Lampiran 3. Kode Program Modeling BERT

```
!pip install transformers==4.18.0

import json
import numpy as np
import pandas as pd
import random
from matplotlib import pyplot as plt
import seaborn as sns
from wordcloud import WordCloud,STOPWORDS
import missingno as msno
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from keras.preprocessing import text
import keras
from keras.models import Sequential
from keras.layers import Dense,Embedding,LSTM,Dropout
from keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.preprocessing.sequence import pad_sequences
import nltk
nltk.download('punkt')
from nltk import word_tokenize
from nltk.stem import PorterStemmer
import torch
from torch.utils.data import Dataset
from transformers import AutoTokenizer,
TFAutoModelForSequenceClassification
from transformers import pipeline
from transformers import DistilBertTokenizerFast
from transformers import BertForSequenceClassification, BertTokenizerFast
from transformers import TFDistilBertForSequenceClassification, TFTrainer,
TFTrainingArguments
from transformers import BertTokenizer, TFBertForSequenceClassification,
BertConfig
from transformers import TrainingArguments, Trainer

def load_json_file(filename):
    with open(filename) as f:
        file = json.load(f)
    return file

filename = '/content/dataset_chatbot.json'

intents = load_json_file(filename)

def create_df():
```

```
    df = pd.DataFrame({
        'Pattern' : [],
        'Tag' : []
    })

    return df

df = create_df()
df

def extract_json_info(json_file, df):

    for intent in json_file['intents']:

        for pattern in intent['patterns']:

            sentence_tag = [pattern, intent['tag']]
            df.loc[len(df.index)] = sentence_tag

    return df

df = extract_json_info(intents, df)
df.head()

df2 = df.copy()
df2.head()

def print_shape_df(df, ds_name="df"):
    print(f"{ds_name} dataset has {df.shape[0]} rows and {df.shape[1]}
columns")

print_shape_df(df, "Chatbot")

def print_dfInfo(df, ds_name="df"):
    print(f"The info of {ds_name} dataset\n")
    print(df.info())

print_dfInfo(df, "Chatbot")

def num_classes(df, target_col, ds_name="df"):
    print(f"The {ds_name} dataset has {len(df[target_col].unique())} classes")

num_classes(df, 'Tag', "Chatbot")

def check_null(df, ds_name='df'):
    print(f"Null Values in each col in the {ds_name} dataset:\n")
    print(df.isnull().sum())
```

```
check_null(df, "Chatbot")

df2.head()
labels = df2['Tag'].unique().tolist()
labels = [s.strip() for s in labels]
labels

num_labels = len(labels)
id2label = {id:label for id, label in enumerate(labels)}
label2id = {label:id for id, label in enumerate(labels)}

df2['labels'] = df2['Tag'].map(lambda x: label2id[x.strip()])
df2.head()

x = list(df2['Pattern'])
X[:5]

y = list(df2['labels'])
y[:5]

X_train,X_test,y_train,y_test = train_test_split(X,y,random_state = 123)
model_name = "indolem/indobert-base-uncased"
max_len = 256

tokenizer = BertTokenizer.from_pretrained(model_name,
                                max_length=max_len)

model = BertForSequenceClassification.from_pretrained(model_name,
                                    num_labels=num_labels,
                                    id2label=id2label,
                                    label2id = label2id)
train_encoding = tokenizer(X_train, truncation=True, padding=True)
test_encoding = tokenizer(X_test, truncation=True, padding=True)

full_data = tokenizer(X, truncation=True, padding=True)
class DataLoader(Dataset):

    def __init__(self, encodings, labels):

        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):

        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item
```

```
    def __len__(self):

        return len(self.labels)

train_dataloader = DataLoader(train_encoding, y_train)
test_dataloader = DataLoader(test_encoding, y_test)

fullDataLoader = DataLoader(full_data, y_test)

def compute_metrics(pred):

    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, preds,
average='macro')
    acc = accuracy_score(labels, preds)

    return {
        'Accuracy': acc,
        'F1': f1,
        'Precision': precision,
        'Recall': recall
    }

training_args = TrainingArguments(
    output_dir='./output',
    do_train=True,
    do_eval=True,
    num_train_epochs=100,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=16,
    warmup_steps=100,
    weight_decay=0.05,
    logging_strategy='steps',
    logging_dir='./multi-class-logs',
    logging_steps=50,
    evaluation_strategy="steps",
    eval_steps=50,
    save_strategy="steps",
    load_best_model_at_end=True
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataloader,
    eval_dataset=test_dataloader,
```

```
    compute_metrics= compute_metrics
)

trainer.train()

q=[trainer.evaluate(eval_dataset=df2) for df2 in [train_dataloader,
test_dataloader]]

pd.DataFrame(q, index=["train","test"]).iloc[:,:5]

model_path = "chatbot"
trainer.save_model(model_path)
tokenizer.save_pretrained(model_path)

!zip -r /content/model.zip /content/chatbot
```

