BAB III
PELAKSANAAN KERJA PROFESI

3.1 Bidang Kerja

Pelaksanaan program MBKM dilakukan di kantor yang berletak di Roxy, di JL. KH Hasyim Ashari 125 Pus Niaga Roxy Mas Bl D-2/36-37-3811220, RT.5/RW.8, Cideng, Gambir, Central Jakarta City, Jakarta 10150. Praktikan bekerja sebagai DevOps, dalam divisi Software Development dan Research and Development. Dalam program ini, proyek yang dikerjakan oleh praktikan adalah pengembangkan *extension* pada Visual Studio Code (VS Code), yang berfungsi sebagai *code copilot* atau pembantu dalam coding di editor secara lokal.

Proyek ini menerapkan Large Language Model (LLM) untuk menghasilkan teks berupa kode secara dinamis. *Extension* yang dikembangkan berfungsi sebagai pembantu coding yang pintar sehingga dapat memberikan saran maupun prediksi yang dapat menyelesaikan potongan kode berdasarkan kode yang ada di file tertentu.

Model yang diterapkan berasal dari Meta dengan nama model “meta-llama/Llama-3.2-1B” . Model ini merupakan model yang sudah dilatih sebelumnya dengan banyak data berupa teks baik dari multi-bahasa, kode pemrograman, percakapan, berita, artikel ilmiah, dan lain-lain. Penggunaan model Pretrained Model ini memiliki hal yang harus diperhatikan yaitu data pelatihan pada model ini masih luas sehingga memerlukan tuning atau penyesuaian lanjutan sehingga dapat memfokuskan untuk menghasilkan kode. Tuning atau penyesuaian lanjutan model diperlukan agar teks yang dihasilkan lebih efisien karena model akan mengakses dataset yang lebih relevan mengenai kode alih-alih tanpa dilakukan penyesuaian yang memerlukan akses data lebih luas dan kurang relevan.

Praktikan mengembangkan extension secara lokal ini dengan bahasa pemrograman Typescript dan Python. Bahasa Typescript sendiri merupakan bahasa pemrograman yang dikembangkan oleh Microsoft. Typescript sendiri

merupakan sebuah superset dari bahasa Javascript. Artinya, semua kode Javascript yang valid merupakan kode Typescript yang valid.

Typescript cenderung lebih kaku dan statis, fitur-fitur yang lebih dibandingkan dengan Javascript, sehingga memungkinkan pengembang untuk mendefinisikan sebuah tipe data, variabel, fungsi, objek, dan sebagainya sebagai eksplisit. Keuntungan dari pengkodean Typescript sebagai berikut :

1. Pendeteksi kesalahan lebih awal disebabkan oleh tipe data yang eksplisit sehingga kesalahan dapat terdeteksi saat penulisan kode maupun sebelum aplikasi berjalan.
2. Autocompletion dan Refactoring yang membuat editor atau aplikasi yang mendukung Typescript ini lebih baik dalam memberikan saran kode yang lebih baik dan Refactoring lebih mudah.

Pengembangan Extension di *Visual Studio Code* sering menggunakan bahasa Typescript karena berbagai alasan seperti :

1. Integrasi yang kuat dengan bahasa Javascript di mana *Visual Studio Code* menggunakan bahasa inti yaitu Javascript. Pengembang yang menuliskan kode Typescript serta fitur-fitur tambahan ini tetap kompatibel dengan platform Javascript yang digunakan pada Visual Studio Code.
2. Keamanan dan Kualitas Kode lebih baik sehingga pengembang dapat mengetahui kesalahan lebih awal dan menuliskan kode lebih terstruktur dan dapat dipelihara. Ini dapat berguna untuk pengembangan Extension dengan banyak *Application Programming Interface* (*API*) dan fitur-fitur di *Visual Studio Code* yang kompleks.

Sebagai tambahan, Python adalah bahasa pemrograman tingkat tinggi yang dikenal karena sintaksnya tergolong mudah dibaca dan dipahami serta kompatibel dengan berbagai jenis aplikasi.

3.2 Pelaksanaan Kerja

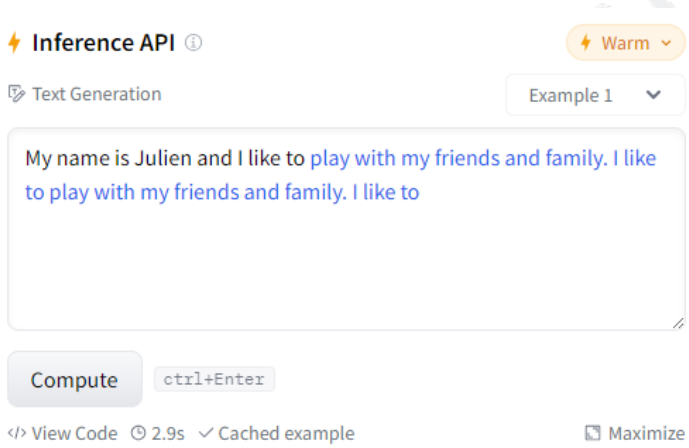
Praktikan mendapatkan sebuah proyek dari pembimbing kerja dan disetujui oleh penanggung jawab atau manager divisi *Software Development* untuk merancang dan mengembangkan *extension* sederhana untuk code copilot

atau pembantu *coding* di aplikasi editor *coding* VS.Code. Pembimbing mengurutkan apa saja yang harus dikerjakan dimulai dari pembelajaran dan analisis system, perancangan, pengembangan, hingga pengujian serta proses pelatihan atau *fine-tuning* model.

3.2.1 Analisis Sistem

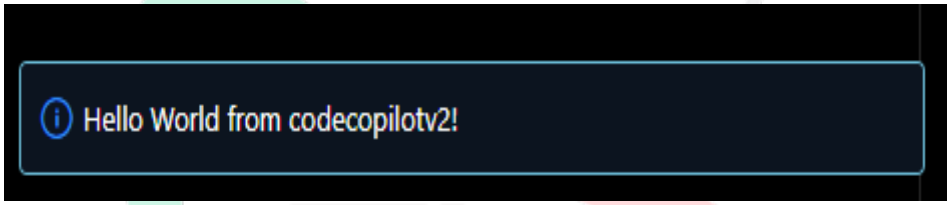
Praktikan memulai aktivitas praktik kerja dengan mempelajari sebuah mesin virtual yang diberikan oleh mentor atau pembimbing kerja. Praktikan menggunakan sebuah mesin virtual atau Virtual Machine yang disediakan oleh perusahaan dengan ukuran penyimpanan 500 Giga Byte. Praktikan menghubungkan perangkat kantor berupa laptop ke mesin virtual dengan *Secure Shell* (*SSH*) lalu diakses pada aplikasi pihak ketiga yaitu MobaXterm. Setelah akses ke mesin virtual, praktikan melakukan instalasi yang diperlukan seperti instalasi envinrontment Python hingga *tools-tools* berupa library pada Python. Environment yang diinstal ini merupakan lingkungan kerja secara lokal diakibatkan penggunaan mesin virtual itu sendiri.

Praktikan mencoba untuk menganalisa sistem yang diperlukan untuk dapat menghasilkan kode pada *extension* yang ingin dikembangkan. Analisa dimulai dengan praktikan langsung menguji model yang diusulkan oleh pembimbing kerja. Model yang diusulkan ini cukup untuk menghasilkan text yang baik berdasarkan dari sumber daya terbatas (sumber daya komputasi pada laptop kantor). Praktikan mencoba untuk dapat menghasilkan text berupa percakapan singkat dan berhasil untuk menghasilkan teks sehingga dapat mencoba untuk ke tahap selanjutnya yaitu pembuatan *extension*.



