar kalimat akan menggunakan '<q>'.

Segmen Program 4.2 Fungsi *preprocessing* data

```
def preprocessing(text):
    # Delimiter of split sentence
    delimiter = '<q>'

    # Case folding (lowercase text)
    text = text.lower()

    # Clean excessive space, tab, and newline
    text = delimiter.join([' '.join(sent.split()) for sent in text.split(delimiter)])

    return text
```

4.3 Konfigurasi google colaboratory

Proses awal yang dilakukan sebelum melakukan implementasi proses training dari model BERT adalah menghubungkan *google colaboratory* dengan *google drive*. Tujuan dari proses ini adalah untuk mempermudah dalam pembacaan data sehingga tidak perlu melakukan inisialisasi data berulang kali pada *google colaboratory*. Proses penghubungan *google colaboratory* dengan *google drive* dapat dilihat pada Segmen Program 4.3.

Segmen Program 4.3 Menghubungkan *google colaboratory* dengan *google drive*

```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
```

Setelah menghubungkan *google colaboratory* dengan *google drive*, dilanjutkan dengan menghubungkan *google colaboratory* dengan GPU. Dalam menghubungkan langkah yang dilakukan adalah dengan cara memilih menu *Runtime* > *Change runtime type* > *Hardware*

accelerator > ubah *hardware accelerator* menjadi GPU. Mengenai keseluruhan *setup* sebelum memulai training pada *google colaboratory* dapat dilakukan dengan *command line*:

Keterangan setup	Command line
Cek GPU yang dipinjamkan *google colaboratory*	!nvidia-smi
Install library pytorch-lightning versi 1.4.9	!pip install pytorch_lightning==1.4.9
Install library transformers	!pip install transformers
Install library untuk *logger wandb*	!pip install wandb
Install dan *setup library pyrouge*	!git clone https://github.com/andersjo/pyrouge.git /content/rouge !git clone https://github.com/bheinzerling/pyrouge /content/pyrouge %cd /content/pyrouge !python setup.py install !pyrouge_set_rouge_path '/content/rouge/tools/ROUGE-1.5.5' %cd /content/rouge/tools/ROUGE-1.5.5/data !rm "WordNet-2.0.exc.db" !perl ./WordNet-2.0-Exceptions/buildExeptionDB.pl ./WordNet-2.0-Exceptions ./smart_common_words.txt ./WordNet-2.0.exc.db !cpan install XML::DOM !python -m pyrouge.test

4.4 Inisialisasi hyperparameter

Sebelum melakukan training, perlu dilakukan inisialisasi terhadap *hyperparameter* yang akan digunakan oleh model. Proses untuk inisialisasi *hyperparameter* beserta *argument* lain dari model *extractive summarizer* akan dilakukan dengan menggunakan *argument parser* seperti yang dapat dilihat pada Segmen Program 4.4. Kemudian, untuk *argument parser* ini akan dipanggil oleh *parent parser* pada bagian *main*. *Argument parser* yang terdapat pada bagian *main* ini akan digunakan sebagai input *argument* dari *trainer pytorch lightning*. Mengenai

argument parser pada bagian *main* dapat dilihat pada Segmen Program 4.5. Mengenai penjelasan untuk tiap *argument* dapat dilihat pada bagian *help* dari tiap *argument*.

Segmen Program 4.4 Inisialisasi *hyperparameter* pada *argument parser extractive summarizer*

```
  @staticmethod
  def add_model_specific_args(parent_parser):
    """Arguments specific to this model"""
    parser = ArgumentParser(parents=[parent_parser],description='Extractive
Summarization on Indonesian News using BERT')
    parser.add_argument("--save_dir", type=str, default="./bert_checkpoints",
help="Directory path to save model")
    parser.add_argument("--ref_summary", type=int, default=0, help="Reference summary
for scoring evaluation (0 = gold label, 1 = gold summary)")
    parser.add_argument(
      "--model_name_or_path",
      type=str,
      default="bert-base-multilingual-uncased",
      help="Path to pre-trained model or shortcut name.",
    )
    parser.add_argument(
      "--model_type",
      type=str,
      default="bert",
      help="Used model type.",
    )
    parser.add_argument(
      "--max_seq_length",
      type=int,
      default=512,
      help="The maximum sequence length of BERT and transformer model.",
    )
    parser.add_argument(
      "--data_path", type=str, default='./prepared_data_basic/indo_news_',
help="Directory containing used data."
    )
    parser.add_argument(
      "--pooling_mode",
      type=str,
      default="sent_rep_tokens",
      help="Convert word vectors to sentence embeddings.",
    )
    parser.add_argument(
      "--num_frozen_steps",
      type=int,
      default=0,
      help="Freeze (don't train) the word embedding model for this many steps.",
    )
    parser.add_argument(
```

```
        "--batch_size",
        default=8,
        type=int,
        help="Batch size per GPU/CPU for training/evaluation/testing.",
    )
    parser.add_argument(
        "--dataloader_num_workers",
        default=2,
        type=int,
        help="""The number of workers to use when loading data. A general place to
        start is to set num_workers equal to the number of CPU cores used machine.""",
    )
    parser.add_argument(
        "--no_use_token_type_ids",
        action="store_true",
        help="Set to not train with `token_type_ids`.",
    )
    parser.add_argument(
        "--classifier",
        type=str,
        default="transformer_position",
        help="""Which classifier/encoder to use to reduce the hidden dimension of the
sentence vectors""",
    )
    parser.add_argument(
        "--classifier_dropout",
        type=float,
        default=0.1,
        help="The value for the dropout layers in the classifier.",
    )
    parser.add_argument(
        "--classifier_transformer_num_layers",
        type=int,
        default=2,
        help='The number of layers for the `transformer` classifier.',
    )
    parser.add_argument(
        "--train_name",
        type=str,
        default="train",
        help="name for set of training files on disk.",
    )
    parser.add_argument(
        "--val_name",
        type=str,
        default="dev",
        help="name for set of validation files on disk.",
    )
    parser.add_argument(
```

```
        "--test_name",
        type=str,
        default="test",
        help="name for set of testing files on disk.",
    )
    parser.add_argument(
        "--test_k",
        type=int,
        default=3,
        help="The `k` parameter to chose top k predictions from the model for evaluation
scoring (default: 3)",
    )
    parser.add_argument(
        "--n_gram_blocking",
        type=int,
        default=3,
        help="number of n-gram blocking for testing"
    )
    parser.add_argument(
        "--no_test_block_ngrams",
        action="store_true",
        help="Disable n-gram blocking when calculating ROUGE scores during testing.",
    )

    return parser
```