Lampiran 4. Kode Program API Chatbot dengan Flask

```
from flask import Flask, request, jsonify
from transformers import BertForSequenceClassification, BertTokenizerFast,
pipeline
from flask_cors import CORS
import pandas as pd
import random
import json

# Load intents dataset
def load_json_file(filename):
    with open(filename) as f:
        return json.load(f)

filename = 'dataset_chatbot.json'
intents = load_json_file(filename)

# Create DataFrame from intents
def create_df(intents):
    patterns = []
    tags = []
    for intent in intents['intents']:
        for pattern in intent['patterns']:
            patterns.append(pattern)
            tags.append(intent['tag'])

    df = pd.DataFrame({
        'Pattern': patterns,
        'Tag': tags
    })
```

```
    return df

df = create_df(intents)

labels = df['Tag'].unique().tolist()
labels = [s.strip() for s in labels]

num_labels = len(labels)
id2label = {id: label for id, label in enumerate(labels)}
label2id = {label: id for id, label in enumerate(labels)}

model_path = "model/content/chatbot"
model = BertForSequenceClassification.from_pretrained(model_path,
num_labels=num_labels)
tokenizer = BertTokenizerFast.from_pretrained(model_path)
classifier = pipeline("text-classification", model=model, tokenizer=tokenizer)

# Setting up the API
app = Flask(__name__)
CORS(app)

# @app.route("/chat", methods=["POST"])
# def chat():
#     input_text = request.json["input_text"]
#     # Use the classifier pipeline to get the text classification result
#     result = classifier(input_text)
#     predicted_label = result[0]["label"]

#     # Directly use the label if the classifier returns the correct intent
#     if predicted_label in label2id:
#         intent_tag = predicted_label

#     else:
#         # Handle potential mismatches or unexpected label formats
#         intent_tag = id2label[int(predicted_label.split('_')[-1])] if
predicted_label.split('_')[-1].isdigit() else 'unknown'

#     # Get a random response from the corresponding intent
#     response = "maaf aku tidak paham."
#     for intent in intents['intents']:
#         if intent['tag'] == intent_tag:
#             response = random.choice(intent['responses'])
#             break

#     return jsonify({"response_text": response})

@app.route("/chat", methods=["POST"])
def chat():
```

```
    input_text = request.json["input_text"]
    # Use the classifier pipeline to get the text classification result
    result = classifier(input_text)
    score = result[0]['score']
    predicted_label = result[0]["label"]

    if score < 0.8:
        response = "Mohon maaf saya tidak dapat menjawab pertanyaan itu karena
tidak ada dalam data saya."
    else:
        # Directly use the label if the classifier returns the correct intent
        if predicted_label in label2id:
            intent_tag = predicted_label
        else:
            # Handle potential mismatches or unexpected label formats
            intent_tag = id2label[int(predicted_label.split('_')[-1])] if
predicted_label.split('_')[-1].isdigit() else 'unknown'

        # Get a random response from the corresponding intent
        for intent in intents['intents']:
            if intent['tag'] == intent_tag:
                response = random.choice(intent['responses'])
                break

    return jsonify({"response_text": response})

if __name__ == "__main__":
    app.run()
```

