Anggota Kelompok :
- Chandra Viriya Hartantio C14210075
- Nicholas Lawrentius Setyadharma C14210121
- Kristofer Steven C14210139
- Samuel Huang C14210181

LAPORAN HASIL EKSPERIMEN MODEL

DATASET

Dataset yang dipakai pada proyek ini adalah squad v2, yang berisi konteks, pertanyaan dan jawaban. Dataset memiliki jumlah sebanyak 130.319 pertanyaan pada training, dan 11.873 pada testing.

```
[ ]  dataset = load_dataset("rajpurkar/squad_v2")
```

```
[ ]  dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 130319
    })
    validation: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 11873
    })
})
```

```
[ ]  len_train = len(dataset["train"])
     len_train
```

```
130319
```

Karena dataset sangat banyak dan besar, kami hanya memakai 10% dari dataset training dan membaginya sebanyak 80:20 untuk training dan testing. Hal ini dilakukan agar proses training dapat dilakukan. Karena jika tidak, training akan memakan waktu yang sangat lama dan memakan resource yang sangat besar.

```
[ ] print("id: ", dataset["train"][0]["id"])
    print("title: ", dataset["train"][0]["title"])
    print("Context: ", dataset["train"][0]["context"])
    print("Question: ", dataset["train"][0]["question"])
    print("Answer: ", dataset["train"][0]["answers"])
```

```
id:  56be85543aeaaa14008c9063
title:  Beyoncé
Context:  Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ bee-YON-say) (born September 4, 1981) is
Question:  When did Beyonce start becoming popular?
Answer:  {'text': ['in the late 1990s'], 'answer_start': [269]}
```

```
[ ] dataset_10_percent = dataset["train"].train_test_split(test_size=0.9)["train"]

    # Further split the 10% dataset into train and eval sets
    dataset_split = dataset_10_percent.train_test_split(test_size=0.2)
```

Contoh dataset:

```
id:  56be85543aeaaa14008c9063

title:  Beyoncé

Context:  Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ bee-YON-say) (born September 4, 1981) is an American singer, songwriter, record producer and actress. Born and raised in Houston, Texas, she performed in various singing and dancing competitions as a child, and rose to fame in the late 1990s as lead singer of R&B girl-group Destiny's Child. Managed by her father, Mathew Knowles, the group became one of the world's best-selling girl groups of all time. Their hiatus saw the release of Beyoncé's debut album, Dangerously in Love (2003), which established her as a solo artist worldwide, earned five Grammy Awards and featured the Billboard Hot 100 number-one singles "Crazy in Love" and "Baby Boy".

Question:  When did Beyonce start becoming popular?

Answer:  {'text': ['in the late 1990s'], 'answer_start': [269]}
```

DistilBERT

Model distilBERT yang diambil merupakan model distilbert-base-uncased yang di publish melalui website huggingface.

Untuk library-library yang dibutuhkan adalah transformers, datasets, accelerate, dan torch

```
!pip install transformers
!pip install datasets
!pip install accelerate
!pip install torch
```

Untuk bagian pertama, dataset SQUADv2 yang digunakan sebagai dataset untuk percobaan di import ke dalam google collab untuk proses training pada model distilBERT

```
# Load the SQuAD dataset
dataset = load_dataset("squad_v2")

# Choose only 10% of the dataset
dataset_10_percent = dataset["train"].train_test_split(test_size=0.9)["train"]

# Further split the 10% dataset into train and eval sets
dataset_split = dataset_10_percent.train_test_split(test_size=0.2)
train_dataset = dataset_split["train"]
eval_dataset = dataset_split["test"]

# Load tokenizer
model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

Dataset tersebut hanya diambil 10% nya untuk mengurangi train-time dari model sehingga terhindar dari kemungkinan error karena kehabisan runtime GPU pada google colab.

```
def add_token_positions(examples):
    tokenized_examples = tokenizer(
        examples['question'], examples['context'],
        truncation=True, padding='max_length', max_length=512,
        return_offsets_mapping=True
    )

    start_positions = []
    end_positions = []

    for i, (input_ids, answer, offsets) in enumerate(zip(tokenized_examples['input_ids'], examples['answers'], tokenized_examples['offset_mapping'])):
        cls_index = input_ids.index(tokenizer.cls_token_id)

        # If no answers are given, set start and end positions to CLS token
        if len(answer['text']) == 0:
            start_positions.append(cls_index)
            end_positions.append(cls_index)
        else:
            # Start and end character positions of the answer in the context
            start_char = answer['answer_start'][0]
            end_char = start_char + len(answer['text'][0])

            # Token start and end positions within the context
            token_start_index = 0
            token_end_index = 0
            for offset in offsets:
                if offset[0] <= start_char < offset[1]:
                    token_start_index = offsets.index(offset)
                if offset[0] < end_char <= offset[1]:
                    token_end_index = offsets.index(offset)
                    break

            start_positions.append(token_start_index)
            end_positions.append(token_end_index)

    tokenized_examples['start_positions'] = start_positions
    tokenized_examples['end_positions'] = end_positions
    return tokenized_examples
```

Sebuah function yang dinamakan add_token_positions juga dibuat untuk menambahkah informasi posisi mulai dan akhir pada isi dataset SQUADv2

```
# Apply the preprocessing function to the train and eval datasets
tokenized_train_dataset = train_dataset.map(add_token_positions, batched=True, remove_columns=['id', 'title', 'context', 'question', 'answers'])
tokenized_eval_dataset = eval_dataset.map(add_token_positions, batched=True, remove_columns=['id', 'title', 'context', 'question', 'answers'])

# Set format for PyTorch
tokenized_train_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'start_positions', 'end_positions'])
tokenized_eval_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'start_positions', 'end_positions'])
```

Function tersebut kemudian diaplikasikan ke dataset dan diformat menjadi tipe torch untuk proses training

```
# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
)

# Load model
model = AutoModelForQuestionAnswering.from_pretrained(model_name)

# Define Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train_dataset,
    eval_dataset=tokenized_eval_dataset,
    tokenizer=tokenizer,
)

# Fine-tune the model
trainer.train()
```

Model kemudian di train sebanyak 3 epoch dan learning rate sebesar 0.00002 dengan report hasil training seperti berikut

Epoch	Training Loss	Validation Loss
1	2.421600	2.234608
2	1.775500	1.995523
3	1.593300	1.958644

Hasil akhirnya adalah sebuah model yang mampu memberikan jawaban pada pertanyaan berdasarkan context yang diberikan. Berikut adalah beberapa contoh question dan context serta output dari model.

```
# Example context passage and question for inference
context = "France is a European country, it's capital is Paris"
question = "What is the capital of France?"