Segmen Program 4.5 Inisialisasi *main argument parser*

```
import logging
from pytorch_lightning import Trainer
from extractive import ExtractiveSummarizer
from argparse import ArgumentParser
from pytorch_lightning.loggers import WandbLogger
from pytorch_lightning.callbacks import LearningRateMonitor
from pytorch_lightning import seed_everything


logger = logging.getLogger(__name__)

def main(args):
  if args.seed:
    seed_everything(args.seed, True)
    args.deterministic = True

  model = ExtractiveSummarizer(hparams=args)
  lr_logger = LearningRateMonitor()

  type_used_data = args.data_path.split('/')[-2].split('_')[-1]
  if args.no_use_token_type_ids:
```

```
        temp_token_type_ids = "no-token-type-ids"
    else:
        temp_token_type_ids = "use-token-type-ids"
    temp_pooling_mode = args.pooling_mode.replace('_','-')

    if args.use_logger == "wandb":
        wandb_name =
f"{args.model_name_or_path}_{type_used_data}_{temp_token_type_ids}_{temp_pooling_
mode}_{args.classifier}_{args.classifier_transformer_num_layers}_{args.seed}_{args.learning
_rate}_{args.classifier_dropout}_{args.batch_size}_{args.max_epochs}"
        wandb_logger = WandbLogger(
            name=wandb_name,project=args.wandb_project, log_model=(not
args.no_wandb_logger_log_model)
        )
        args.logger = wandb_logger

    args.callbacks = [lr_logger]
    trainer = Trainer.from_argparse_args(args)

    if args.do_train:
        trainer.fit(model)
    if args.do_test:
        trainer.test(model)


if __name__ == "__main__":
    parser = ArgumentParser(add_help=False)
    parser.add_argument(
        "--default_root_dir", type=str, default=None, help="Default path for logs and weights.",
    )
    parser.add_argument(
        "--weights_save_path",
        type=str,
        default=None,
        help="""Where to save weights if specified. Will override `--default_root_dir` for
        checkpoints only.""",
    )
    parser.add_argument(
        "--learning_rate",
        default=1e-5,
        type=float,
        help="The initial learning rate for the optimizer.",
    )
    parser.add_argument(
        "--min_epochs",
        default=1,
        type=int,
        help="Limits training to a minimum number of epochs",
    )
```

```
    parser.add_argument(
        "--max_epochs",
        default=4,
        type=int,
        help="Limits training to a max number number of epochs",
    )
    parser.add_argument(
        "--min_steps",
        default=None,
        type=int,
        help="Limits training to a minimum number number of steps",
    )
    parser.add_argument(
        "--max_steps",
        default=None,
        type=int,
        help="Limits training to a max number number of steps",
    )
    parser.add_argument(
        "--accumulate_grad_batches",
        default=1,
        type=int,
        help="""Accumulates grads every k batches.""",
    )
    parser.add_argument(
        "--check_val_every_n_epoch",
        default=1,
        type=int,
        help="Check val every n train epochs.",
    )
    parser.add_argument(
        "--gpus",
        default=1,
        type=int,
        help="Number of GPUs to train on or Which GPUs to train on. (-1 = all gpus, 1 = only
using one GPU)",
    )
    parser.add_argument(
        "--gradient_clip_val", default=1.0, type=float, help="Gradient clipping value (default
gradient clipping algorithm is set to 'norm' and clip global norm to <=1.0)"
    )
    parser.add_argument(
        "--fast_dev_run",
        action="store_true",
        help="Runs 1 batch of train, test and val to find any bugs (ie: a sort of unit test).",
    )
    parser.add_argument(
        "--limit_train_batches",
        default=1.0,
```

```
        type=float,
        help="How much of training dataset to check. Useful when debugging or testing
something that happens at the end of an epoch.",
    )
    parser.add_argument(
        "--limit_val_batches",
        default=1.0,
        type=float,
        help="How much of validation dataset to check. Useful when debugging or testing
something that happens at the end of an epoch.",
    )
    parser.add_argument(
        "--limit_test_batches",
        default=1.0,
        type=float,
        help="How much of test dataset to check.",
    )
    parser.add_argument(
        "--precision",
        type=int,
        default=32,
        help="Full precision (32). Can be used on CPU, GPU or TPUs.",
    )
    parser.add_argument(
        "--seed",
        type=int,
        default=1,
        help="Seed for reproducible results and fold id.",
    )
    parser.add_argument(
        "--profiler",
        default="simple",
        type=str,
        help="To profile individual steps during training and assist in identifying bottlenecks.",
    )
    parser.add_argument(
        "--progress_bar_refresh_rate",
        default=50,
        type=int,
        help="How often to refresh progress bar (in steps).",
    )
    parser.add_argument(
        "--num_sanity_val_steps",
        default=0,
        type=int,
        help="Sanity check runs n batches of val before starting the training routine. This
catches any bugs in your validation without having to wait for the first validation check.",
    )
    parser.add_argument(
```

```
      "--val_check_interval",
      default=1.0,
      help="How often within one training epoch to check the validation set. Can specify as
float or int. Use float to check within a training epoch. Use int to check every n steps
(batches).",
   )
   parser.add_argument(
      "--use_logger",
      default="wandb",
      type=str,
      help="Which program to use for logging. Default to `wandb`.",
   )
   parser.add_argument(
      "--wandb_project",
      default="skripsi",
      type=str,
      help="The wandb project to save training runs.",
   )
   parser.add_argument(
      "--do_train", action="store_true", help="Run the training procedure."
   )
   parser.add_argument(
      "--do_test", action="store_true", help="Run the testing procedure."
   )
   parser.add_argument(
      "--load_checkpoint",
      default=None,
      type=str,
      help="Loads the model weights and hyperparameters from a given checkpoint.",
   )
   parser.add_argument(
      "--no_wandb_logger_log_model",
      action="store_true",
      help="Only applies when using the `wandb` logger. Set this argument to NOT save
checkpoints in wandb directory to upload to W&B servers.",
   )
   parser.add_argument(
      "--adam_epsilon", default=1e-8, type=float, help="Epsilon for Adam optimizer.",
   )
   parser.add_argument("--weight_decay", default=1e-2, type=float, help="weight decay for
adam")
   parser.add_argument(
      "--optimizer_type",
      default="adamw",
      type=str,
      help="""Which optimizer to use: `adamw` (default)""",
   )
   parser.add_argument(
      "--warmup_steps",
```

```
        default=0,
        type=int,
        help="Linear warmup over warmup_steps.",
    )
    parser.add_argument(
        "--use_scheduler",
        default="linear",
        type=str,
        help="`linear`: Use a linear schedule that inceases linearly over `--warmup_steps` to `--
learning_rate` then decreases linearly for the rest of the training process.",
    )
    parser.add_argument(
        "--log",
        dest="logLevel", # name of the attribute to be added to the object
        default="INFO",
        choices=["DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL"],
        help="Set the logging level (default: 'Info').",
    )

    parser = ExtractiveSummarizer.add_model_specific_args(parser)
    main_args = parser.parse_args()

    # Setup logging config
    logging.basicConfig(
        format="%(asctime)s|%(name)s|%(levelname)s> %(message)s",
        level=logging.getLevelName(main_args.logLevel),
    )

    # Train and Test
    main(main_args)
```

4.5 Bert Data dan Dataloader

Setelah melakukan inisialisasi untuk *hyperparameter*, akan dilakukan pembuatan *class* BertData yang akan menyimpan atribut dan fungsi-fungsi yang akan digunakan pada model BERT seperti *tokenizer*, model, token spesial, dan token id yang digunakan. Proses pembuatan *class* BertData dapat dilihat pada Segmen Program 4.6. Kemudian, BertData yang telah dibuat akan digunakan untuk mendapatkan *input feature* yang diperlukan untuk melatih model. *Input feature* yang akan digunakan ke dalam model BERT adalah setelah dilakukan pembuatan *dataloader*. Untuk menghasilkan *input feature* pada saat proses pembuatan *dataloader* akan digunakan fungsi *pad batch collate* karena input data yang dimasukkan berupa *list dictionary*. Fungsi *pad batch collate* dapat dilihat pada Segmen Program 4.7. Kemudian, mengenai proses pembuatan *dataloader* untuk training, validasi, dan testing dapat dilihat pada Segmen Program 4.8.