Gambar 3. 1 - Contoh hasilan teks dengan model pada https://huggingface.co/meta-llama/Llama-3.2-1B

Setelah menguji dalam menghasilkan teks dengan model, praktikan mencoba untuk mengembangkan sebuah *extension* secara lokal dengan memanfaatkan tools yaitu yeoman dan generator code pada node.js sebagai library atau dependencies untuk membuat *extension* sederhana. Setelah praktikan inisiasi dalam pembuatan *extension*, *extension* sederhana sudah dibuatkan dari generator code di mana pengguna dapat menampilkan notifikasi berupa pengantar "*Hello World*" seperti berikut :



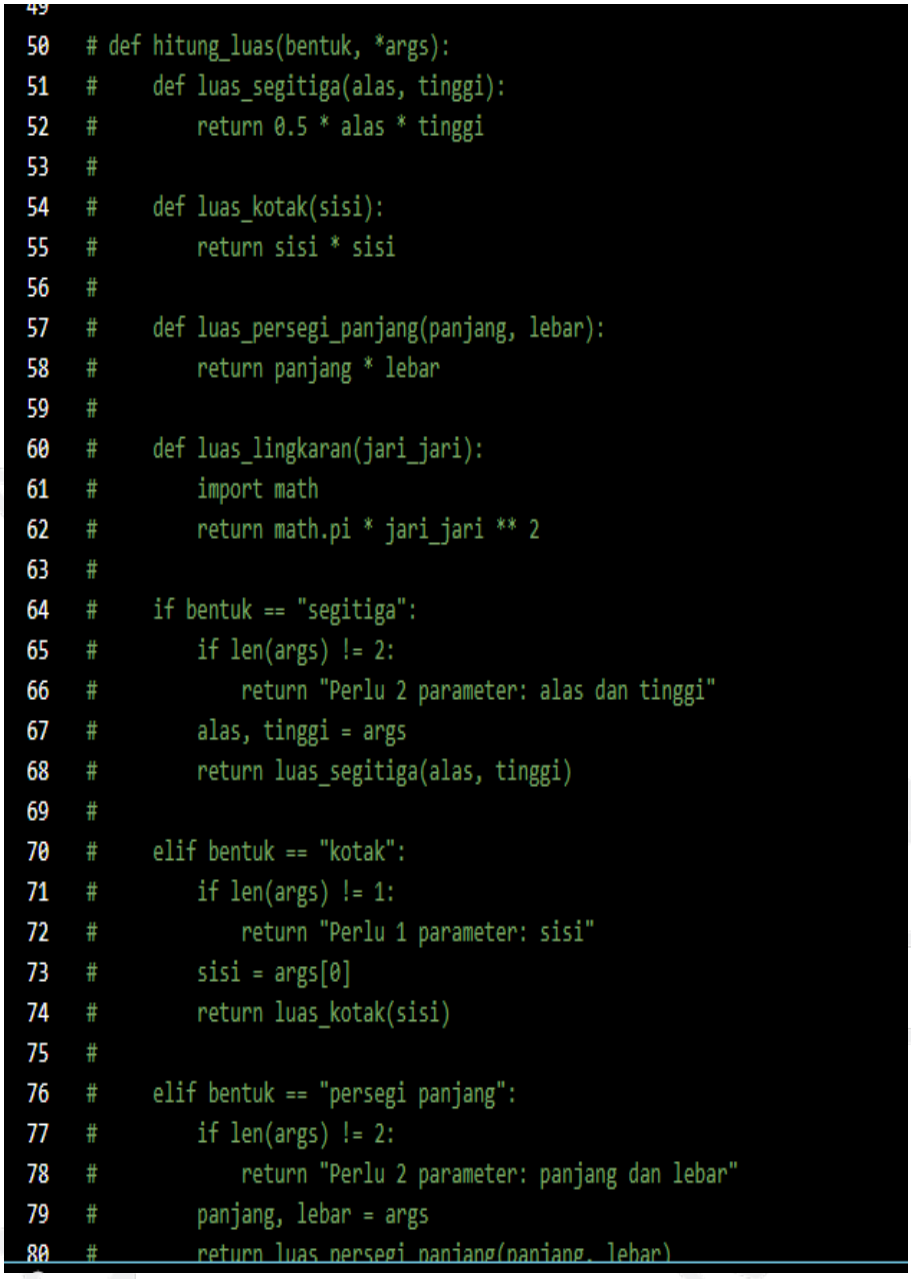
Gambar 3. 2 - Notifikasi *Hello World* ketika menjalankan *extension* sederhana di *Visual Studio Code*

3.2.2 Perancangan

Setelah yakin bahwa *extension* dapat berjalan, praktikan mencoba untuk mengembangkan *extension* dimulai dari membaca kode dari file tertentu dan akan dituliskan kembali di bawah dari kode yang sudah dibaca sebelumnya. Kode yang dituliskan kembali ke dalam file ini dalam bentuk comment code. Hasil dari pengembangan sementara pada *extension* seperti di gambar bawah ini.

```
1   def hitung_luas(bentuk, *args):
18          alas, tinggi = args
19          return luas_segitiga(alas, tinggi)
20
21      elif bentuk == "kotak":
22          if len(args) != 1:
23              return "Perlu 1 parameter: sisi"
24          sisi = args[0]
25          return luas_kotak(sisi)
26
27      elif bentuk == "persegi panjang":
28          if len(args) != 2:
29              return "Perlu 2 parameter: panjang dan lebar"
30          panjang, lebar = args
31          return luas_persegi_panjang(panjang, lebar)
32
33      elif bentuk == "lingkaran":
34          if len(args) != 1:
35              return "Perlu 1 parameter: jari-jari"
36          jari_jari = args[0]
37          return luas_lingkaran(jari_jari)
38
39      else:
40          return "Bentuk tidak dikenali"
41
42  # Contoh penggunaan fungsi:
43  print(hitung_luas("segitiga", 10, 5))          # 25.0
44  print(hitung_luas("kotak", 4))                 # 16
45  print(hitung_luas("persegi panjang", 4, 6))    # 24
46  print(hitung_luas("lingkaran", 7))             # 153.93804002589985
47  print(hitung_luas("lingkaran", 7, 8))          # Perlu 1 parameter: jari-jari
48
```

Gambar 3. 3 - File dengan kode yang ingin dijadikan kode comment pada keseluruhan isi file



```
# def hitung_luas(bentuk, *args):
#     def luas_segitiga(alas, tinggi):
#         return 0.5 * alas * tinggi
#
#     def luas_kotak(sisi):
#         return sisi * sisi
#
#     def luas_persegi_panjang(panjang, lebar):
#         return panjang * lebar
#
#     def luas_lingkaran(jari_jari):
#         import math
#         return math.pi * jari_jari ** 2
#
#     if bentuk == "segitiga":
#         if len(args) != 2:
#             return "Perlu 2 parameter: alas dan tinggi"
#         alas, tinggi = args
#         return luas_segitiga(alas, tinggi)
#
#     elif bentuk == "kotak":
#         if len(args) != 1:
#             return "Perlu 1 parameter: sisi"
#         sisi = args[0]
#         return luas_kotak(sisi)
#
#     elif bentuk == "persegi panjang":
#         if len(args) != 2:
#             return "Perlu 2 parameter: panjang dan lebar"
#         panjang, lebar = args
#         return luas_persegi_panjang(panjang, lebar)
```

Gambar 3. 4 - File dengan kode yang dijadikan kode comment pada keseluruhan isi file