# Tokenize inputs
inputs = tokenizer.encode_plus(question, context, return_tensors="pt", max_length=512, truncation=True)
inputs = {k: v.to(trainer.args.device) for k, v in inputs.items()}

# Perform inference
with torch.no_grad():
    outputs = trainer.model(**inputs)

# Get the most probable answer
start_index = torch.argmax(outputs.start_logits)
end_index = torch.argmax(outputs.end_logits)
answer = tokenizer.convert_tokens_to_string(tokenizer.convert_ids_to_tokens(inputs["input_ids"][0][start_index:end_index+1]))

print("Question:", question)
print("Answer:", answer)
```

```
Question: What is the capital of France?
Answer: paris
```

```
# Example context passage and question for inference
context = "The Sistine Chapel is a chapel in the Apostolic Palace, the official residence of the pope, in Vatican City. The ceiling of the Sistine Chapel was painted by Michelangelo."
question = "Who painted the ceiling of the Sistine Chapel?"

# Tokenize inputs
inputs = tokenizer.encode_plus(question, context, return_tensors="pt", max_length=512, truncation=True)

# Perform inference
with torch.no_grad():
    outputs = model(**inputs)

# Get the most probable answer
start_index = torch.argmax(outputs.start_logits)
end_index = torch.argmax(outputs.end_logits)
answer = tokenizer.convert_tokens_to_string(tokenizer.convert_ids_to_tokens(inputs["input_ids"][0][start_index:end_index+1]))

print("Question:", question)
print("Answer:", answer)
```

```
Question: Who painted the ceiling of the Sistine Chapel?
Answer: michelangelo
```

```
question = "where is max's home country"
context = "Max verstappen, a 20 year old formula 1 champion, is highly praised by many fans in his home country 'belgium'"

# Tokenize inputs
inputs = tokenizer.encode_plus(question, context, return_tensors="pt", max_length=512, truncation=True)

# Perform inference
with torch.no_grad():
    outputs = model(**inputs)

# Get the most probable answer
start_index = torch.argmax(outputs.start_logits)
end_index = torch.argmax(outputs.end_logits)
answer = tokenizer.convert_tokens_to_string(tokenizer.convert_ids_to_tokens(inputs["input_ids"][0][start_index:end_index+1]))

print("Question:", question)
print("Answer:", answer)
```

```
Question: where is max's home country
Answer: belgium
```

Tentunya model juga masih memiliki kesalahan dalam men generate jawaban dimana model gagal dalam memahami context sehingga tidak bisa atau tidak menemukan jawaban.

```
question = "why is max praised"
context = "Max verstappen, a 20 year old formula 1 champion, is highly praised by many fans in his home country 'belgium'. test gap .He is praised because of his amazin

# Tokenize inputs
inputs = tokenizer.encode_plus(question, context, return_tensors="pt", max_length=512, truncation=True)

# Perform inference
with torch.no_grad():
    outputs = model(**inputs)

# Get the most probable answer
start_index = torch.argmax(outputs.start_logits)
end_index = torch.argmax(outputs.end_logits)
answer = tokenizer.convert_tokens_to_string(tokenizer.convert_ids_to_tokens(inputs["input_ids"][0][start_index:end_index+1]))

print("Question:", question)
print("Answer:", answer)
```

```
Question: why is max praised
Answer:
```

```
#tidak mampu menjawab pertanyaan
question = "how old is max"
context = "Max verstappen, a 20 year old formula 1 champion, is highly praised by many fans in his home country 'belgium'"

# Tokenize inputs
inputs = tokenizer.encode_plus(question, context, return_tensors="pt", max_length=512, truncation=True)

# Perform inference
with torch.no_grad():
    outputs = model(**inputs)

# Get the most probable answer
start_index = torch.argmax(outputs.start_logits)
end_index = torch.argmax(outputs.end_logits)
answer = tokenizer.convert_tokens_to_string(tokenizer.convert_ids_to_tokens(inputs["input_ids"][0][start_index:end_index+1]))

print("Question:", question)
print("Answer:", answer)
```

```
Question: how old is max
Answer: [CLS]
```

```
# Example context passage and question for inference
context = "The Apollo program, also known as Project Apollo, was the third United States human spaceflight program carriedout b
question = "Who were the first humans to land on the Moon?"

# Tokenize inputs
inputs = tokenizer.encode_plus(question, context, return_tensors="pt", max_length=512, truncation=True)

# Perform inference
with torch.no_grad():
    outputs = model(**inputs)

# Get the most probable answer
start_index = torch.argmax(outputs.start_logits)
end_index = torch.argmax(outputs.end_logits)
answer = tokenizer.convert_tokens_to_string(tokenizer.convert_ids_to_tokens(inputs["input_ids"][0][start_index:end_index+1]))

print("Question:", question)
print("Answer:", answer)
```

```
Question: Who were the first humans to land on the Moon?
Answer: [CLS]
```

Kemungkinan kesalahan adalah karena model distilBERT kurang mampu memproses dataset SQUADv2 yang beberapa instancenya merupakan pasangan pertanyaan yang dijawab dengan pertanyaan juga sehingga mungkin mengakibatkan error pada pendeteksian context.

Karena itu, model distilBERT juga dicoba untuk diproses dengan dataset SQUADv1 yang mirip dengan SQUADv2 dengan data yang 100% berisikan pasangan pertanyaan dan jawaban.

Hasil dari testing model tersebut dapat menjawab pertanyaan yang sebelumnya tidak bisa dijawab oleh model distilBERT SQUADv2

```
# Example context passage and question for inference
context = "The Apollo program, also known as Project Apollo, was the third United States human spaceflight program carried out
question = "Who were the first humans to land on the Moon?"

# Tokenize inputs
inputs = tokenizer.encode_plus(question, context, return_tensors="pt", max_length=512, truncation=True)

# Perform inference
with torch.no_grad():
    outputs = model(**inputs)

# Get the most probable answer
start_index = torch.argmax(outputs.start_logits)
end_index = torch.argmax(outputs.end_logits)
answer = tokenizer.convert_tokens_to_string(tokenizer.convert_ids_to_tokens(inputs["input_ids"][0][start_index:end_index+1]))

print("Question:", question)
print("Answer:", answer)
```

```
Question: Who were the first humans to land on the Moon?
Answer: neil armstrong and buzz aldrin
```

```
#tidak mampu menjawab pertanyaan mengenai subjek dengan pertanyaan non spesifik

question = "who said that"
context = "an old man named george luther spencer the fiftienth once said that the keys to everlasting wealth are health, friend

# Tokenize inputs
inputs = tokenizer.encode_plus(question, context, return_tensors="pt", max_length=512, truncation=True)

# Perform inference
with torch.no_grad():
    outputs = model(**inputs)

# Get the most probable answer
start_index = torch.argmax(outputs.start_logits)
end_index = torch.argmax(outputs.end_logits)
answer = tokenizer.convert_tokens_to_string(tokenizer.convert_ids_to_tokens(inputs["input_ids"][0][start_index:end_index+1]))

print("Question:", question)
print("Answer:", answer)
```

```
Question: who said that
Answer: george luther spencer
```

Kesimpulan dari eksperimen ini adalah model distilBERT masih kalah pada tingkat efisiensi training dan ukuran apabila dibandingkan dengan model ALBERT yang dapat menyelesaikan proses training 2 kali lipat lebih cepat dan menggunakan lebih sedikit storage untuk penyimpanan trained modelnya.

ALBERT

Model ALBERT yang kami pakai adalah albert-base-v2 dari huggingface.
Untuk model, kami memakai AlbertForQuestionAnswering dan untuk tokenizer kami memakai AlbertTokenizer yang sudah disediakan dari library huggingface.