Segmen Program 4.6 BertData

```
class BertData():
   def __init__(self, pre_trained_bert_model):
      self.model = BertModel.from_pretrained(pre_trained_bert_model)
      self.tokenizer = BertTokenizer.from_pretrained(pre_trained_bert_model)
      self.sep_token = '[SEP]'
      self.cls_token = '[CLS]'
      self.pad_token = '[PAD]'
      self.sep_vid = self.tokenizer.vocab[self.sep_token]
      self.cls_vid = self.tokenizer.vocab[self.cls_token]
      self.pad_vid = self.tokenizer.vocab[self.pad_token]

   def get_input_features(self, source, target, labels, min_seq_length=1,
max_seq_length=512, delimiter='<q>'):
      source = source.split(delimiter)
      original_src_txt = source

      source = [sent.strip() for sent in source]
      idxs = [i for i, s in enumerate(source) if (len(s) >= min_seq_length)]

      labels = labels.split('<q>')
      labels = [int(l) for l in labels]

      tokenized_source = self.tokenizer.tokenize(self.sep_token.join(source))
      temp = []
      tokens = []
      flag = True
      for sub_token in tokenized_source:
         if flag:
            tokens.append(self.cls_token)
            flag = False

         tokens.append(sub_token)

         if sub_token == self.sep_token:
            temp.append(tokens)
            tokens = []
            flag = True

      # Check exceeded length (max. token length = 512)
      res = []
      total_len = 0
      for idx, l in enumerate(temp):
         total_len+=len(l)
         if total_len > max_seq_length:
            break
         res.append(temp[idx])
```

```
        source_subtokens = [t for s in res for t in s]
        input_ids = self.tokenizer.convert_tokens_to_ids(source_subtokens)

        # Segments ids / Token type ids
        _segs = [-1] + [i for i, t in enumerate(input_ids) if t == self.sep_vid]
        segs = [_segs[i] - _segs[i - 1] for i in range(1, len(_segs))]
        segments_ids = []
        for i, s in enumerate(segs):
            if (i % 2 == 0):
                segments_ids += s * [0]
            else:
                segments_ids += s * [1]

        cls_ids = [i for i, t in enumerate(input_ids) if t == self.cls_vid]
        labels = labels[:len(cls_ids)]
        source = [original_src_txt[i] for i in idxs]

        return input_ids, segments_ids, labels, cls_ids, source, target

class DatasetIndoNews(torch.utils.data.Dataset):
    def __init__(self, doc, bert, max_seq_length=512):
        super(DatasetIndoNews,self).__init__()
        self.doc = doc
        self.bert = bert
        self.features = self.get_input_features(doc)

    def get_input_features(self, doc):
        input_features = []
        for item in doc:
            input_ids, token_type_ids, labels, cls_ids, source, target =
self.bert.get_input_features(item['source'], item['target'], item['labels'])
            bert_features = {"input_ids": input_ids, "token_type_ids": token_type_ids,
                    "labels": labels, "sent_rep_token_ids": cls_ids,
                    "source":source, "target":target}
            input_features.append(bert_features)
        return input_features

    def get_bert(self):
        return self.bert

    def get_doc(self, idx):
        return self.doc[idx]

    def get_len_doc(self):
        return len(self.doc)

    def __getitem__(self, idx):
        return self.features[idx]
```

```
    def __len__(self):
        return len(self.features)
```

Segmen Program 4.7 Fungsi *pad batch collate*

```
def pad_batch_collate(batch):
    elem = batch[0]
    final_dictionary = {}

    # Iterate through all key dictionary
    for key in elem:

        # For each data key in batch append to list of feature
        feature_list = [d[key] for d in batch]
        if key == "sent_rep_token_ids":

            feature_list = pad(feature_list, -1)
            sent_rep_token_ids = torch.tensor(feature_list, dtype=torch.long)

            sent_rep_mask = ~(sent_rep_token_ids == -1)
            sent_rep_token_ids[sent_rep_token_ids == -1] = 0

            final_dictionary["sent_rep_token_ids"] = sent_rep_token_ids
            final_dictionary["sent_rep_mask"] = sent_rep_mask
            continue
        if key == "input_ids":
            input_ids = feature_list

            # Attention
            # The mask has 1 for real tokens and 0 for padding tokens. Only real
            # tokens are attended to.
            attention_mask = [[1] * len(ids) for ids in input_ids]

            input_ids_width = max([len(ids) for ids in input_ids])
            input_ids = pad(input_ids, 0, width=input_ids_width)
            input_ids = torch.tensor(input_ids, dtype=torch.long)

            attention_mask = pad(attention_mask, 0)
            attention_mask = torch.tensor(attention_mask, dtype=torch.long)

            final_dictionary["input_ids"] = input_ids
            final_dictionary["attention_mask"] = attention_mask

            continue

        if key in ("source", "target"):
            final_dictionary[key] = feature_list
            continue
```

```
        if key in ("labels", "token_type_ids"):
            feature_list = pad(feature_list, 0)

        feature_list = torch.tensor(feature_list, dtype=torch.long)
        final_dictionary[key] = feature_list

    return final_dictionary
```

Segmen Program 4.8 Proses pembuatan *dataloader*

```
    def train_dataloader(self):
        if self.train_dataloader_object:
            return self.train_dataloader_object
        if not hasattr(self,"datasets"):
            self.prepare_data()
        self.global_step_tracker = 0

        train_dataset = self.datasets[self.hparams.train_name]
        train_dataloader = DataLoader(
            train_dataset,
            num_workers=self.hparams.dataloader_num_workers,
            batch_size=self.hparams.batch_size,
            collate_fn=self.pad_batch_collate,
            shuffle=True
        )
        self.train_dataloader_object = train_dataloader
        return train_dataloader

    def val_dataloader(self):
        valid_dataset = self.datasets[self.hparams.val_name]
        valid_dataloader = DataLoader(
            valid_dataset,
            num_workers=self.hparams.dataloader_num_workers,
            batch_size=self.hparams.batch_size,
            collate_fn=self.pad_batch_collate,
            shuffle=False
        )
        return valid_dataloader

    def test_dataloader(self):
        test_dataset = self.datasets[self.hparams.test_name]
        test_dataloader = DataLoader(
            test_dataset,
            num_workers=self.hparams.dataloader_num_workers,
            batch_size=self.hparams.batch_size,
            collate_fn=self.pad_batch_collate,
            shuffle=False
        )
        return test_dataloader
```

Mengenai salah satu contoh *input feature* yang akan diteruskan ke dalam model BERT dapat dilihat pada Segmen Data 4.1. Lalu, untuk keterangan lebih lanjut tentang *input feature* BERT *summarization* yang dihasilkan pada *dataloader* dapat dilihat pada Tabel 4.2.