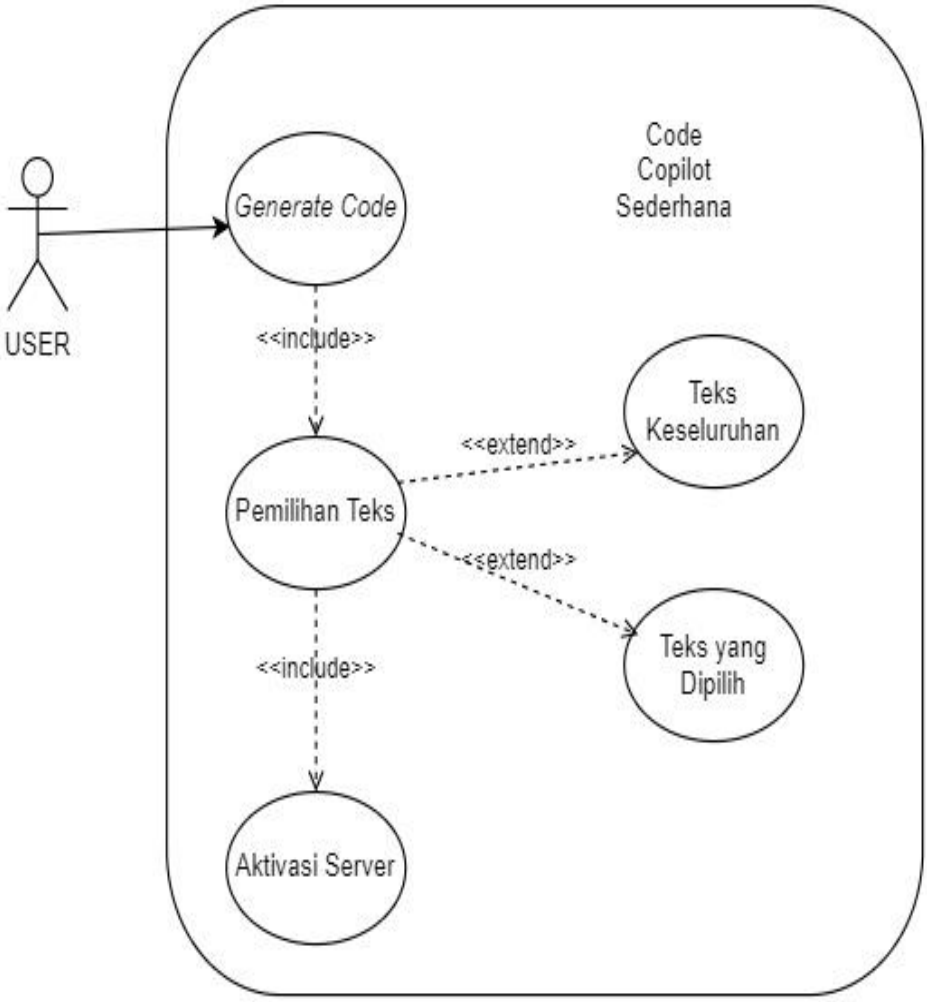
```
if bentuk == "segitiga":
    if len(args) != 2:
        return "Perlu 2 parameter: alas dan tinggi"
    alas, tinggi = args
    return luas_segitiga(alas, tinggi)

elif bentuk == "kotak":
    if len(args) != 1:
```


Gambar 3. 5 - Notifikasi berhasil comment code

Praktikan mencoba untuk membaca kode dari file lalu kode akan dibaca oleh model. Seusai model membaca kode yang diberikan oleh praktikan, model akan memproses kode lalu akan menghasilkan kode lanjutan yang cocok berdasarkan kode sebelumnya.

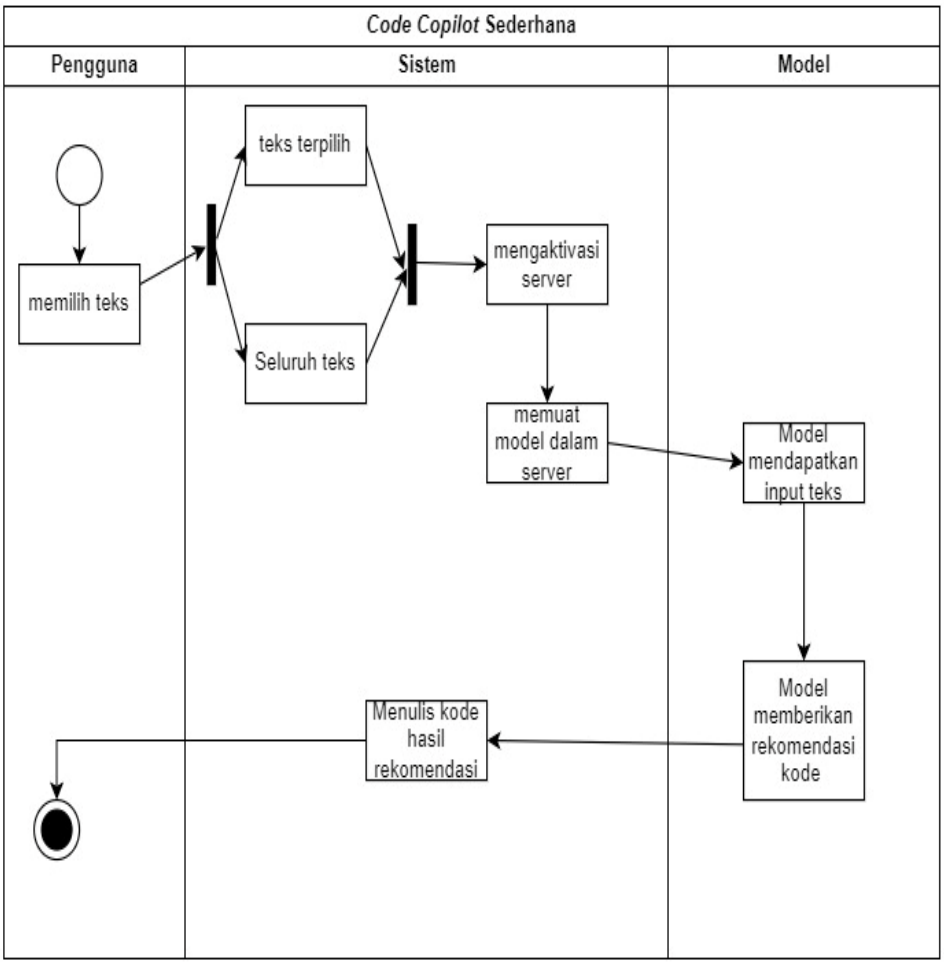
Praktikan merancang *extension* sederhana ini meliputi sistem dapat menangkap kode pada file, lalu kode yang ditangkap akan dibaca dan dijadikan teks. Teks kode ini akan dijadikan sebuah input ke model. Model harus dimuat terlebih dahulu lalu teks input berupa kode ini akan dijadikan sebuah input. Hasil teks kode dari model akan dikirimkan kembali ke *extension* dan dijadikan teks kode pada file.

Praktikan merancang sistem dengan membuat sebuah *Unified Modeling Language* (*UML*). Menurut (Booch et al, 2005), UML adalah bahasa pemodelan standar untuk sistem perangkat lunak yang memungkinkan oleh pengembang untuk dapat memodelkan, merancang, hingga ke tahapan implementasi sistem agar lebih efektif. Praktikan merancang dan memodelkan sebuah sistem dengan *Use Case Diagram* dan *Activity Diagram*. Use Case Diagram ini merepresentasikan visual dari interaksi antar pengguna dan sistem. Diagram ini menjelaskan aktor sebagai pelaku, kasus penggunaan sebagai interaksi antar aktor dan sistem, dan skenario sebagai urutan aksi-aksi yang dilakukan oleh aktor. Lalu, Activity Diagram adalah sebuah representasi visual dari alur kerja sebuah sistem. Activity Diagram terdiri dari aktivitas yaitu langkah langkah yang dilakukan, transisi yaitu hubungan antar aktivitas, dan sebuah keputusan di mana sistem membuat sebuah keputusan.



Gambar 3. 6 - Use Case Diagram pada Extension Sederhana

Praktikan merancang sistem di mana pengguna dapat menghasilkan kode dari *extension*. *User* atau pengguna memerlukan aksi berupa pemilihan teks yang diperlukan untuk dapat menghasilkan kode ke sistem. Pengguna dapat memilih teks yang dipilih maupun seluruh kode dalam suatu file yang dijadikan sebuah teks. Setelah pengguna melakukan aksi pemilihan teks, sistem memerlukan user untuk mengaktivasi server terlebih dahulu.



Gambar 3. 7 - Activity Diagram pada Extension Sederhana

Praktikan merancang alur kerja sebuah sistem pada *extension code copilot* sederhana. Praktikan memulai sebuah alur atau *state* awal dari pengguna yang memilih teks ini dapat dibagi menjadi dua alur pada sistem di mana teks dapat terpilih oleh pengguna maupun seluruh teks ini yang dijadikan input. Selanjutnya, aktivasi server diperlukan untuk dapat memuat model dalam server. Model mendapatkan sebuah masukkan atau input berupa teks sehingga model dapat memberikan rekomendasi kode. Hasil rekomendasi akan diberikan ke sistem dan menuliskan kembali kode hasil rekomendasi ke file tersebut sebagai *state* atau keadaan akhir dari sistem.