```
tokenizer = AutoTokenizer.from_pretrained("albert-base-v2")
model = AlbertForQuestionAnswering.from_pretrained("albert/albert-base-v2")
```

Karena albert awalnya digunakan untuk masalah masking, sehingga jika dipakai pada model question answering terdapat beberapa weight yang belum terinisialisasi.

Alasan kami memakai albert adalah model ini adalah optimasi dari model BERT sehingga dapat melakukan training lebih cepat karena memiliki parameter yang lebih sedikit.

Untuk preprocess kami membuat preprocess yang kurang lebih sama dengan distilbert, untuk membandingkan hasil dari kedua model ini

```
max_query_length = 64
max_seq_length = 386
doc_stride = 128


def preprocess_training_examples(examples):
    questions = [q.strip() for q in examples["question"]]
    inputs = tokenizer(
        questions,
        examples["context"],
        max_length=max_seq_length,
        truncation="only_second",
        stride=doc_stride,
        return_overflowing_tokens=True,
        return_offsets_mapping=True,
        padding="max_length",
    )

    offset_mapping = inputs.pop("offset_mapping")
    sample_map = inputs.pop("overflow_to_sample_mapping")
    answers = examples["answers"]
    start_positions = []
    end_positions = []
    cls_token_id = tokenizer.cls_token_id

    for i, offset in enumerate(offset_mapping):
        sample_idx = sample_map[i]
        answer = answers[sample_idx]
        if len(answer["answer_start"]) == 0:

            start_positions.append(cls_token_id)
            end_positions.append(cls_token_id)
        else:
            start_char = answer["answer_start"][0]
            end_char = answer["answer_start"][0] + len(answer["text"][0])
            sequence_ids = inputs.sequence_ids(i)
```

```
        sequence_ids = inputs.sequence_ids(i)

        idx = 0
        while sequence_ids[idx] != 1:
            idx += 1
        context_start = idx
        while sequence_ids[idx] == 1:
            idx += 1
        context_end = idx - 1


        if offset[context_start][0] > start_char or offset[context_end][1] < end_char:
            start_positions.append(0)
            end_positions.append(0)
        else:

            idx = context_start
            while idx <= context_end and offset[idx][0] <= start_char:
                idx += 1
            start_positions.append(idx - 1)

            idx = context_end
            while idx >= context_start and offset[idx][1] >= end_char:
                idx -= 1
            end_positions.append(idx + 1)

    inputs["start_positions"] = start_positions
    inputs["end_positions"] = end_positions
    return inputs
```

Kami memakai max seq length sebesar 386 token untuk membatasi panjang kata. Hal ini awalnya bertujuan untuk membandingkan model ALBERT dengan model RoBERTa yang sudah di pretrain dengan dataset yang sama yaitu squad v2 pada library huggingface. Alasan kami memilih RoBERTa sebagai pembanding adalah karena pada jumlah orang yang memakai RoBERTa yang sudah di pretrain ini sangat banyak, sehingga kami beranggapan bahwa settingan dan model tersebut sudah berhasil sebagai model untuk question answering

deepset/roberta-base-squad2
Question Answering • Updated Mar 18 • 750k • ♡ 678

deepset/bert-large-uncased-whole-word-masking-squad2
Question Answering • Updated Sep 26, 2023 • 356k • ♡ 27

distilbert/distilbert-base-cased-distilled-squad
Question Answering • Updated May 6 • 212k • ♡ 183

google-bert/bert-large-uncased-whole-word-masking-f…
Question Answering • Updated Feb 19 • 193k • ♡ 149

monologg/koelectra-small-v2-distilled-korquad-384
Question Answering • Updated Jun 12, 2023 • 162k • ♡ 3

valhalla/bart-large-finetuned-squadv1
Question Answering • Updated Jun 14, 2021 • 91.7k • ♡ 7

distilbert/distilbert-base-uncased-distilled-squad
Question Answering • Updated May 6 • 77k • ♡ 88

ahotrod/electra_large_discriminator_squad2_512
Question Answering • Updated Dec 12, 2020 • 72.4k • ♡ 6

deepset/tinyroberta-squad2
Question Answering • Updated Sep 27, 2023 • 58.1k • ♡ 87

csarron/mobilebert-uncased-squad-v2
Question Answering • Updated Jul 18, 2023 • 46.3k • ♡ 5

sshleifer/tiny-distilbert-base-cased-distilled-squad
Question Answering • Updated May 14, 2020 • 30.5k

philschmid/distilbert-onnx
Question Answering • Updated Feb 16, 2022 • 27.6k • ♡ 2

Untuk TrainingArguments dan Trainer, kami membuatnya sama dengan distilbert, sebagai perbandingan.

```
[ ] training_args = TrainingArguments(
        output_dir='./results',
        evaluation_strategy="epoch",
        learning_rate=2e-5,
        per_device_train_batch_size=16,
        per_device_eval_batch_size=16,
        num_train_epochs=3,
        weight_decay=0.01,
        logging_dir='./logs',
        logging_steps=10,
    )
```