Segmen Data 4.1 Contoh *input features* BERT untuk proses training

```
{
'input_ids': [3, 22087,  4403,    17,  4403,  9282, 19433,  1836,  2699,  3761,
        43,  1485,  6887,    18,    4,    3, 19433, 22087, 20872,  3761,
        43,    28,    18,    20,    12,  5446,    13,  1709,  3956, 26997,
     22958,  1889,  2115, 12422,  1620,  4766,    18,    4,    3,  1684,
      1841,  4440,  2390,  1485,  3956,  7300,  1540,  4433,  1545,  2699,
      2442,    16,  2936,  1560,  6368, 14664,  1501, 17002,  7756,    18,
         4,    3, 22087, 20872,  3761,    43,    28,    18,    20,    12,
      5446,    13,  5346,  6672,    65,   962,  1534,    12, 14588,   946,
        66,  8040, 13673,  4758,    13,    28, 16870,  1501,  6099,  1617,
     18098, 11795,  9091,  1502,  8596, 10222,  6113,   959, 25799,  1497,
      2528,  3695,  5750, 14214,  4831,  1485,    21,    18,    24,  7582,
       951,    18,    4,    3,  1624,  1560,  6287,  1519,    16,  1684,
      2778,  6944,    22, 17964,  1501, 10529,  7163,  2635,  2542, 17964,
      1497,  1708,  1485,  1562, 21332,  1545,  5190,  8689,   933,  1967,
     25804, 17964,    18,    4,    3, 19433,  1497,  1798,  1777, 21910,
      1485,  6887,  1540,  3375,  1545,  5052, 18279,    27,    18,    21,
     19685,  1534,   930,    16,  1501,  3669,  1713,  5190,  3728, 23278,
        18,    4,    3,  4433,  1545,  3956,  4766,    16, 20872,  3761,
        43,    28,    18,    20,    12,  5446,    13,  2101,  4440,  1485,
      1823,  7756,    16,  1485,  2420,  2338,  1677,  6668,  7756,  2408,
        28,  3349,  1545,  1675, 10486,  7885,    48,    19,    21,    18,
        29,    16,  1501,  6869,  7756,  2616,    25,  3349,    18,    4,
         3,  1624,  1797,  2873,    16,  3956, 20872,  3761, 12422,  2461,
      4112,  1545,  7756,  3615,    25,  3349,  2327, 15378, 15900,  1501,
      7756,  2616,  2616,  1821,    22,  3349,    18,    4,    3,  4153,
      1704,  1533,  7056,  1559, 20872,  3761,    43,    28,    18,    20,
        12,  5446,    13,  5408,    24,    49, 12538,   929,    16,  7371,
        17,  3801,    43,    19,    44,    19,    49,    19,    56,    16,
     27079,    24,    18,    22,    16,  7360, 12609,    23,    18,    25,
      6546,    16,  1501, 22949,    15, 31318,  4294,   944,    18,    4,
         3, 19433,  7542, 18100,    18,    21,  1060, 10002,   947,    18,
        21,  1060,   963,    18,    29,  6546,  1501,  3314,  5305,   947,
      7585,  1487,  1540,  2891,  1716,  8666,  1501,  3314,  8036,  3956,
      1620,  4766,    16,  3464,  1686, 14664,  1497,  4474,  1614, 16740,
     21395,  2071,  1497, 17834,  1614,  1716,  1819,    18,    4,    3,
      2530,  3524, 22087,    18,    4,    3, 11985,  5470, 25852,    18,
      2593,  1581, 17440,  3524, 14249,  1562,  1501,  9602,  3279,    18,
         4,    0,    0,    0,    0,    0,    0,    0,    0,    0,
```

```
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0],
'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
'labels': [0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
'sent_rep_token_ids': [0, 15, 38, 61, 113, 144, 172, 220, 248, 300, 349, 355,   0,   0,
       0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
'sent_rep_mask': [True, True, True, True, True, True, True, True, True, True, True, True,
False, False, False, False, False, False, False, False, False, False, False, False, False, False,
False]
}
```

Tabel 4.2 *Input features* model BERT untuk proses training

#	Feature	Keterangan
1	input_ids	id token dari model BERT yang telah diinisialisasi sebelumnya oleh pembuat model. Pada input_ids ini juga akan berisi id dari *special token* [CLS], [SEP], dan [PAD]
2	attention_mask	Berisi *integer* berupa 0 dan 1 untuk membedakan token *padding* dan token sebenarnya dari id token. 0 merupakan token *padding* sedangkan 1 merupakan token sebenarnya (selain token *padding*)
3	token_type_ids	Berisi *integer* berupa 0 dan 1 untuk membedakan antara kalimat posisi indeks ganjil dan genap.
4	labels	Berupa list yang berisi integer berupa 0 dan 1 sebagai label untuk referensi ekstraktif
5	sent_rep_token_ids	Berisi *integer* berupa indeks token untuk *special token* [CLS]
6	sent_rep_mask	Berisi *boolean* berupa True dan False untuk membedakan token *padding* dan token sebenarnya dari indeks *special token* [CLS]. False menunjukkan token *padding* sedangkan True menunjukkan token sebenarnya (selain token *padding*)

4.6 Training dan Testing

Proses training dimulai dengan menginisialisasikan class *trainer* dari *pytorch lightning* dengan input *main argument parser* dari Segmen Program 4.5. *Pytorch lightning* akan menampung model, *optimizer*, dan *train/val/test step*. Kemudian, *pytorch lightning* akan

memulai proses training dengan mendefinisikan *action* yang diperlukan untuk proses training. List *action* yang digunakan pada modul *pytorch lightning* terdiri atas:

Action	Keterangan
setup	Untuk melakukan pengaturan sebelum proses *training* dan *testing*
prepare_data	Menyiapkan data yang akan digunakan sebelum diteruskan ke dalam *dataloader*
train_dataloader	Membuat *dataloader* untuk pelatihan model
val_dataloader	Membuat *dataloader* untuk melakukan evaluasi dari proses *training*
test_dataloader	Membuat *dataloader* untuk melakukan *testing* dari model yang sudah dilatih
configure_optimizers	Melakukan konfigurasi untuk *optimizer* dan *scheduler* yang dipakai untuk melatih model
training_step	Melakukan proses *training* menggunakan *optimizer* dan *scheduler* yang sudah dikonfigurasikan pada configure_optimizers
training_epoch_end	Akhir tiap *epoch training* untuk menerima output dari training_step
validation_step	Melakukan proses validasi/evaluasi dari training
validation_epoch_end	Akhir tiap *epoch* validasi/evaluasi untuk menerima output dari validation_step
compute_loss	Melakukan perhitungan *loss* antara output model dengan label target
test_step	Melakukan proses *testing* dari model yang sudah dilatih
test_epoch_end	Akhir tiap *epoch* test untuk menerima output dari test_step
foward	Melakukan *forward* dari model dengan meneruskan input ke dalam model untuk dicari tahu nilai dari outputnya

Selain itu, *command* yang perlu dilakukan berulang kali atau dikenal dengan istilah *boilerplate* akan ditangani secara otomatis oleh *pytorch lightning* untuk membantu mempermudah dalam melakukan penelitian sehingga peneliti dapat fokus terhadap bagian utama dari proses *training* dan *testing*. Beberapa *command* tersebut diantaranya:

Command code	Keterangan
.to(device)	Meletakkan *batch* dan komputasi ke dalam *device* yang digunakan
.set_grad_enabled()	Mengaktifkan gradien untuk proses training
.train()	Mengubah ke mode training
.eval()	Mengubah ke mode eval
.zero_grad()	Mengatur gradien untuk semua parameter model menjadi nol
.backward()	Melakukan *backward* untuk menghitung gradien dalam model
.step()	Melakukan *update parameter* dari model

Pada setiap akhir *epoch training* akan dilakukan validasi atau evaluasi yang bertujuan untuk mengetahui seberapa baik model belajar. Mengenai segmen program yang digunakan untuk melakukan keseluruhan proses *training*, evaluasi, dan *testing* pada model yang sudah dibuat dapat dilihat pada Segmen Program 4.9.