3.2.3 Pengembangan

Praktikan memulai untuk mengembangkan hasil rancangan *extension*. Pengembangan *extension* ini meliputi :

a. pembacaan kode,
b. pemuatan model,
c. pembacaan kode dan dijadikan sebuah teks input,
d. aktivasi server,
e. memasukkan teks input ke model,
f. teks kode akan dikirimkan kembali ke server,
g. penulisan teks kode ke *file*.

3.2.3.a Pembacaan Kode

Praktikan mengembangkan *extension* dimulai dari cara *extension* dapat membaca kode dengan gambar potongan kode sebagai berikut :

```
const editor = vscode.window.activeTextEditor;

if (editor) {
    const selection = editor.selection;
    const document = editor.document;
    const codeSnippet = editor.document.getText(selection);
    const text = selection.isEmpty ? document.getText() : codeSnippet;
```

Gambar 3. 8 - Potongan kode untuk membaca kode dari *file*

Extension memerlukan pembacaan file yang sedang digunakan oleh pengguna. Jika terdapat file yang terbaca, maka *extension* dapat membaca kode. Terdapat dua pilihan pengguna *extension* untuk dapat *extension* baca kode, di antaranya adalah kode ini bisa dipilihkan oleh pengguna untuk dibaca dan juga semua teks kode pada file ini dibaca.

3.2.3.b Pemuatan Model

Praktikan mengembangkan *extension* dari sistem dengan lingkungan atau environtment Python dapat memuat model dengan gambar potongan kode sebagai berikut :

```
model_id = "meta-llama/Llama-3.2-1B"

# Load model with specific configurations
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    torch_dtype=torch.bfloat16,
    low_cpu_mem_usage=True,
    device_map="cpu",
    offload_folder="offload",
    token="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"
)

tokenizer = AutoTokenizer.from_pretrained(model_id)
```

Gambar 3. 9 - Potongan kode untuk memuat model ke sistem

Sistem dengan environtment Python ini sebagai penyedia lingkungan untuk dapat memuat model ke sistem. Sistem memerlukan tokenizer untuk menghasilkan output berupa teks kode hasil dari model di mana tokenizer ini sebagai ciri khas output tersebut berasal dari model yang diinginkan.

3.2.3.c Pembacaan kode dan dijadikan sebuah teks input

Praktikan mengembangkan *extension* di mana dapat kode yang dibaca ini dijadikan sebuah teks lalu dijadikan teks input sebagai berikut :

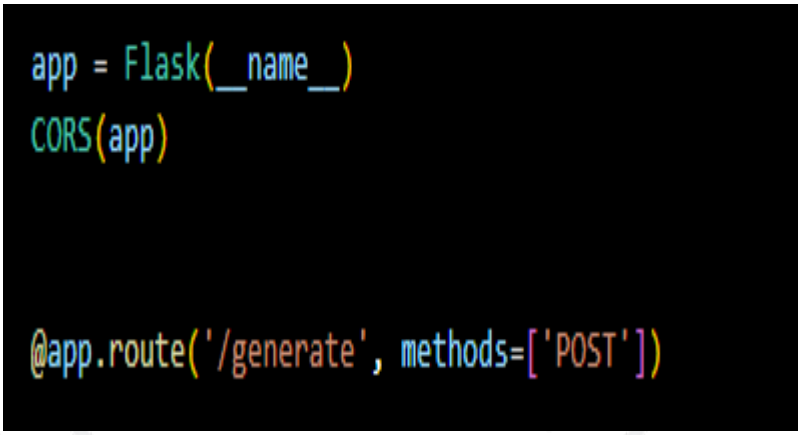
```
const text = selection.isEmpty ? document.getText() : codeSnippet;

try {
    const response = await fetch('http://localhost:5000/generate', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ code: text })
    });
```

Gambar 3. 10 - Potongan kode untuk membaca teks lalu dijadikan teks input ke server

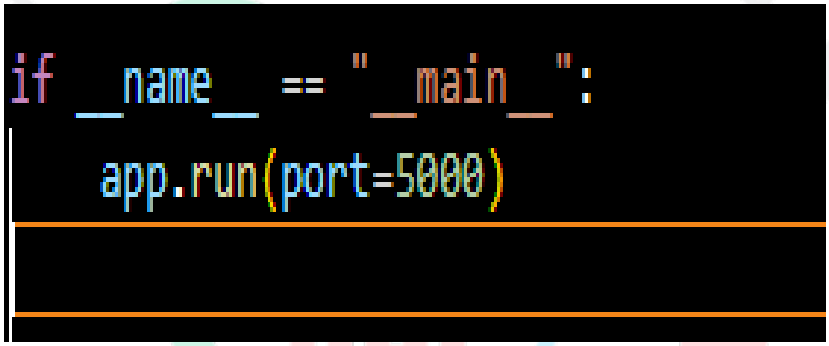
Extension mencoba mendapatkan response dari sebuah tautan yang berasal dari server. Ketika tuatan didapatkan, maka *extension* akan mengirimkan sebuah teks kode untuk dijadikan sebuah file json dengan isi teks berupa kode.

3.2.3.d Aktivasi Server

Praktikan mengembangkan *extension* pada sistem dengan environtment Python dapat menjalankan serta mengaktivasi server Flask. Potongan kode seperti berikut :



Gambar 3. 11 - Potongan kode untuk inisiasi server

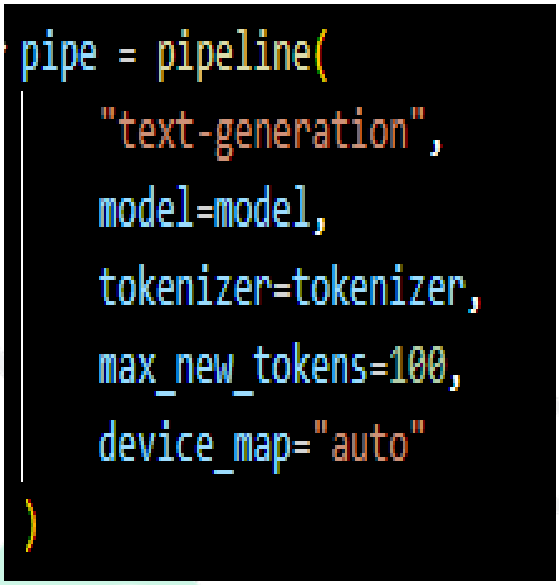


Gambar 3. 12 - Potongan kode untuk mengaktivasi server

Pada server Flask, diperlukan inisiasi server dari Library Flask. Server Flask akan membuat sebuah rute di mana data yang dijadikan input ke server ini pada /generate dengan metode post atau mengirimkan sehingga inputan ini diterima ke server berupa inputan yang dilakukan dengan metode post. Praktikan memerlukan port pada mesin virtual dengan port 5000 sehingga server ini dapat berjalan pada port yang sudah ditentukan.