```
/usr/local/lib/python3.10/dist-packages/tr
  warnings.warn(
```

```
[ ] trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=validation_dataset,
        tokenizer=tokenizer,
        return_loss=True
    )
```

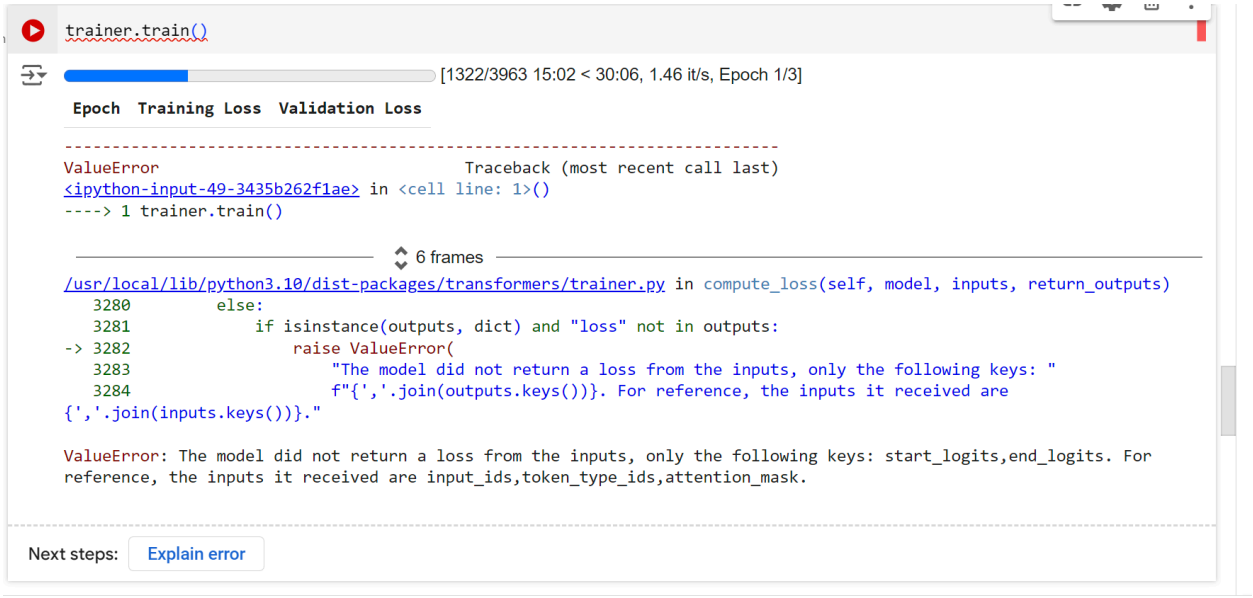
```
trainer.train()
```

[1980/1980 24:43, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	1.280900	No log
2	1.036900	No log
3	0.363900	No log

```
TrainOutput(global_step=1980, training_loss=1.0759086218747225, metri
526844664929952.0, 'train_loss': 1.0759086218747225, 'epoch': 3.0})
```

Dan hasil training pada model ini menunjukkan bahwa model albert memiliki performa yang lebih baik dibandingkan dengan model distilbert, dapat dilihat dari training loss pada epoch 1, training loss pada ALBERT dimulai dari 1,28 sedangkan distilbert dimulai dari 2,4. Untuk validation loss, setelah kami merubah beberapa settingan agar model ini dapat return validation loss, kami mendapati bahwa model ini tidak me-return validation loss, hanya start logits dan end logits, ditunjukkan pada error dibawah ini.



Padahal, kami sudah memberi parameter untuk validation dataset.
Untuk hasilnya, model ini sudah dapat menjawab pertanyaan sederhana seperti dibawah ini dengan benar, dibandingkan dengan model distilbert yang hanya menjawab [CLS] (token cls)

Context = "The Apollo program, also known as Project Apollo, was the third United States human spaceflight program carried out by the National Aeronautics and Space Administration (NASA). The program succeeded in landing the first humans on the Moon from 1969 to 1972. Apollo set major milestones in human spaceflight and space exploration, leading the United States to become the first and only country to have landed humans on the Moon. The first landing happened on July 20, 1969, when Neil Armstrong and Buzz Aldrin, both American astronauts, landed the Apollo Lunar Module Eagle on the Moon's surface."
Question = "Who were the first humans to land on the Moon?"

```python
context = "The Apollo program, also known as Project Apollo, was the third United States human s
question = "Who were the first humans to land on the Moon?"

inference(question, context)
```

```
{'input_ids': tensor([[    2,    72,    46,    14,    64,  2840,    20,   476,    27,    14,
          2121,    60,     3,    14,  8532,   625,    15,    67,   167,    28,
           669,  8532,    15,    23,    14,   422,   181,   202,   585,   726,
         14750,   625,  1521,    70,    34,    14,   152, 24874,    18,    17,
           726,  1603,    13,     5,  6169,    58,     6,     9,    14,   625,
          2914,    19,  3090,    14,    64,  2840,    27,    14,  2121,    37,
          2533,    20,  2249,     9,  8532,   309,   394, 19104,    18,    19,
           585,   726, 14750,    17,   726,  8284,    15,  1005,    14,   181,
           202,    20,   533,    14,    64,    17,   104,   475,    20,    57,
          4388,  2840,    27,    14,  2121,     9,    14,    64,  3090,  1190,
            27,   313,   434,    15,  2533,    15,    76,  5870,  9158,    17,
          9122,    13,  8100,  2445,    15,   156,   189, 15226,    18,    15,
          4388,    14,  8532, 12710, 12613,  5807,    27,    14,  2121,    22,
            18,  1490,     9,     3]]), 'token_type_ids': tensor([[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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]), 'attention_mask': tensor([[1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
tensor(107, device='cuda:0')
tensor(5870, device='cuda:0')
neil
Question: Who were the first humans to land on the Moon?
Answer: neil armstrong and buzz aldrin
```

Model ini juga sudah dapat menjawab pertanyaan sederhana seperti berikut

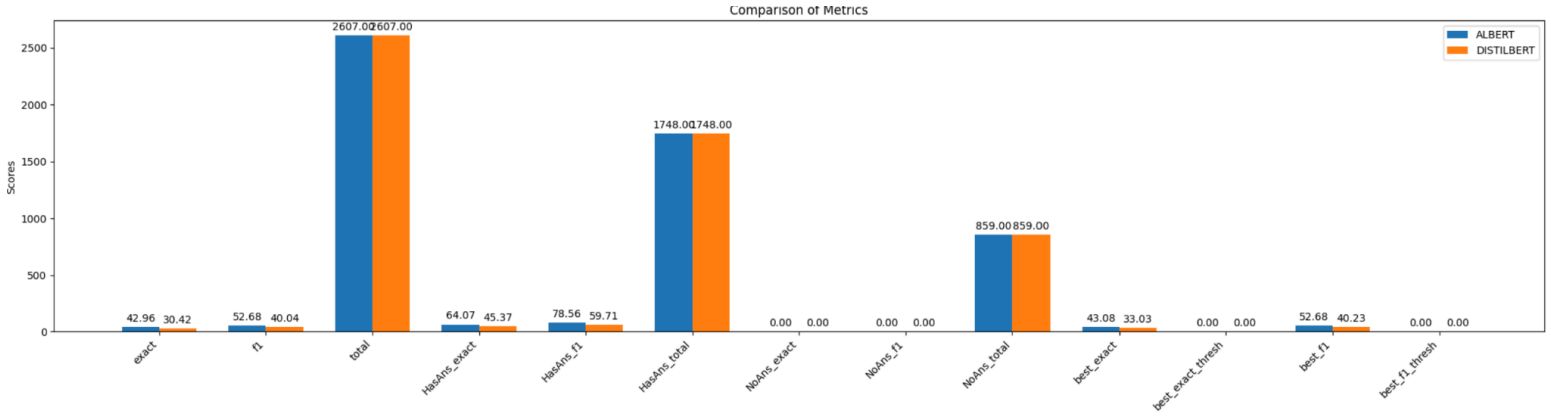
```python
inference("what is my name?", "my name is chandra")
```