Segmen Program 4.9 Implementasi proses *training*, *validasi*, dan *testing*

```
def forward(
      self,
      input_ids,
      attention_mask,
      sent_rep_mask=None,
      token_type_ids=None,
      sent_rep_token_ids=None,
      **kwargs,
   ):
      inputs = {
         "input_ids": input_ids,
         "attention_mask": attention_mask,
      }
```

```
        if not self.hparams.no_use_token_type_ids:
            inputs["token_type_ids"] = token_type_ids

        outputs = self.word_embedding_model(**inputs, **kwargs)
        word_vectors = outputs[0]

        sents_vec, mask = self.pooling_model(
            word_vectors=word_vectors,
            sent_rep_token_ids=sent_rep_token_ids,
            sent_rep_mask=sent_rep_mask,
        )

        sent_scores = self.encoder(sents_vec, mask)
        return sent_scores, mask

    def unfreeze_word_embedding_model(self):
        for param in self.word_embedding_model.parameters():
            param.requires_grad = True

    def freeze_word_embedding_model(self):
        for param in self.word_embedding_model.parameters():
            param.requires_grad = False

    def compute_loss(self, outputs, labels, mask):
        loss = self.loss_func(outputs, labels.float())

        # Set all padding values to zero
        loss = loss * mask.float()

        # Add up all the loss values for each sequence (including padding because
        # padding values are zero and thus will have no effect)
        sum_loss_per_sequence = loss.sum(dim=1)

        # Count the number of losses that are not padding per sequence
        num_not_padded_per_sequence = mask.sum(dim=1).float()

        # Find the average loss per sequence
        average_per_sequence = sum_loss_per_sequence / num_not_padded_per_sequence

        # Get the sum of the average loss per sequence
        sum_avg_seq_loss = average_per_sequence.sum()

        # Get the mean of `average_per_sequence`
        batch_size = average_per_sequence.size(0)
        mean_avg_seq_loss = sum_avg_seq_loss / batch_size

        return mean_avg_seq_loss
```

```
    def setup(self, stage):
        if stage == "fit":
            self.word_embedding_model = self.bert.get_model_with_config(
                self.hparams.model_name_or_path,
model_config=self.word_embedding_model.config
            )
            if self.checkpoint is not None:
                self.load_state_dict(self.checkpoint['model_state_dict'])
                self.epoch = self.checkpoint['epoch'] + 1
                logger.info("Epoch start from %s", self.epoch)

                self.list_train_loss_epoch = self.checkpoint['train_histories']['loss']
                self.list_val_loss_epoch = self.checkpoint['val_histories']['loss']
                self.min_loss = min(self.list_val_loss_epoch)
            else:
                logger.info("Training without checkpoint")

        if stage == "test":
            if self.checkpoint is None:
                logger.info("Need to specify path checkpoint model to test model")
                sys.exit(1)
            self.load_state_dict(self.checkpoint['model_state_dict'])


    def prepare_data(self):
        datasets = {}
        data_splits = [
            self.hparams.train_name,
            self.hparams.val_name,
            self.hparams.test_name,
        ]

        for corpus_type in data_splits:
            full_path = self.hparams.data_path + str(self.hparams.seed) + "/" + corpus_type +
".indonews.bert.pt"
            torch_data = torch.load(full_path)
            data = [x for x in torch_data]
            max_seq_length = min(round(get_average_length(self.bert, data)),
self.hparams.max_seq_length)
            datasets[corpus_type] = DatasetIndoNews(data, self.bert, max_seq_length)

        self.datasets = datasets
        self.pad_batch_collate = pad_batch_collate


    def train_dataloader(self):
        if self.train_dataloader_object:
            return self.train_dataloader_object
        if not hasattr(self,"datasets"):
```

```
        self.prepare_data()
    self.global_step_tracker = 0

    train_dataset = self.datasets[self.hparams.train_name]
    train_dataloader = DataLoader(
        train_dataset,
        num_workers=self.hparams.dataloader_num_workers,
        batch_size=self.hparams.batch_size,
        collate_fn=self.pad_batch_collate,
        shuffle=True
    )
    self.train_dataloader_object = train_dataloader
    return train_dataloader

  def val_dataloader(self):
    valid_dataset = self.datasets[self.hparams.val_name]
    valid_dataloader = DataLoader(
        valid_dataset,
        num_workers=self.hparams.dataloader_num_workers,
        batch_size=self.hparams.batch_size,
        collate_fn=self.pad_batch_collate,
        shuffle=False
    )
    return valid_dataloader

  def test_dataloader(self):
    test_dataset = self.datasets[self.hparams.test_name]
    test_dataloader = DataLoader(
        test_dataset,
        num_workers=self.hparams.dataloader_num_workers,
        batch_size=self.hparams.batch_size,
        collate_fn=self.pad_batch_collate,
        shuffle=False
    )
    return test_dataloader

  def configure_optimizers(self):
    self.train_dataloader_object = self.train_dataloader()
    optimizer = AdamW(self.parameters(), lr=self.hparams.learning_rate,
                eps=self.hparams.adam_epsilon, weight_decay=self.hparams.weight_decay)

    last_epoch = -1

    # Check load checkpoint model
    if self.checkpoint:
        logger.info("Currently using loaded optimizer")
        optimizer.load_state_dict(self.checkpoint['optimizer_state_dict'])
        last_epoch = self.checkpoint['epoch'] - 1
```

```
        total_steps = len(self.train_dataloader_object) * self.hparams.max_epochs
        scheduler = {
            'scheduler': get_linear_schedule_with_warmup(optimizer,
num_warmup_steps=self.hparams.warmup_steps, num_training_steps=total_steps,
last_epoch=last_epoch),
            'interval':'step'
            }

        return [optimizer], [scheduler]


    def training_step(self, batch, batch_idx):
        # Get batch information
        labels = batch["labels"]
        sources = batch["source"]

        # Delete labels, source, target so now batch contains everything to be inputted into the
model
        del batch["labels"]
        del batch['source']
        del batch['target']


        # If global_step has increased by 1:
        # Begin training the `word_embedding_model` after `num_frozen_steps` steps
        if (self.global_step_tracker + 1) == self.trainer.global_step:
            self.global_step_tracker = self.trainer.global_step

            if self.emd_model_frozen and (self.trainer.global_step >
self.hparams.num_frozen_steps):
                self.emd_model_frozen = False
                self.unfreeze_word_embedding_model()

        # Compute model forward (forward pass to compute output with mask by passing
batch data to the model)
        outputs, mask = self.forward(**batch)

        # Compute loss
        train_loss = self.compute_loss(outputs, labels, mask)
        outputs = torch.sigmoid(outputs)

        # For compute F1 ROUGE score
        system_summaries = []
        ref_summaries = []
        for idx, label in enumerate(labels):
            temp = ""
            for idy, l in enumerate(label):
                if l:
                    temp += sources[idx][idy]
```

```
                temp += '<q>'
            temp = temp[:-3]
            ref_summaries.append(temp)

        source_ids = (
            torch.argsort(outputs, dim=1, descending=True)
        )

        for idx, (source, source_ids, target) in enumerate(
            zip(sources, source_ids, ref_summaries)
        ):
            pos = []
            for sent_idx, i in enumerate(source_ids):
                if i >= len(source):
                    continue

                pos.append(i.item())

                if len(pos) == self.hparams.test_k:
                    break
            pos.sort()
            selected_sentences = "<q>".join([source[i] for i in pos])
            system_summaries.append(selected_sentences)


        if self.hparams.no_use_token_type_ids:
            temp_token_type_ids = "no-token-type-ids"
        else:
            temp_token_type_ids = "use-token-type-ids"
        model_name_or_path = self.hparams.model_name_or_path.replace('/','-')
        self.save_file = "{}_{}_{}_{}_{}_{}_{}_{}".format(
            model_name_or_path, temp_token_type_ids,
            self.hparams.classifier_transformer_num_layers, self.hparams.seed,
self.hparams.learning_rate,
            self.hparams.classifier_dropout, self.hparams.batch_size, self.hparams.max_epochs
            )

        self.temp_train_gold = self.save_file + "/train_gold_" + str(self.epoch) + ".txt"
        self.temp_train_pred = self.save_file + "/train_pred_" + str(self.epoch) + ".txt"

        os.makedirs(self.save_file, exist_ok=True)

        for pred in system_summaries:
            self.all_pred_train += str(pred).strip() + "\n"
        for gold in ref_summaries:
            self.all_gold_train += str(gold).strip() + "\n"

        train_dict = {
            'epoch': self.epoch,
```

```
            'loss': train_loss,
            }
        for name, value in train_dict.items():
            self.log('train/'+name, float(value), prog_bar=True, sync_dist=True)

        return {'loss':train_loss}

    def training_epoch_end(self, outputs):
        avg_train_loss = torch.stack(
            [x['loss'] for x in outputs]
            ).mean()

        with open(self.temp_train_pred, 'w') as save_pred, open(self.temp_train_gold, 'w') as
save_gold:
            save_pred.write(self.all_pred_train)
            save_gold.write(self.all_gold_train)

        self.list_train_loss_epoch.append(avg_train_loss)

        train_dict = {
            'epoch': self.epoch,
            'loss': avg_train_loss
            }
        for name, value in train_dict.items():