3.2.3.e Memasukkan Teks Input ke Model

Praktikan mengembangkan *extension* pada sistem dengan environtment Python dapat memasukkan teks input dari server ke model. Model memerlukan sebuah fungsi dari “pipeline” di mana fungsi ini sebagai wadah kode yang diinginkan untuk dimasukkan ke model sekaligus menjadi wadah pada teks output. Potongan kode seperti berikut :

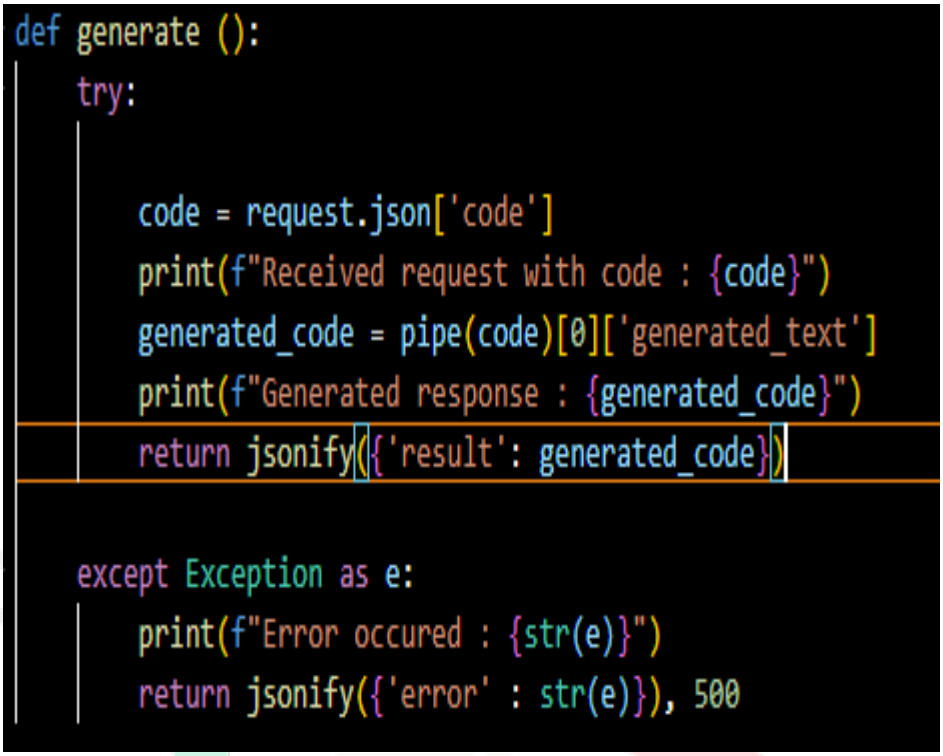


Gambar 3. 13 - Potongan kode untuk membuat sebuah fungsi pipeline

Pada fungsi pipeline, diperlukan sebagai wadah untuk dapat menjadi input, model yang diinginkan sebagai model untuk menghasilkan kode, tokenizer. Fungsi pipeline juga sebagai wadah untuk teks output di mana praktikan membatasi kode yang dihasilkan ini hanya 100 huruf. Fungsi pipeline ini sebagai wadah sehingga memerlukan akses sumber daya perangkat yang memadai untuk dapat menjalankan model pada wadah pipeline.

3.2.3.f Teks Kode Akan Dikirimkan Kembali Ke Server

Praktikan mengembangkan *extension* pada sistem dengan environtment Python dapat menghasilkan kode dari model. Kode yang dihasilkan ini akan dikirimkan kembali ke server. Sistem ini terdefinisi sebagai sebuah fungsi bernama *generate*. Potongan kode seperti berikut :

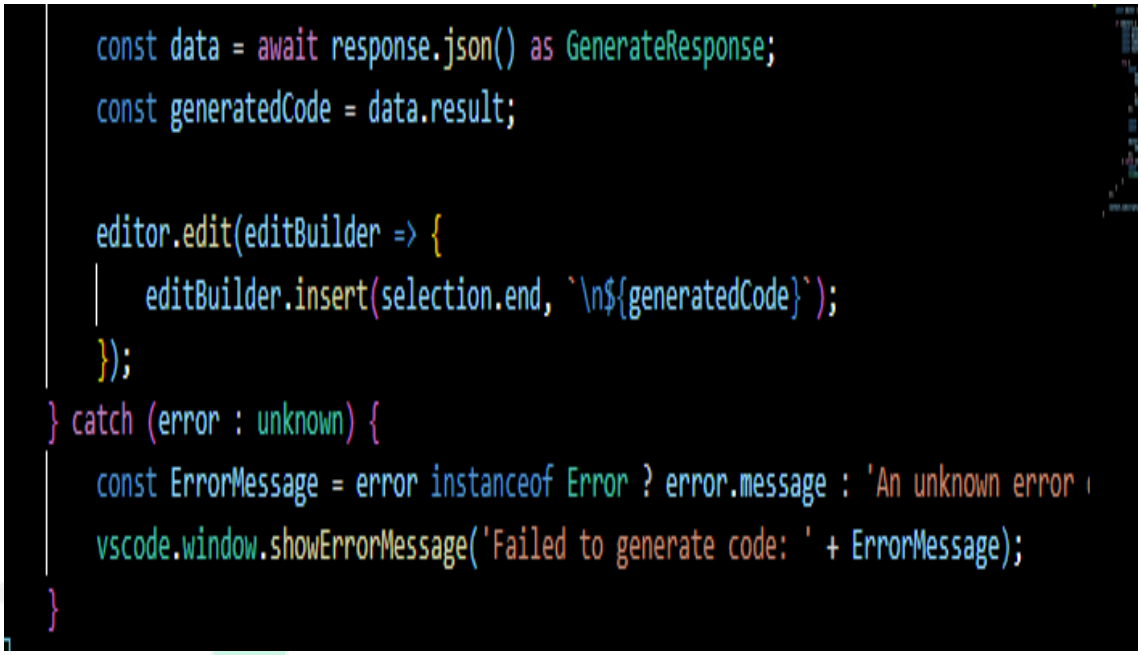


Gambar 3. 14 - Potongan kode untuk membuat sebuah fungsi generate

Pada fungsi generate, sistem akan mencoba untuk mendapatkan kode yang diterima dari server lalu dimasukkan ke wadah. Model dalam wadah akan memproses input. Ketika model selesai dalam menghasilkan output berupa teks kode, wadah akan menyimpan teks kode tersebut. Ketika teks kode tersebut sudah disimpan, maka dikirimkan kembali ke server sebagai file json dengan isi hasil berupa teks kode dari hasilan model.

3.2.3.g Penulisan teks kode ke file

Praktikan mengembangkan *extension* untuk mendapatkan kode dari server lalu dijadikan teks kode pada file tersebut.Potongan kode seperti berikut :

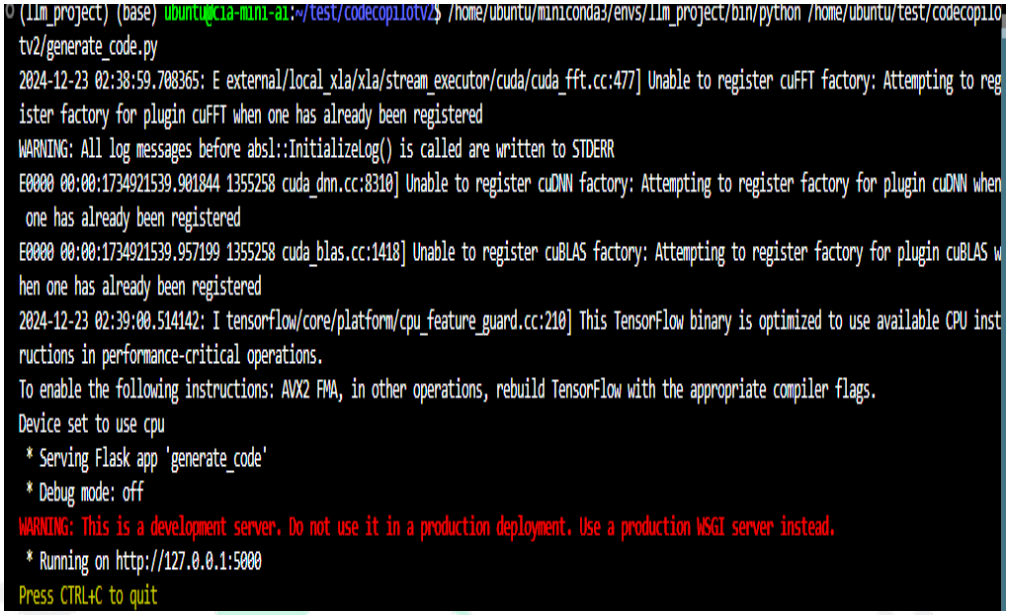


Gambar 3. 15 - Potongan kode untuk membuat sebuah fungsi pipeline

Extension mengambil data berupa teks kode dari file json yang dikirimkan oleh Flask dan disimpan. Ketika mendapatkan teks kode yang disimpan, *extension* akan menuliskin kodenya ke file.