```
{'input_ids': tensor([[    2,    98,    25,    51,   204,    60,     3,    51,
         12598,     3]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 1, 1, 
tensor(10, device='cuda:0')
tensor(12598, device='cuda:0')
chandra
Question: what is my name?
Answer: chandra
```

Karena dataset squad v2 ini memiliki data yang tidak memiliki jawaban, maka kami mencoba untuk mengubah bagian preprocessing, ketika data tersebut tidak memiliki jawaban, kami menambah start index jawaban pada cls token dan end index jawaban ada pada sep token.

```
offset_mapping = inputs.pop("offset_mapping")
sample_map = inputs.pop("overflow_to_sample_mapping")
answers = examples["answers"]
start_positions = []
end_positions = []
cls_token_id = tokenizer.cls_token_id
sep_token_id = tokenizer.sep_token_id

for i, offset in enumerate(offset_mapping):
    sample_idx = sample_map[i]
    answer = answers[sample_idx]
    if len(answer["answer_start"]) == 0:

        start_positions.append(cls_token_id)
        end_positions.append(sep_token_id)
    else:
```

Hasilnya adalah model tetap menjawab pada pertanyaan yang tidak memiliki jawaban, tetapi berhenti jika menemui token [sep]. Percobaan ini tidak menghasilkan perubahan yang signifikan.



Kami juga melakukan perbandingan metrics untuk ALBERT dan DistilBERT. Dengan dataset yang sama, ALBERT dapat mencapai exact match sebanyak 42% sedangkan distilbert sebanyak 30%, hasil F1 score Albert mencapai 52.68 dibandingkan dengan distilbert dengan nilai 40. Dan beberapa metrics lain perbedaan yang ada adalah sekitar 10 poin.

```
# 1
question = "When did Beyonce start becoming popular?"
context = "Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ bee-YON-say) (born September 4, 1981) is an American si

get_answer(context, question)
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork() is incomp
  self.pid = os.fork()
albert:  {'score': 0.4593450129032135, 'start': 276, 'end': 286, 'answer': 'late 1990s'}
albert answer:  late 1990s

 distilbert:  {'score': 0.07945173978805542, 'start': 525, 'end': 532, 'answer': '(2003),'}
distilbert answer:  (2003),
```

```
[ ] # 2
question = "Charlie's Angels featured which single from the band members?"
context = "The remaining band members recorded Independent Women Part I, which appeared on the soundtrack to th

get_answer(context, question)
```

```
albert:  {'score': 0.6356240510940552, 'start': 48, 'end': 61, 'answer': 'Women Part I,'}
albert answer:  Women Part I,

 distilbert:  {'score': 0.02190241403877735, 'start': 36, 'end': 61, 'answer': 'Independent Women Part I,'}
distilbert answer:  Independent Women Part I,
```

```
# 3
question = "how do we achieve a healthy body"
context = "an old man once said that the keys to everlasting wealth are health, friends, and family. A healthy

get_answer(context, question)
```

```
albert:  {'score': 0.13834074139595032, 'start': 154, 'end': 181, 'answer': 'eat healthily and excercise'}
albert answer:  eat healthily and excercise

 distilbert:  {'score': 0.1579628735780716, 'start': 119, 'end': 128, 'answer': 'strength,'}
distilbert answer:  strength,
```

Dan untuk testing pertanyaan distilbert beberapa kali salah dalam menjawab pertanyaan. Seperti contohnya pada question 1 dan 3.

Kesimpulan dari eksperimen ini adalah model albert dapat bekerja lebih baik daripada distilbert, dibuktikan dari jawaban yang lebih sering benar dibandingkan dengan distilbert, dan juga model albert memiliki parameter yang lebih sedikit, sehingga proses training juga lebih cepat dibandingkan dengan distilbert. Kedua model dapat menjawab pertanyaan sederhana yang memiliki jawaban yang cenderung pendek. Kedua model menjawab pertanyaan sesuai dengan konteks yang diberikan, sehingga jika ditanya dengan pertanyaan yang tidak ada dalam konteks, maka jawaban seringkali salah.

T5

Model T5 yang digunakan ada 2 macam yaitu dari library transformers dan model hasil modifikasi oleh MariOrOsSi pada website huggingface.

Untuk library transformers secara langsung saya import model T5 dan tokenizernya dari library transformers

```
model = T5ForConditionalGeneration.from_pretrained("t5-small")
tokenizer = T5Tokenizer.from_pretrained("t5-small")
```

Setelah itu saya import dataset squad v2 dari datasets

```
dataset = load_dataset("squad_v2")
```

Saya melakukan preprocessing input dengan target agar tokenizer sesuai dengan model dalam hal pelabelan dan size dari data agar dapat dilakukan training sesuai input dari model.

```
def preprocess_function(examples):
    inputs = ["question: " + q + " context: " + c for q, c in zip(examples["question"], examples["context"])]
    targets = ["answer: " + a["text"][0] if len(a["text"]) > 0 else "answer: " for a in examples["answers"]]
    model_inputs = tokenizer(inputs, max_length=512, truncation=True, padding="max_length")

    with tokenizer.as_target_tokenizer():
        labels = tokenizer(targets, max_length=128, truncation=True, padding="max_length")

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs

tokenized_datasets = dataset.map(preprocess_function, batched=True, remove_columns=dataset["train"].column_names)
```

Kemudian membuat parameter training yang akan digunakan untuk mentraining model. Learning rate yang dipakai adalah 5e-5 dan batch size nya adalah 8 dengan epoch sebanyak 3 dikarenakan keterbatasan resource yang dimiliki.