Lampiran 5. Kode Program Aplikasi Chatbot dengan Flutter

MAIN.DART

```
import 'package:flutter/material.dart';
import 'chat_screen.dart';
import 'about_screen.dart';

void main() {
  runApp(ChatApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Daftar Nama',
      theme: ThemeData(
        primarySwatch: Colors.blue,
```

```
      ),
      home: NamaList(),
    );
  }
}

class NamaList extends StatelessWidget {
  // Fungsi untuk menghasilkan daftar nama menggunakan looping
  List<String> generateNama(int count) {
    List<String> namaList = [];
    for (int i = 1; i <= count; i++) {
      namaList.add('Nama $i');
    }
    return namaList;
  }

  @override
  Widget build(BuildContext context) {
    final List<String> nama = generateNama(10);

    return Scaffold(
      appBar: AppBar(
        title: Text('Daftar Nama'),
      ),
      body: ListView.builder(
        itemCount: nama.length,
        itemBuilder: (context, index) {
          return ListTile(
            title: Text(nama[index]),
          );
        },
      ),
    );
  }
}

class ChatApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'ChatBot App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: HomeScreen(),
      routes: {
        '/chat': (context) => ChatScreen(),
        '/about': (context) => AboutScreen()
```

```
      },
    );
  }
}

class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Home'),
      ),
      backgroundColor: Color(0xFFA5DD9B), // Set background color here
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            ElevatedButton(
              onPressed: () {
                Navigator.pushNamed(context, '/chat');
              },
              child: Text('ChatBot'),
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: () {
                Navigator.pushNamed(context, '/about');
              },
              child: Text('About'),
            ),
          ],
        ),
      ),
    );
  }
}
```

CHAT_SCREEN.DART

```
import 'package:flutter/material.dart';
import 'chat_screen.dart';
import 'about_screen.dart';

void main() {
  runApp(ChatApp());
}

class MyApp extends StatelessWidget {
  @override
```

```
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Daftar Nama',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: NamaList(),
    );
  }
}

class NamaList extends StatelessWidget {
  // Fungsi untuk menghasilkan daftar nama menggunakan looping
  List<String> generateNama(int count) {
    List<String> namaList = [];
    for (int i = 1; i <= count; i++) {
      namaList.add('Nama $i');
    }
    return namaList;
  }

  @override
  Widget build(BuildContext context) {
    final List<String> nama = generateNama(10);

    return Scaffold(
      appBar: AppBar(
        title: Text('Daftar Nama'),
      ),
      body: ListView.builder(
        itemCount: nama.length,
        itemBuilder: (context, index) {
          return ListTile(
            title: Text(nama[index]),
          );
        },
      ),
    );
  }
}

class ChatApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'ChatBot App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
```

```
      ),
      home: HomeScreen(),
      routes: {
        '/chat': (context) => ChatScreen(),
        '/about': (context) => AboutScreen()
      },
    );
  }
}

class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Home'),
      ),
      backgroundColor: Color(0xFFA5DD9B), // Set background color here
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            ElevatedButton(
              onPressed: () {
                Navigator.pushNamed(context, '/chat');
              },
              child: Text('ChatBot'),
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: () {
                Navigator.pushNamed(context, '/about');
              },
              child: Text('About'),
            ),
          ],
        ),
      ),
    );
  }
}
```

ABOUT_SCREEN.DART

```
import 'package:flutter/material.dart';

class AboutScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
```

```
    return Scaffold(
      appBar: AppBar(
        title: Text('Tentang El-Huda Bot'),
      ),
      backgroundColor: Color(0xFFA5DD9B), // Set background color here
      body: SingleChildScrollView(
        padding: EdgeInsets.all(20),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(
              'El-Huda Bot: Asisten Fiqih Berbasis NLP dengan Model BERT',
              style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
            ),
            SizedBox(height: 20),
            Text(
              'Penjelasan Singkat',
              style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
            ),
            SizedBox(height: 10),
            Text(
              '''
              El-Huda Bot adalah asisten cerdas berbasis Natural Language Processing (NLP) yang dirancang untuk menjawab pertanyaan seputar ilmu fiqih berdasarkan kitab Fathul Qorib. Bot ini memanfaatkan model BERT (Bidirectional Encoder Representations from Transformers) untuk memahami dan menghasilkan jawaban yang relevan dan akurat.
              ''',
              style: TextStyle(fontSize: 16),
            ),
            SizedBox(height: 20),
            Text(
              'Komponen Utama El-Huda Bot',
              style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
            ),
            SizedBox(height: 10),
            Text(
              '''
              1. Natural Language Processing (NLP)
                - Definisi: NLP adalah cabang kecerdasan buatan yang berfokus pada interaksi antara komputer dan bahasa manusia.
                - Peran dalam El-Huda Bot: Memungkinkan bot untuk memproses dan memahami pertanyaan yang diajukan oleh pengguna dalam bahasa manusia.

              2. Model BERT
```

- Definisi: BERT adalah model transformer yang dilatih untuk memahami konteks kata dalam teks dengan melihat dua arah (kiri dan kanan) sekaligus.
- Peran dalam El-Huda Bot: Digunakan untuk memahami pertanyaan pengguna dan mencari jawaban yang tepat dari teks kitab Fathul Qorib.