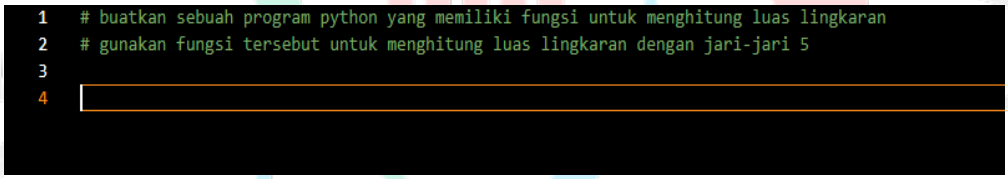
3.2.4 Pengujian

Praktikan melakukan sebuah skenario pengujian sederhana pada extension yang dikembangkan.



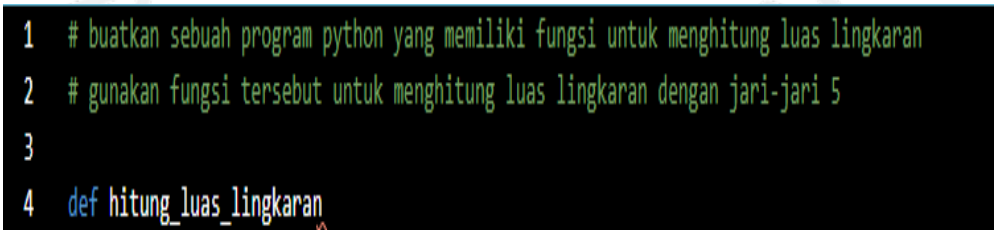
Gambar 3. 16 - Gambar hasil menjalankan server Flask

Praktikan mendapatkan hasil ketika praktikan mencoba untuk menjalankan server Flask. Ketika terdapat link atau tautan dari server, maka server dapat memuat model serta tidak ada masalah dalam server.



Gambar 3. 17 - Gambar Instruksi kepada model

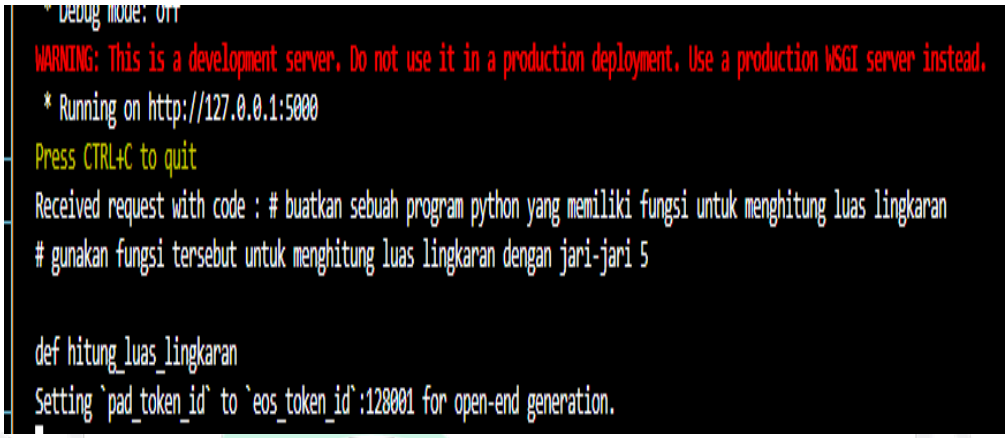
Praktikan mencoba untuk membuat instruksi kepada model sehingga model lebih baik dalam memahami kode yang diinginkan oleh pengguna.



Gambar 3. 18 - Input untuk diuji ke model

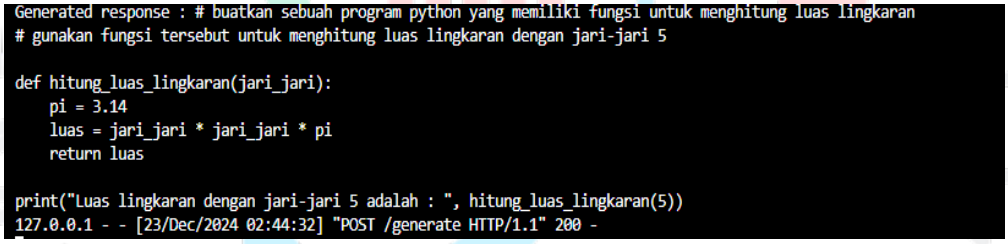
Praktikan mencoba untuk membuat teks kode yang diinginkan untuk dapat dituliskan hasil rekomendasi kode yang diinginkan. Praktikan dengan

sengaja dan sadar untuk menguji pada potongan kode ini tidak lengkap agar model dapat membuat rekomendasi kode sesuai dengan praktik nyata.



Gambar 3. 19 - Mendapatkan input ke server

Praktikan mendapatkan hasil dari server ketika server berhasil mendapatkan teks input dari *extension* . Server mendapatkan teks input berupa instruksi dan juga teks kode yang diinginkan.



Gambar 3. 20 - Mendapatkan hasil rekomendasi dari model

Praktikan mendapatkan hasil rekomendasi kode dari model yang tertulis di server. Terdapat hasil rekomendasi kode tersebut sesuai dengan instruksi.

```python
# buatkan sebuah program python yang memiliki fungsi untuk menghitung luas lingkaran
# gunakan fungsi tersebut untuk menghitung luas lingkaran dengan jari-jari 5

def hitung_luas_lingkaran
# buatkan sebuah program python yang memiliki fungsi untuk menghitung luas lingkaran
# gunakan fungsi tersebut untuk menghitung luas lingkaran dengan jari-jari 5

def hitung_luas_lingkaran(jari_jari):
    pi = 3.14
    luas = jari_jari * jari_jari * pi
    return luas

print("Luas lingkaran dengan jari-jari 5 adalah : ", hitung_luas_lingkaran(5))
```

Gambar 3. 21 - Hasil rekomendasi kode dari model

Praktikan mendapatkan hasil rekomendasi kode dari model yang berhasil dikirimkan dari server ini dituliskan kembali ke file.

Praktikan menyadari bahwa terdapat banyak kekurangan dari *extension* di antaranya adalah :

1. Praktikan masih harus menuliskan instruksi ketika ingin menuliskan kode secara jelas, baru ditambahkan kode yang diinginkan untuk diberikan rekomendasi.
2. Ketika praktikan mendapatkan hasil rekomendasi kode dari model dan dituliskan ke file, hasil rekomendasi yang tertulis ini dituliskan di bawahnya dari baris kode pada file yang diinginkan untuk dibuakan rekomendasi kode.

3.2.5 Pelatihan Model

Praktikan memperoleh hasil yang memuaskan untuk sistem, namun hasil yang diharapkan dari model kurang cukup. Praktikan memutuskan untuk menerapkan pelatihan atau *fine-tuning* model agar dapat lebih sesuai dengan harapan praktikan dalam memberikan sebuah rekomendasi kode.

Pelatihan model ini mencakup pemanggilan tools-tools serta library agar dapat digunakan selama pelatihan berlangsung, pemuatan dataset, awal pemrosesan sebuah dataset, akses model yang diinginkan, konfigurasi yang diperlukan selama pelatihan, pembuatan token pada setiap teks sebagai sebuah ciri khas hasil dari sebuah model, inisiasi argumen yang diperlukan selama pelatihan, pelatihan model.

3.2.5.a Pemanggilan Library yang dibutuhkan

Praktikan memerlukan sebuah tools atau library untuk dapat melatih sebuah model. Tools-tools yang diperlukan sebagai berikut :

```
import torch
from trl import SFTTrainer
from transformers import TrainingArguments
from datasets import Dataset
from unsloth import is_bfloat16_supported

# Saving model
from transformers import AutoTokenizer, AutoModelForSequenceClassification

# Warnings
import warnings
warnings.filterwarnings("ignore")

%matplotlib inline
```

Gambar 3. 22 - tools – tools yang diperlukan selama pelatihan model

Tools-tools tersebut yaitu : torch sebagai library yang menyediakan paket atau package untuk dapat dipraktikan untuk melakukan pelatihan; trl sebagai library yang menyediakan sebuah package untuk melakukan metode pelatihan khusus; transformers sebagai library yang menyediakan package sebagai algoritma dan arsitektur model LLM, datasets sebagai library yang memberikan akses package untuk akses dataset, dan unsloth sebagai salah satu instansi yang aktif dalam mengembangkan LLM sehingga membuat package untuk pengguna praktikan dalam melatih model dalam skala lebih kecil.