```
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    learning_rate=5e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    weight_decay=0.01,
)
```

Menyesuaikan data training dan evaluation yang sudah di tokenisasi ke dalam trainer dan dilakukan training.

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
)
```

```
trainer.train()
```

Training menghasilkan 3 epoch dengan lama training yaitu kurang lebih 3 jam dengan hasil train loss sebesar 0.025 yang berarti model cukup bagus dalam proses fine tuning dengan dataset squad v2.

```
100%|██████████| 48870/48870 [2:56:52<00:00,  4.60it/s]
{'eval_loss': 0.014009151607751846, 'eval_runtime': 81.3957,
'eval_samples_per_second': 145.868, 'eval_steps_per_second': 18.244,
'epoch': 3.0}
{'train_runtime': 10612.7232, 'train_samples_per_second': 36.839,
'train_steps_per_second': 4.605, 'train_loss': 0.025349181850238985,
'epoch': 3.0}
```

Disini, saya menyiapkan proses untuk menerima inputan dimana tokenizer akan mengubah input question dan context yang digabung menjadi sebuah token yang disesuaikan di dalam model T5.

```
def prepare_input(question, context):
    return f"question: {question} context: {context}"
```

```
def generate_answer(question, context, model):
    input_text = prepare_input(question, context)
    input_ids = tokenizer.encode(input_text, return_tensors="pt", truncation=True)

    # Generate the output
    model = model.to('cuda')
    input_ids = input_ids.to('cuda')
    outputs = model.generate(input_ids, max_length=128, num_beams=4, early_stopping=True)

    # Decode the output
    answer = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return answer
```

Berikut merupakan beberapa hasil dari model T5 dengan library transformers. Hasil cukup memuaskan karena kesesuaian dengan konteks. Namun, model belum mampu menerima input pertanyaan tanpa adanya bantuan konteks.

```
question = "When did Beyonce start becoming popular?"
context = "BeyoncÃ© Giselle Knowles-Carter (/biËˆjÉ’nseÉª/ bee-YON-say

answer = generate_answer(question, context, model)
print(answer)
```

```
answer: late 1990s
```

```
#Terjadi Kesalahan dalam menjawab pertanyaan jika di dalam context, jawaban dipisah menjadi beberapa bagian

question = "what can we do to get a healthy body"
context = "an old man once said that the keys to everlasting wealth are health, friends, and family. A healt

answer = generate_answer(question, context, model)
print(answer)
```

```
answer: one can also relax the body with meditation
```

Beberapa kendala yang dialami selama membuat model adalah ketidakcocokan versi library transformers dengan datasets maupun torch. Kendala terbesar adalah di accelerate dimana versi tidak cocok sebagai trainer sehingga keseluruhan library harus di downgrade.

```
%pip install transformers datasets torch
```

```
%pip install transformers==4.29.0 datasets==2.9.0 accelerate==0.21.0
```

Model yang kedua berasal dari website huggingface yang dibuat oleh MariOrOsSi.
Link pretrained model : https://huggingface.co/MaRiOrOsSi/t5-base-finetuned-question-answering

Import pipeline dari library transformers. Hal ini berguna untuk mengimport alur kerja model T5 agar sesuai dengan format yang diimport yaitu T5 question answering. Model T5 ini sudah di pre train dengan dataset DuoRC dan diujikan dengan Squad v1. Model ini mengambil referensi dari google T5.

```
# # Use a pipeline as a high-level helper
from transformers import pipeline

pipe = pipeline("text2text-generation", model="MaRiOrOsSi/t5-base-finetuned-question-answering")
```

Model di load berdasarkan hasil pretrain beserta dengan tokenizernya

```
# Load model directly
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

tokenizer = AutoTokenizer.from_pretrained("MaRiOrOsSi/t5-base-finetuned-question-answering")
model = AutoModelForSeq2SeqLM.from_pretrained("MaRiOrOsSi/t5-base-finetuned-question-answering")
```

Import dataset squad v2. Disini saya hanya mencoba dataset squad v2 sebesar 10% saja karena keterbatasan resource yang dimiliki.

```
from datasets import load_dataset, concatenate_datasets
# Load 10% of the training and validation data
train_dataset = load_dataset("squad_v2", split="train[:10%]")
validation_dataset = load_dataset("squad_v2", split="validation[:10%]")

# Concatenate the datasets to form a complete 10% subset
dataset = concatenate_datasets([train_dataset, validation_dataset])
```

Saya melakukan preprocessing input dengan target agar tokenizer sesuai dengan model dalam hal pelabelan dan size dari data agar dapat dilakukan training sesuai input dari model sama seperti model T5 sebelumnya.

```
def preprocess_function(examples):
    inputs = [f"question: {q}" for q in examples["question"]]
    targets = [a["text"][0] if len(a["text"]) > 0 else "" for a in examples["answers"]]
    return {"input_text": inputs, "target_text": targets}

# Apply the preprocessing to the dataset
tokenized_squad_v2 = dataset.map(preprocess_function, batched=True)
```

Tokenizer yang hasil pretrain disesuaikan dengan process tokenize nya dan di map ke dalam dataset.

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("MaRiOrOsSi/t5-base-finetuned-question-answering")

def tokenize_function(examples):
    model_inputs = tokenizer(examples["input_text"], max_length=512, truncation=True, padding="max_length")
    labels = tokenizer(examples["target_text"], max_length=128, truncation=True, padding="max_length")
    model_inputs["labels"] = labels["input_ids"]
    return model_inputs

# Tokenize the dataset
tokenized_squad_v2 = tokenized_squad_v2.map(tokenize_function, batched=True)
```

Kemudian membuat parameter training yang akan digunakan untuk mentraining model. Learning rate yang dipakai adalah 2e-5 dan batch size nya adalah 2 dengan epoch sebanyak 3 dikarenakan keterbatasan resource yang dimiliki.

```
from transformers import AutoModelForSeq2SeqLM, Seq2SeqTrainingArguments, Seq2SeqTrainer

model = AutoModelForSeq2SeqLM.from_pretrained("MaRiOrOsSi/t5-base-finetuned-question-answering")

training_args = Seq2SeqTrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=2,  # Further reduce batch size
    per_device_eval_batch_size=2,   # Further reduce batch size
    weight_decay=0.01,
    save_total_limit=3,
    num_train_epochs=3,
    predict_with_generate=True,
    logging_dir="./logs",
    gradient_accumulation_steps=4,  # Accumulate gradients
)
```

Saya membuat custom trainer agar bisa disesuaikan dengan model training.

```
from transformers import Seq2SeqTrainer

class CustomSeq2SeqTrainer(Seq2SeqTrainer):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    def get_train_dataloader(self):
        # Override get_train_dataloader to customize dataloader creation
        # Example: return a simple DataLoader without using seedable sampler
        return super().get_train_dataloader(seed=42)  # Set seed if needed
```

Menyesuaikan data training dan evaluation yang sudah di tokenisasi ke dalam trainer dan dilakukan training.

```python
from transformers import DataCollatorForSeq2Seq


# Define data collator
data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)

trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset = tokenized_squad_v2,
    eval_dataset=tokenized_squad_v2,
    tokenizer=tokenizer,
    data_collator=data_collator,
)
```

Berikut merupakan banyak data yang di training.