            self.log('train/'+name, float(value), prog_bar=True, sync_dist=True)

        self.avg_train_loss = avg_train_loss

        if self.hparams.no_use_token_type_ids:
            temp_token_type_ids = "no-token-type-ids"
        else:
            temp_token_type_ids = "use-token-type-ids"
        model_name_or_path = self.hparams.model_name_or_path.replace('/','-')
        self.save_file = "{}_{}_{}_{}_{}_{}_{}_{}".format(
            model_name_or_path, temp_token_type_ids,
            self.hparams.classifier_transformer_num_layers, self.hparams.seed,
self.hparams.learning_rate,
            self.hparams.classifier_dropout, self.hparams.batch_size, self.epoch
            )

        ckpt_path = f"{self.dir_path}/{self.save_file}.bin"
        optimizer = self.optimizers()
        optimizer = optimizer.optimizer

        train_histories = {
            'loss':self.list_train_loss_epoch,
            }
        val_histories = {
            'loss':self.list_val_loss_epoch,
```

```
        }

    saved_checkpoint = {
        'epoch': self.epoch,
        'hyperparameters': self.hparams,
        'model_state_dict': self.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'train_histories': train_histories,
        'val_histories': val_histories,
        }
    os.makedirs(f"{self.dir_path}",exist_ok=True)
    logger.info("Currently saving model in epoch %s", str(self.epoch))
    torch.save(saved_checkpoint, ckpt_path)

    best_path = "best_bert_model_checkpoints"
    best_name = ""
    best_checkpoint = {}

    # Save best model checkpoint if current validation loss is lower than minimum loss of
previous epoch
    if not self.min_loss:
        self.min_loss = self.avg_val_loss
        best_name = self.save_file
        best_checkpoint = saved_checkpoint
    else:
        if self.avg_val_loss < self.min_loss:
            self.min_loss = self.avg_val_loss
            best_name = self.save_file
            best_checkpoint = saved_checkpoint
            os.makedirs(best_path,exist_ok=True)
            logger.info("Currently saving best bert model in epoch %s", str(self.epoch))
            best_ckpt_path = f"{best_path}/{best_name}.bin"
            torch.save(best_checkpoint, best_ckpt_path)

    self.all_pred_train = ""
    self.all_gold_train = ""
    logger.info("Epoch %2d | Min val loss: %f | Current val loss: %f" %(self.epoch,
self.min_loss, self.avg_val_loss))
    self.epoch += 1


  def validation_step(self, batch, batch_idx):
    # Get batch information
    labels = batch["labels"]
    sources = batch["source"]

    # Delete labels, source, target so now batch contains everything to be inputted into the
model
    del batch["labels"]
```

```
del batch["source"]
del batch["target"]

# Compute model forward
outputs, mask = self.forward(**batch)

# Compute loss
val_loss = self.compute_loss(outputs, labels, mask)
outputs = torch.sigmoid(outputs)

# For compute F1 ROUGE score
system_summaries = []
ref_summaries = []
for idx, label in enumerate(labels):
   temp = ""
   for idy, l in enumerate(label):
      if l:
         temp += sources[idx][idy]
         temp += '<q>'
   temp = temp[:-3]
   ref_summaries.append(temp)
source_ids = (
   torch.argsort(outputs, dim=1, descending=True)
)
for idx, (source, source_ids, target) in enumerate(
   zip(sources, source_ids, ref_summaries)
):
   pos = []
   for sent_idx, i in enumerate(source_ids):
      if i >= len(source):
         continue

      pos.append(i.item())

      if len(pos) == self.hparams.test_k:
         break
   pos.sort()
   selected_sentences = "<q>".join([source[i] for i in pos])
   system_summaries.append(selected_sentences)


if self.hparams.no_use_token_type_ids:
   temp_token_type_ids = "no-token-type-ids"
else:
   temp_token_type_ids = "use-token-type-ids"
model_name_or_path = self.hparams.model_name_or_path.replace('/','-')
self.save_file = "{}_{}_{}_{}_{}_{}_{}_{}".format(
   model_name_or_path, temp_token_type_ids,
```

```
        self.hparams.classifier_transformer_num_layers, self.hparams.seed,
self.hparams.learning_rate,
        self.hparams.classifier_dropout, self.hparams.batch_size, self.hparams.max_epochs
        )

    self.temp_val_gold = self.save_file + "/val_gold_"+ str(self.epoch) +".txt"
    self.temp_val_pred = self.save_file + "/val_pred_"+ str(self.epoch) +".txt"

    os.makedirs(self.save_file, exist_ok=True)

    for pred in system_summaries:
      self.all_pred_val += str(pred).strip() + "\n"
    for gold in ref_summaries:
      self.all_gold_val += str(gold).strip() + "\n"

    val_dict = {
      'epoch': self.epoch,
      'loss': val_loss
      }

    for name, value in val_dict.items():
      self.log('val/'+name, float(value), prog_bar=True, sync_dist=True)

    return {'loss':val_loss}

  def validation_epoch_end(self, outputs):
    # Get the average loss over all evaluation runs
    avg_val_loss = torch.stack(
      [x['loss'] for x in outputs]
    ).mean()

    with open(self.temp_val_pred, 'w') as save_pred, open(self.temp_val_gold, 'w') as
save_gold:
      save_pred.write(self.all_pred_val)
      save_gold.write(self.all_gold_val)

    self.avg_val_loss = avg_val_loss
    self.list_val_loss_epoch.append(avg_val_loss)

    val_dict = {
      'epoch': self.epoch,
      'loss': avg_val_loss,
      }

    for name, value in val_dict.items():
      self.log('val/'+name, float(value), prog_bar=True, sync_dist=True)

    self.all_pred_val = ""
    self.all_gold_val = ""
```

```
  def test_step(self, batch, batch_idx):
    # Get batch information
    labels = batch["labels"]
    sources = batch["source"]
    targets = batch["target"]

    # Delete labels, source, and target so now batch contains everything to be inputted into
the model
    del batch["labels"]
    del batch["source"]
    del batch["target"]

    # Compute model forward
    outputs, _ = self.forward(**batch)
    outputs = torch.sigmoid(outputs)

    sorted_ids = (
      torch.argsort(outputs, dim=1, descending=True).detach().cpu().numpy()
    )

    predictions = []

    if self.ref_summary:
      ref_summaries = targets
    else:
      ref_summaries = []
      for idx, label in enumerate(labels):
        temp = ""
        for idy, l in enumerate(label):
          if l:
            temp += sources[idx][idy]
            temp += '<q>'
        temp = temp[:-3]
        ref_summaries.append(temp)


    # Get ROUGE scores for each (source, target) pair
    for idx, (source, source_ids, target) in enumerate(
      zip(sources, sorted_ids, ref_summaries)
    ):
      current_prediction = []
      pos = []
      for sent_idx, i in enumerate(source_ids):
        if i >= len(source):
          continue

        candidate = source[i].strip()
```

```
            if (not self.hparams.no_test_block_ngrams) and (
                not block_ngrams(candidate, current_prediction, self.hparams.n_gram_blocking)
            ):
                current_prediction.append(candidate)
                pos.append(i.item())

            if len(current_prediction) == self.hparams.test_k:
                break
        pos.sort()
        current_prediction = "<q>".join([source[i] for i in pos])
        predictions.append(current_prediction)

    if self.hparams.no_use_token_type_ids:
        temp_token_type_ids = "no-token-type-ids"
    else:
        temp_token_type_ids = "use-token-type-ids"
    model_name_or_path = self.hparams.model_name_or_path.replace('/','-')
    self.save_file = "{}_{}_{}_{}_{}_{}_{}_{}".format(
        model_name_or_path, temp_token_type_ids,
        self.hparams.classifier_transformer_num_layers, self.hparams.seed,
self.hparams.learning_rate,
        self.hparams.classifier_dropout, self.hparams.batch_size, self.epoch
        )
    self.temp_test_gold = self.save_path_test + "/" + self.save_file + "_test_gold.txt"
    self.temp_test_pred = self.save_path_test + "/" + self.save_file + "_test_pred.txt"

    # Gather all summaries in single text file
    os.makedirs(self.save_path_test, exist_ok=True)

    for pred in predictions:
        self.all_pred_test += str(pred).strip() + "\n"
    for gold in ref_summaries:
        self.all_gold_test += str(gold).strip() + "\n"

    return None

  def test_epoch_end(self, outputs):
    with open(self.temp_test_pred, 'w') as save_pred, open(self.temp_test_gold, 'w') as
save_gold:
        save_pred.write(self.all_pred_test)
        save_gold.write(self.all_gold_test)

    # ROUGE scoring
    raw_rouge, rouge_score = compute_rouge_score(
        self.fold, self.epoch, self.save_file, self.temp_test_dir, self.temp_test_pred,
self.temp_test_gold
        )
    results_dir = "results"
```

```
    os.makedirs(results_dir,exist_ok=True)
    type_ref = "_abstractive" if self.ref_summary else "_extractive"
    file = results_dir + "/" + self.save_file + type_ref + ".txt"
    with open(file,'w') as f:
      f.write("Precision\n")
      f.write(
        "ROUGE-1 : {} \nROUGE-2 : {} \nROUGE-L : {}\n\n".format(
          rouge_score['precision-rouge-1'],
          rouge_score['precision-rouge-2'],
          rouge_score['precision-rouge-l']
          )
        )
      f.write("Recall\n")
      f.write(
        "ROUGE-1 : {} \nROUGE-2 : {} \nROUGE-L : {}\n\n".format(
          rouge_score['recall-rouge-1'],
          rouge_score['recall-rouge-2'],
          rouge_score['recall-rouge-l']
          )
        )
      f.write("F1-Score\n")
      f.write(
        "ROUGE-1 : {} \nROUGE-2 : {} \nROUGE-L : {}\n\n".format(
          rouge_score['f1-rouge-1'],
          rouge_score['f1-rouge-2'],
          rouge_score['f1-rouge-l']
          )
        )


    test_dict = {
      **rouge_score,
      **raw_rouge
      }

    # Generate logs
    for name, value in test_dict.items():
      self.log('test/'+name, float(value), prog_bar=True, sync_dist=True)


    self.test_preds = []
    self.test_labels = []
    self.all_pred_test = ""
    self.all_gold_test = ""
```