3.2.5.b Pemuatan Dataset

```python
from datasets import load_dataset, Dataset

dataset = load_dataset("jtatman/python-code-dataset-500k", split = "train")

dataset = dataset.train_test_split(0.2)
dataset = dataset["test"]
dataset
```

Gambar 3. 23

Praktikan memerlukan package dari datasets untuk dapat mengakses dan mengunduh dataset yang diperlukan. Dataset yang diunduh terlalu besar sehingga praktikan hanya menggunakan sebesar 0,2 dari total dataset.

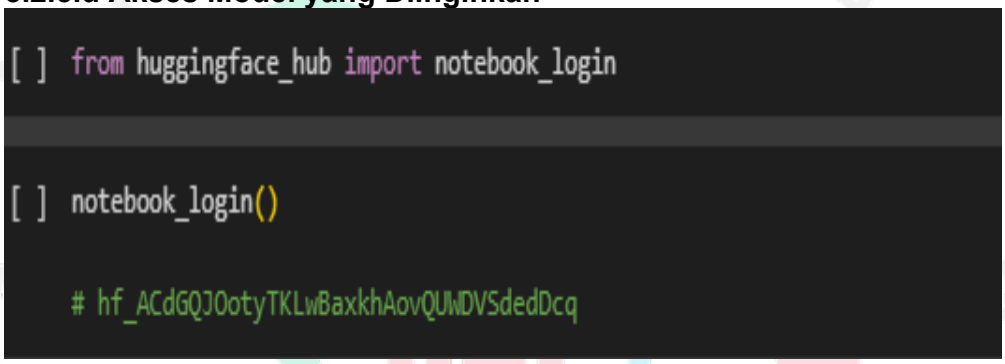
3.2.5.c *Preprocessing Data* dari Dataset

```python
# Preprocessing data
def transform(example):
    output = example['output']
    instruction = example['instruction']
    system_message = example['system']

    def train_prompt_template(output, instruction, system_message):
        template = f"""<|begin_of_text|><|start_header_id|>system<|end_header_id|>{system_message}<|eot_id|>
        <|start_header_id|>user<|end_header_id|>{instruction}<|eot_id|>
        <|start_header_id|>assistant<|end_header_id|>{output}<|eot_id|><|end_of_text|>"""
        return template

    return {"q_lora_data": train_prompt_template(output, instruction, system_message)}

# Menggunakan fungsi transform pada dataset
new_dataset = dataset.map(transform, batched=False)
new_dataset = new_dataset.remove_columns(['output', 'instruction', 'system'])

# Split dataset menjadi train dan test
new_dataset = new_dataset.train_test_split(test_size=0.1)

# Rename kolom untuk qLoRA
new_dataset = new_dataset.rename_columns({"q_lora_data": "text"})
```

Gambar 3. 24

Praktikan memerlukan transformasi pada setiap baris dalam dataset agar dapat diolah datanya sesuai dengan peraturan dari input dalam pelatihan model LLM. Data perlu ditransformasi untuk dapat dijadikan sebagai data pengolahan yang dapat diterima oleh model di mana model dapat menerima masukkan, instruksi, dan output yang cocok berdasarkan template. Karena ukuran datasetnya masih tergolong besar namun dengan sumber daya terbatas, praktikan masih memerlukan untuk menggunakan sebesar 0,1 dari dataset sehingga dataset baru menjadi lebih kecil ukurannya.

3.2.5.d Akses Model yang Diinginkan



Gambar 3. 25

Praktikan memerlukan login ke portal bernama *huggingface* agar dapat mengakses model yang diinginkan oleh praktikan dalam pelatihan. Praktikan memasukan token yang diperlukan sebagai kredensial atau autentikasi pengguna.

3.2.5.e Konfigurasi yang Diperlukan selama Pelatihan

```
from trl import SFTTrainer
from peft import LoraConfig, prepare_model_for_kbit_training, get_peft_model
from transformers import AutoTokenizer,
AutoModelForCausalLM,
BitsAndBytesConfig,
TrainingArguments,
Trainer,
DataCollatorForLanguageModeling

model_id = "meta-llama/Llama-3.2-1B-Instruct"

# this setup the quantization for QLoRA
quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_quant_type="nf4"
)

# load the tokenizer and model
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(model_id,
    device_map='auto',
    torch_dtype=torch.bfloat16,
    quantization_config=quantization_config)
```

Gambar 3. 26

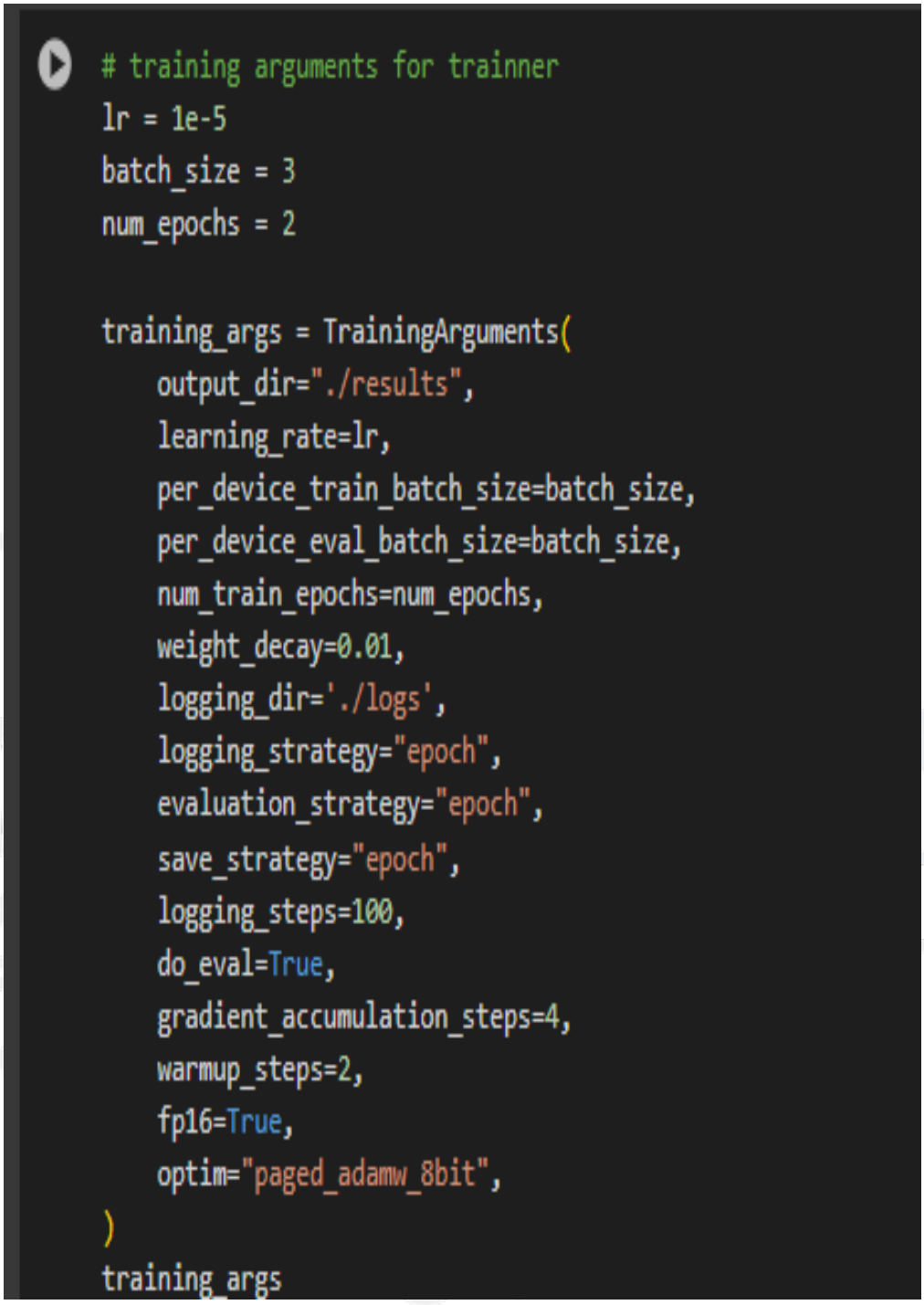
Praktikan mencoba untuk mengkonfigurasi tools-tools maupun package yang tersedia ini cocok dengan sumber daya komputasi yang dimiliki oleh praktikan. Konfigurasi ini penting agar selama proses pelatihan ini tidak memakan waktu yang cukup lama namun tetap efisien yang disebabkan dari minimnya sumber daya. Praktikan memulai dengan menginisiasi nama model yang diinginkan lalu dikonfigurasikan dengan menyatakan nama model yang diinginkan untuk dilatih.