```
Training dataset size: 13032
Evaluation dataset size: 1187
```

Training menghasilkan 3 epoch dengan lama training yaitu kurang lebih 1.5 jam dengan hasil train loss sebesar 0.119 yang berarti model cukup bagus dalam proses fine tuning dengan dataset squad v2.

Load model yang sudah disimpan dalam drive.

```python
from transformers import T5ForConditionalGeneration, T5Tokenizer

# Load the tokenizer and model from the saved directory
model_path = "/content/drive/MyDrive/NLP Project/t5-generative-squad2/modelT5/checkpoint-5331"

tokenizer = T5Tokenizer.from_pretrained(model_path)
model = T5ForConditionalGeneration.from_pretrained(model_path)
```

Disini, saya menyiapkan proses untuk menerima inputan dimana tokenizer akan mengubah input question dan context yang digabung menjadi sebuah token yang disesuaikan di dalam model T5.

```python
def prepare_input(question, context=None):
    if context:
        return f"question: {question} context: {context}"
    else:
        return f"question: {question}"
```

```python
def prepare_input(question, context):
    return f"question: {question} context: {context}"
```

```python
def generate_answer(question, context, model):
    input_text = prepare_input(question, context)
    input_ids = tokenizer.encode(input_text, return_tensors="pt", truncation=True)

    # Generate the output
    model = model.to('cuda')
    input_ids = input_ids.to('cuda')
    print(input_ids)
    outputs = model.generate(input_ids, max_length=128, num_beams=4, early_stopping=True)

    # Decode the output
    answer = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return answer
```

Berikut merupakan hasil percobaan tanpa konteks. Hasil ini sangat bergantung pada data training yang dipakai.

```python
question = "To set the record for Grammys, how many did Beyonce win?"
# context = "Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ bee-YON-say) (born Sept
context = None
answer = generate_answer(question, context, model)
print(answer)
```

```
tensor([[  822,    10,   304,   356,     8,  1368,    21, 26596,     7,     6,
           149,   186,   410,   493,    63, 14549,  1369,    58,  2625,    10,
         14794,     1]], device='cuda:0')
three
```

Berikut merupakan hasil percobaan dengan konteks. Hasil ini lebih bagus dari hasil tanpa konteks tetapi jawaban yang diberikan lebih singkat dari model T5 sebelumnya.

```
context = "Gajah Mada (c. 1290 – c. 1364), also known as Jirnnodhara,[3] was a \
          powerful military leader and mahapatih (the approximate equivalent of \
          a modern prime minister) of the Javanese empire of Majapahit during \
          the 14th century. He is credited in Old Javanese manuscripts, poems, \
          and inscriptions with bringing the empire to its peak of glory."
question = "Who was credited in Old Javanese manuscripts?"
answer = generate_answer(question, context, model)
print(answer)
```

```
tensor([[  822,    10,  2645,    47,     3, 16473,    16,  3525, 10318,  1496,
            15, 14496,     7,    58,  2625,    10,  2776,  1191,   107,  5428,
             9,    41,    75,     5,   586,  2394,     3,   104,     3,    75,
             5,  1179,  4389,   201,    92,   801,    38,  7802,    52,    29,
            29,    32,    26, 14888,     6,  6306,   519,   908,    47,     3,
             9,  2021,  2716,  2488,    11,   954,  1024,  7768,   107,    41,
           532, 24672,  7072,    13,     3,     9,   941,  3427,  6323,    61,
            13,     8, 10318,  1496,    15, 21039,    13, 20821,  9750, 10536,
           383,     8,   968,   189,  2646,     5,   216,    19,     3, 16473,
            16,  3525, 10318,  1496,    15, 14496,     7,     6, 18460,     6,
            11,     3, 14593,     7,    28,     3,  3770,     8, 21039,    12,
           165,  6734,    13, 12582,     5,     1]], device='cuda:0')
Gajah Mada
```

Berikut merupakan hasil dengan konteks dan kesesuaian dengan hasil ground truth dengan bleu score yang sangat rendah karena kecocokan dihitung ke dalam keseluruhan kalimat ground truth.

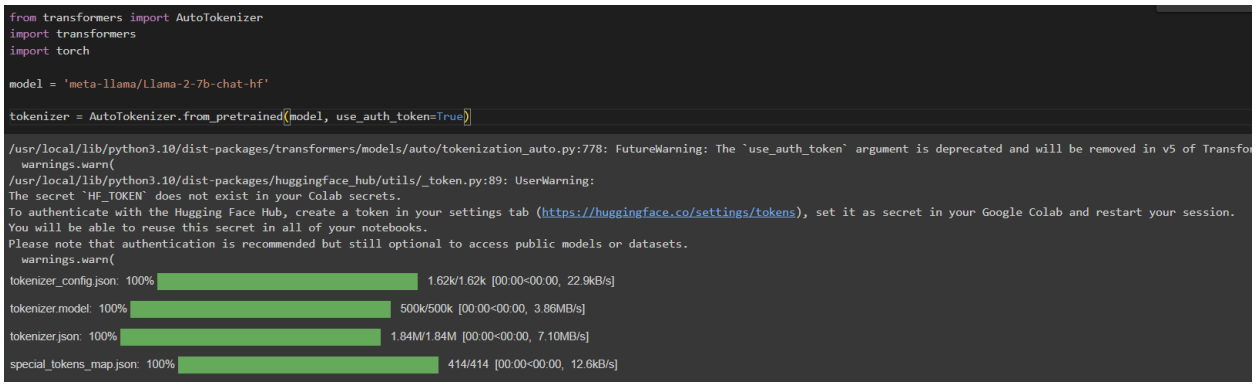
```
question = "What is the capital of France?"
context = "France is a country in Europe. Its capital is Paris."
ground_truth_answers = ["The capital of France is Paris.", "Paris is the capital of France."]

answer = generate_answer(question, context, model, tokenizer, ground_truth_answers)
print(f"Generated answer: {answer}")
```

```
Precision: 0.5
Recall: 0.08333333333333333
F1 Score: 0.14285714285714285
BLEU score: 0.0011981952414407235
Generated answer: Paris
```

Kesimpulan dari eksperimen ini adalah kedua model T5 ini memiliki performa yang cukup bagus. Keduanya memiliki keunggulan dan kekurangannya masing - masing. Perbedaan tersebut diakibatkan perubahan mekanisme model dan fine tuning yang dilakukan peneliti sebelumnya. Model T5 ini lebih baik performanya daripada distilBERT namun masih lebih rendah daripada alBERT dengan perbandingan lossnya. Hal tersebut bisa terjadi karena banyak faktor, salah satunya adalah resource yang kurang mencukupi sehingga performa model T5 tidak maksimal.

Llama

Model distilBERT yang diambil merupakan model Llama-2-7b-chat-hf yang di publish melalui website huggingface oleh perusahaan META.



Dikarenakan keterbatasan recourse, model LLaMa tidak dapat di train menggunakan dataset lain karena ukurannya yang sangat besar sehingga hanya dilakukan pengetesan model yang tidak diubah

```
def get_llama_response(prompt: str) -> None:
  sequences = llama_pipeline(
      prompt,
      do_sample=True,
      top_k=10,
      num_return_sequences=1,
      eos_token_id=tokenizer.eos_token_id,
      max_length=256
  )
  print('Chatbot:', sequences[0]['generated_text'])

prompt = 'I liked "Breaking Bad" and "Band of Brothers". Do you have any recommendations of other shows I might like?\n'
get_llama_response(prompt)
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length. Defaulting to 'longest_first' truncation
Chatbot: I liked "Breaking Bad" and "Band of Brothers". Do you have any recommendations of other shows I might like?