4.7 Transformer encoder

Pada bagian ini akan dilakukan klasifikasi dengan menggunakan encoder dari transformer. Tujuan dari penggunaan *transformer encoder* ini pada layer klasifikasi adalah untuk mempelajari relasi antar kalimat dari output yang dihasilkan oleh model BERT. *Position encoding* yang digunakan pada model BERT akan dipelajari secara otomatis saat melakukan training sedangkan *position encoding* yang dipakai sebagai *classifier* pada tambahan *transformer encoder layer* akan menggunakan *sinusoidal position encoding*. Penggunaan dari *position encoding* bertujuan untuk memberikan informasi mengenai posisi tiap *token* dalam kalimat. Mengenai proses dalam melakukan klasifikasi dapat dilihat pada Segmen Program 4.10.

Segmen Program 4.10 Implementasi klasifikasi pada transformer encoder

```
import logging
import torch
from torch import nn
import math

logger = logging.getLogger(__name__)

# Sinusoidal position encoding
class PositionalEncoding(nn.Module):

    def __init__(self, d_model=768, dropout=0.1, max_len=512):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) /
d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0).transpose(0, 1)
        self.register_buffer('pe', pe)

    def forward(self, x):
        x = x + self.pe[:x.size(0), :]
        return self.dropout(x)

# Transformer encoder with sinusoidal position encoding
class TransformerEncoderClassifier(nn.Module):
    def __init__(
        self,
        d_model=768,
        nhead=8,
        dim_feedforward=2048,
```

```
        dropout=0.1,
        num_layers=2
    ):
        super(TransformerEncoderClassifier, self).__init__()
        self.nhead = nhead
        self.pos_encoder = PositionalEncoding(d_model, dropout)
        encoder_layer = nn.TransformerEncoderLayer(
            d_model, nhead, dim_feedforward=dim_feedforward, dropout=dropout
        )
        layer_norm = nn.LayerNorm(d_model)
        self.encoder = nn.TransformerEncoder(encoder_layer, num_layers, norm=layer_norm)
        wo = nn.Linear(d_model, 1, bias=True)
        self.reduction = wo

    def forward(self, x, mask):

        # apply sinusoidal position encoding to input sequence of token
        x = self.pos_encoder(x)

        # add dimension in the middle
        attn_mask = mask.unsqueeze(1)

        # expand the middle dimension to the same size as the last dimension (the number of
sentences/source length)
        attn_mask = attn_mask.expand(-1, attn_mask.size(2), -1)

        # repeat the mask for each attention head
        attn_mask = attn_mask.repeat(self.nhead, 1, 1)

        # attn_mask is shape (batch size*num_heads, target sequence length, source sequence
length)
        # set all the 0's (False) to negative infinity and the 1's (True) to 0.0 because the
attn_mask is additive
        attn_mask = (
            attn_mask.float()
            .masked_fill(attn_mask == 0, float("-inf"))
            .masked_fill(attn_mask == 1, float(0.0))
        )

        x = x.transpose(0, 1)
        # x is shape (source sequence length, batch size, feature number)

        x = self.encoder(x, mask=attn_mask)
        # x is still shape (source sequence length, batch size, feature number)

        x = x.transpose(0, 1).squeeze()
        # x is shape (batch size, source sequence length, feature number)

        x = self.reduction(x)
```

```
    # x is shape (batch size, source sequence length, 1)
    # mask is shape (batch size, source sequence length)
    sent_scores = x.squeeze(-1) * mask.float()

    # to preserve loss calculation on padding token
    sent_scores[sent_scores == 0] = -9e3

    return sent_scores
```

4.8 Evaluasi dengan ROUGE

Dalam mengukur model yang sudah dibuat, akan dilakukan evaluasi terhadap dataset *test* dengan menggunakan ROUGE score. Setiap pasangan kandidat kalimat yang akan menjadi ringkasan sistem dengan referensi ringkasan akan disimpan ke dalam file teks yang kemudian akan dilakukan evaluasi dengan ROUGE. Implementasi untuk evaluasi ini dapat dilihat pada Segmen Program 4.11.