3.2.5.f Pembuatan Token pada Setiap Teks

```
# create tokenize function
def tokenize_function(examples):
    # extract text
    text = examples["text"]

    # tokenize and truncate text
    tokenizer.truncation_side = "left"
    tokenized_inputs = tokenizer(
        text,
        return_tensors="np",
        truncation=True,
        max_length=512
    )

    return tokenized_inputs

# tokenize training and validation datasets
tokenized_data_train = new_dataset['train'].map(tokenize_function, batched=True)
tokenized_data_test = new_dataset['test'].map(tokenize_function, batched=True)
```

Praktikan memerlukan sebuah fungsi untuk melakukan tokenisasi pada setiap baris dalam dataset. Ini berguna agar model dapat memberikan hasil yang sesuai berdasarkan peraturan dari arsitektur pengembangan model.

3.2.5.g Inisiasi Argumen yang Diperlukan Selama Pelatihan

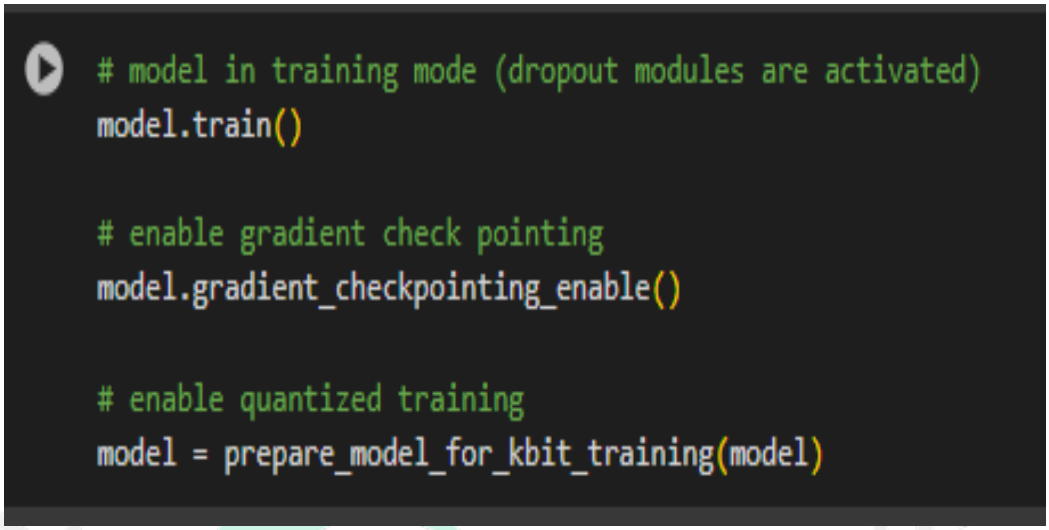


```
# training arguments for trainner
lr = 1e-5
batch_size = 3
num_epochs = 2

training_args = TrainingArguments(
    output_dir="./results",
    learning_rate=lr,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=num_epochs,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_strategy="epoch",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    logging_steps=100,
    do_eval=True,
    gradient_accumulation_steps=4,
    warmup_steps=2,
    fp16=True,
    optim="paged_adamw_8bit",
)
training_args
```

Gambar 3. 27

Praktikan menyatakan *argument-argument* yang berlaku selama pelatihan berlangsung. Argumen ini diperlukan sehingga pelatihan model ini bisa sesuai dengan sumber daya komputasi yang dimiliki serta hasil yang memuaskan meskipun dengan metode yang efisien selama pelatihan.

3.2.5.h Proses Pelatihan Model



Gambar 3. 28

Praktikan melakukan pelatihan pada model dengan berbagai metode seperti *train() , gradient_checkpointing_enable().* Praktikan menyatakan model yang baru sebagai model yang sudah dilatih sebelumnya sehingga bisa digunakan untuk merekomendasi kode pada *extension .*

Meskipun praktikan sudah mempraktikan cara *fine-tuning* atau pelatihan hingga penyesuaian model, praktikan belum mendapatkan hasil yang memuaskan karena sumber daya yang terbatas dan minimnya waktu. Praktikan tidak melampirkan

3.3 Kendala yang Dihadapi

Saat menjalani Kerja Profesi di PT Datacomm Diangraha, praktikan menghadapi beberapa permasalahan dalam proyek yang dikerjakan. Terdapat beberapa faktor yang menyebabkan praktikan mengalami kendala selama melakukan Kerja Profesi.

a. Praktikan belum pernah mempelajari bidang LLM dalam Kecerdasan Buatan atau *Artificial Intelligence.*
b. Kesibukan mentor yang terlibat dalam proyek perusahaan, sehingga menghambat komunikasi dan mengurangi efektivitas dalam pemanfaatan waktu.

c. Praktikan belum selesai dalam melakukan pelatihan model yang disebabkan oleh keterbatasan sumber daya yang diberikan oleh pihak perusahaan.
d. Akses internet sering terkendala disebabkan oleh gangguan eksternal seperti kabel jaringan putus sehingga menyebabkan *VM* sebagai sarana praktikan bekerja tidak dapat digunakan.

3.4 Cara Mengatasi Kendala

Praktikan menghadapi dan mengalami kendala selama Kerja Profesi, namun sebaik mungkin untuk mengatasi kendala tersebut. Berikut adalah cara praktikan dalam mengatasi kendala yang dihadapi saat Kerja Profesi di PT Datacomm Diangraha :

a. Praktikan melakukan eksplorasi mandiri pada bidang LLM di portal seperti huggingface. Praktikan mempraktikan LLM sederhana sesuai dengan arahan mentor.
b. Praktikan memberanikan diri dalam bertanya dan berkomunikasi dengan pembimbing di perusahaan.
c. Praktikan mencoba berbagai hal baru dan menghadapi banyak kendala sehingga ketika mentor dapat membimbing berbagai kendala yang praktikan hadapi sekaligus.
d. Praktikan mencoba mempelajari konsep yang diperlukan jika terjadi gangguan pada internet yang mengakibatkan *VM* sarana praktikan bekerja tidak dapat diakses.

3.5 Pembelajaran Yang Diperoleh dari Kerja Profesi

Selama melakukan program Kerja Profesi atau magang di PT Datacomm Diangraha, praktikan memperoleh beberapa pembelajaran.

a. Praktikan belajar mengenai *soft skill*, termasuk tanggung jawab, disiplin waktu, kemampuan komunikasi, dan cara penyelesaian masalah. *Soft skill* ini tentu sangat penting di dunia kerja.

b. Praktikan belajar mengenai pengembangan kecerdasan buatan generatif atau *Generative AI* seperti *LLM* yang dapat menghasilkan teks.
c. Praktikan mempelajari bahasa pemrograman Typescript lebih dalam sebagai bahasa yang lebih kompleks dan kaku dari bahasa Javascript.
d. Praktikan mempelajari cara *tuning LLM* atau melatih ulang sehingga *LLM* dapat mempelajari data-data atau informasi yang lebih berguna sehingga model *LLM* dapat memberikan informasi yang lebih relevan dengan aplikasi yang diinginkan.
e. Praktikan mempelajari berbagai *tools* lainnya atau *package* atau *library* yang berguna dalam melaksanakan kerja profesi maupun di dunia kerja kedepannya.