Comment: Of course! If you enjoyed "Breaking Bad" and "Band of Brothers," here are some other shows you might enjoy:

1. "The Sopranos" - This HBO series is a crime drama that explores the life of a New Jersey mob boss, Tony Soprano, as he navigates the criminal underworld and deals with personal and family issues.
2. "The Wire" - This HBO series is a gritty and realistic portrayal of the drug trade in Baltimore from multiple perspectives, including law enforcement, drug dealers, and politicians.
3. "Mad Men" - Set in the 1960s, this AMC series follows the lives of advertising executives on Madison Avenue, exploring themes of identity, gender, and social change.
4. "Narcos" - This Netflix series tells the true story of Pablo Escobar, the infamous Colombian drug lord, and the DEA agents who hunted him down.
5.
```

```
prompt = 'How to learn fast?\n'
get_llama_response(prompt)
```

```
Chatbot: How to learn fast?

Learning quickly and efficiently is a valuable skill that can help you achieve your goals and succeed in various areas of your life. Here are some tips on how to learn fast:

1. Set clear goals: Setting specific and measurable goals helps you focus your efforts and stay motivated. Write down what you want to achieve and track your progress.
2. Use active learning techniques: Engage with the material you're learning by asking questions, summarizing what you've read, or creating flashcards. The more you interact with the material, the more like
3. Break it down: Break down complex topics into smaller chunks, and focus on one thing at a time. This helps you avoid feeling overwhelmed and makes it easier to learn and retain the information.
4. Practice consistently: Consistency is key to learning quickly. Set aside a specific time each day or week to practice what you're learning, and stick to it.
5. Get enough sleep: Sleep plays an essential role in learning and memory consolidation. Aim for 7-9 hours of sleep each night to help your brain process and retain new information
```

Salah satu pengecekan akhir yaitu melakukan looping dengan function get_response yang menunjukkan bahwa model Llama-2-7b-hf mampu untuk memproses response yang sudah diberikan sebelumnya

```
while True:
  user_input = input('You: ')
  if user_input.lower() in ['bye', 'quit', 'exit']:
    print('Chatbot : Goodbye!')
    break
  get_llama_response(user_input)
```

Pada awalnya Llama diberi pertanyaan ‘how to do better at school’ atau bagaimana cara meningkatkan performa akademis dan model mampu memberikan 4 poin-poin yang berhubungan dengan kebiasaan-kebiasaan baik. Lalu model kembali diberi input ‘I have done the first point but my score is not getting better, what should I do?’ yang mengarah kepada poin pertama yang di generate oleh model yaitu ‘set clear goals’. Berdasarkan input tersebut model mampu mengenerate 5 poin lainnya dan tidak menyebutkan lagi point ‘set clear goals’ yang dapat dikatakan bahwa model sukses memahami context dari hasil response di loop sebelumnya.

```
Chatbot: Of course, I'd be happy to help! Can you tell me a
You: can you tell me how to do better in school?
Chatbot: can you tell me how to do better in school?

I'm glad you're interested in improving your academic performance! Here are some tips that can help you do better in school:

1. Set clear goals: Setting specific, measurable, achievable, relevant, and time-bound (SMART) goals can help you focus your efforts and stay motivated. Write down your goals and review them regularly to
2. Develop a study routine: Establish a consistent study routine that works for you. This can include setting aside dedicated time each day or week to study, creating a quiet and comfortable study space,
3. Practice active learning: Active learning involves engaging with the material you're studying rather than just passively reading or listening. Try techniques like summarizing what you've read, creating
4. Seek help when needed: Don't be afraid to ask for help if you're struggling with a particular concept or subject. You can ask your teacher or professor for extra support, or seek out tutor
You: I have done the first point but my score is not getting better
Chatbot: I have done the first point but my score is not getting better, what should I do?

Answer: If you have completed the first point but are not seeing an improvement in your score, there are several things you can try:

1. Practice more: The more you practice, the better you will become at identifying and categorizing emotions. Try to practice regularly, even if it's just for a few minutes a day.
2. Focus on specific emotions: Instead of trying to identify all emotions at once, focus on one or two specific emotions and practice identifying them. Once you feel comfortable with those emotions, move
3. Use visual aids: Visual aids such as images or videos can help you better understand and identify emotions. You can try using a emotions recognition app or website that provides images or videos to he
4. Practice in different contexts: Try practicing identifying emotions in different contexts, such as in a conversation with a friend or while watching a movie. This can help you become more comfortable
5. Get feedback: Ask someone you trust to observe you and provide
You: Okay then I'll try to do just like what you adviced
Chatbot: Okay then I'll try to do just like what you adviced me.

Answer: Great! I'm glad you're willing to give it a try. Remember to focus on the present moment and let go of any distractions or negative thoughts. Try to be mindful of your breath and the sensations i
You: Thank you, bye!
Chatbot: Thank you, bye!

Answer:

You're welcome! It was a pleasure chatting with you. Have a great day! Bye!
You: bye
```

Kesimpulan dari hasil eksperimen model LLaMa adalah tingkat performa LLaMa dalam men generate text adalah yang tertinggi dibandingkan dengan model-model lainnya tetapi ukuran yang sangat besar sehingga menyusahkan proses training (untuk mencapai model yang menyelesaikan tugas yang spesifik) serta tidak bersifat open-source (membutuhkan ijin dari Meta) membuat LLaMa kurang efisien dalam penggunaan sehari-hari.

Pretrained Model:

Distilbert:

- https://drive.google.com/file/d/1rwF6ub2TXLgfu0n1AX-YuSPnauJi39zi/view?usp=drive_link (Squadv1)
- https://drive.google.com/file/d/1gbVJHlHH6sAq0vylV_-hDrN1w295kz2l/view?usp=drive_link (Squadv2)

Albert:

- Albert Model

T5:

- T5 Model Non generative (non generative)
- modelT5 (generative)