Segmen Program 4.11 Evaluasi ROUGE

```
def compute_rouge_score(fold, epoch, save_file, temp_dir, cand, ref):
   candidates = [line.strip() for line in open(cand, encoding="utf-8")]
   references = [line.strip() for line in open(ref, encoding="utf-8")]
   assert len(candidates) == len(references)

   cnt = len(candidates)
   current_time = time.strftime("%Y-%m-%d-%H-%M-%S", time.localtime())
   os.makedirs(temp_dir, exist_ok=True)
   tmp_dir = os.path.join(temp_dir, "rouge-score-{}".format(current_time))
   os.makedirs(tmp_dir, exist_ok=True)
   os.makedirs(f"{tmp_dir}/candidate_{fold}", exist_ok=True)
   os.makedirs(f"{tmp_dir}/reference_{fold}", exist_ok=True)

   try:

      for i in range(cnt):
         if len(references[i]) < 1:
            continue
         with open(
            f"{tmp_dir}/candidate_{fold}/cand.{i}.txt", "w", encoding="utf-8"
         ) as f:
            f.write(candidates[i].replace("<q>", "\n"))
         with open(
            f"{tmp_dir}/reference_{fold}/ref.{i}.txt", "w", encoding="utf-8"
         ) as f:
            f.write(references[i].replace("<q>", "\n"))
```

```
    # pyrouge
    r = pyrouge.Rouge155()
    r.model_dir = f"{tmp_dir}/reference_{fold}/"
    r.system_dir = f"{tmp_dir}/candidate_{fold}/"
    r.model_filename_pattern = "ref.#ID#.txt"
    r.system_filename_pattern = "cand.(\d+).txt"
    command = '-e /content/rouge/tools/ROUGE-1.5.5/data -a -b 75 -c 95 -m -n 2'
    rouge_results = r.convert_and_evaluate(rouge_args=command)
    results_dict_rouge = r.output_to_dict(rouge_results)

    rouge_1_recall = "{:.2f}".format(float(results_dict_rouge['rouge_1_recall'] * 100))
    rouge_2_recall = "{:.2f}".format(float(results_dict_rouge['rouge_2_recall'] * 100))
    rouge_l_recall = "{:.2f}".format(float(results_dict_rouge['rouge_l_recall'] * 100))

    rouge_1_precision = "{:.2f}".format(float(results_dict_rouge['rouge_1_precision'] *
100))
    rouge_2_precision = "{:.2f}".format(float(results_dict_rouge['rouge_2_precision'] *
100))
    rouge_l_precision = "{:.2f}".format(float(results_dict_rouge['rouge_l_precision'] * 100))

    rouge_1_f_score = "{:.2f}".format(float(results_dict_rouge['rouge_1_f_score'] * 100))
    rouge_2_f_score = "{:.2f}".format(float(results_dict_rouge['rouge_2_f_score'] * 100))
    rouge_l_f_score = "{:.2f}".format(float(results_dict_rouge['rouge_l_f_score'] * 100))
    results = {}

    # Recall
    results['recall-rouge-1'] = rouge_1_recall
    results['recall-rouge-2'] = rouge_2_recall
    results['recall-rouge-l'] = rouge_l_recall

    # Precision
    results['precision-rouge-1'] = rouge_1_precision
    results['precision-rouge-2'] = rouge_2_precision
    results['precision-rouge-l'] = rouge_l_precision

    # F1-Score
    results['f1-rouge-1'] = rouge_1_f_score
    results['f1-rouge-2'] = rouge_2_f_score
    results['f1-rouge-l'] = rouge_l_f_score



    # save rouge score at specified file
    corpus_type = str(tmp_dir.split('_')[0])
    evaluation_dir = "final_evaluation_" + corpus_type
    os.makedirs(evaluation_dir, exist_ok=True)
    with open(os.path.join(evaluation_dir, f"{save_file}.txt"), 'w', encoding='utf-8') as f:
      f.write("ROUGE Score\n")
      f.write(rouge_results + '\n\n')
  finally:
```

```
    if os.path.isdir(tmp_dir):
        shutil.rmtree(tmp_dir)

  return results_dict_rouge, results
```

4.9 Prediksi ringkasan

Dalam melakukan prediksi suatu ringkasan, akan dilakukan *load* dari *checkpoint model*. Input teks artikel berita yang dimasukkan ke dalam model akan dilakukan pemotongan apabila melebihi *maximum length* dari model BERT. Mengenai implementasi dari prediksi ringkasan dapat dilihat pada Segmen Program 4.12.

Segmen Program 4.12 Prediksi ringkasan berita

```
import os
import sys
import requests
import torch
from argparse import ArgumentParser
from newspaper import Article

sys.path.insert(0, os.path.abspath("./src"))
from extractive import ExtractiveSummarizer

parser = ArgumentParser(description='Indonesian News Summarization')
parser.add_argument('--model', type=str, default='./models/indolem-indobert-base-
uncased_basic_use-token-type-ids_sent-rep-tokens_transformer_position_2_2_3e-
05_0.3_8_1.bin',help='trained model')
parser.add_argument('--source',type=str, default='./kumpulan_berita/berita_4_cnn.txt',
help='source of news')
parser.add_argument('--percentages',type=float, default='20.0',help='percentages of
summary sentences')
parser.add_argument('--save_dir', type=str, default='./kumpulan_berita', help='directory of
saved news article')

args = parser.parse_args()

checkpoint = torch.load(args.model, map_location=torch.device('cpu'))
state_dict = checkpoint['model_state_dict']
model = ExtractiveSummarizer(checkpoint['hyperparameters'])
model.load_state_dict(checkpoint['model_state_dict'])


# Check source
try:
  #Get news article
  assert(requests.get(args.source))
```

```
    news_article = Article(args.source)
    news_article.download()
    news_article.parse()
    temp = news_article.text.split('\n\n')
    contents = [sent+"\n" for sent in temp]
    file_name = news_article.title.replace(' ','-') + '.txt'
    source = 'url/' + file_name
    with open(os.path.join(args.save_dir, file_name), 'w', encoding='utf-8') as f:
      f.write('\n'.join(contents))
except:
    # Open text file
    with open(args.source) as f:
      contents = f.readlines()
    source = 'file/' + os.path.basename(args.source)

# Convert percentages to number of sentences (based on number of sentences in news text)
num_sentences = int((args.percentages*0.01) * len(contents))

# Check if sentences less than 1, then set number of sentence to min. 1
if num_sentences < 1:
        num_sentences = 1

# Predict
output = model.predict(contents, source, num_sentences)
print(output)
```