3. Kitab Fathul Qorib
- Definisi: Fathul Qorib adalah kitab fiqih klasik yang sering digunakan sebagai referensi dalam mempelajari hukum-hukum Islam.
- Peran dalam El-Huda Bot: Digunakan sebagai basis pengetahuan untuk menjawab pertanyaan terkait fiqih.
'''
style: TextStyle(fontSize: 16),
),
SizedBox(height: 20),
Text(
'Langkah-Langkah Pembuatan El-Huda Bot',
style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
),
SizedBox(height: 10),
Text(
'''
1. Persiapan Data
- Pengumpulan Teks Kitab: Kumpulkan teks kitab Fathul Qorib dalam format digital.
- Pra-pemrosesan: Lakukan tokenisasi, pembersihan teks, dan anotasi untuk memudahkan pemrosesan oleh model NLP.

2. Pelatihan Model BERT
- Pra-pelatihan: Gunakan versi pra-terlatih dari BERT yang kemudian disesuaikan (fine-tuned) dengan teks Fathul Qorib.
- Fine-Tuning: Latih model BERT dengan teks Fathul Qorib untuk memahami konteks dan detail hukum-hukum fiqih.

3. Pengembangan Bot
- Integrasi NLP dan BERT: Integrasikan model BERT yang sudah dilatih dengan sistem NLP untuk memproses input dari pengguna dan menghasilkan jawaban.
- Pengujian dan Validasi: Uji bot dengan berbagai pertanyaan untuk memastikan keakuratan dan relevansi jawaban yang diberikan.

4. Implementasi dan Penggunaan
- Deployment: Implementasikan bot di platform yang mudah diakses oleh pengguna, seperti situs web atau aplikasi mobile.
- Penggunaan: Pengguna dapat mengetik pertanyaan seputar fiqih, dan El-Huda Bot akan memberikan jawaban berdasarkan teks Fathul Qorib.
''',
style: TextStyle(fontSize: 16),

```
          ),
          SizedBox(height: 20),
          Text(
            'Contoh Penggunaan El-Huda Bot',
            style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
          ),
          SizedBox(height: 10),
          Text(
            '''
            1. Pertanyaan Pengguna
              - "Apa hukum puasa bagi orang yang sedang sakit menurut Fathul Qorib?"

            2. Proses di El-Huda Bot
              - Analisis Pertanyaan: Bot menggunakan NLP untuk memahami maksud pertanyaan.
              - Pencarian Jawaban: Model BERT mencari teks yang relevan dalam kitab Fathul Qorib.
              - Penyusunan Jawaban: Bot menyusun jawaban yang mudah dipahami berdasarkan teks yang ditemukan.

            3. Jawaban Bot
              - "Menurut Fathul Qorib, orang yang sedang sakit yang khawatir sakitnya bertambah parah boleh tidak berpuasa dan menggantinya di hari lain setelah sembuh."
            ''',
            style: TextStyle(fontSize: 16),
          ),
          SizedBox(height: 20),
          Text(
            'Keunggulan El-Huda Bot',
            style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
          ),
          SizedBox(height: 10),
          Text(
            '''
            - Akurasi dan Relevansi: Memanfaatkan kekuatan NLP dan BERT untuk memberikan jawaban yang akurat dan relevan.
            - Ketersediaan dan Aksesibilitas: Dapat diakses kapan saja dan di mana saja, memudahkan pengguna untuk mendapatkan jawaban fiqih tanpa harus membuka kitab fisik.
            - Efisiensi: Mempercepat proses pencarian jawaban dari kitab klasik yang biasanya memakan waktu jika dilakukan secara manual.
            ''',
            style: TextStyle(fontSize: 16),
          ),
        ],
      ),
```

```
      ),
    );
  }
}
```