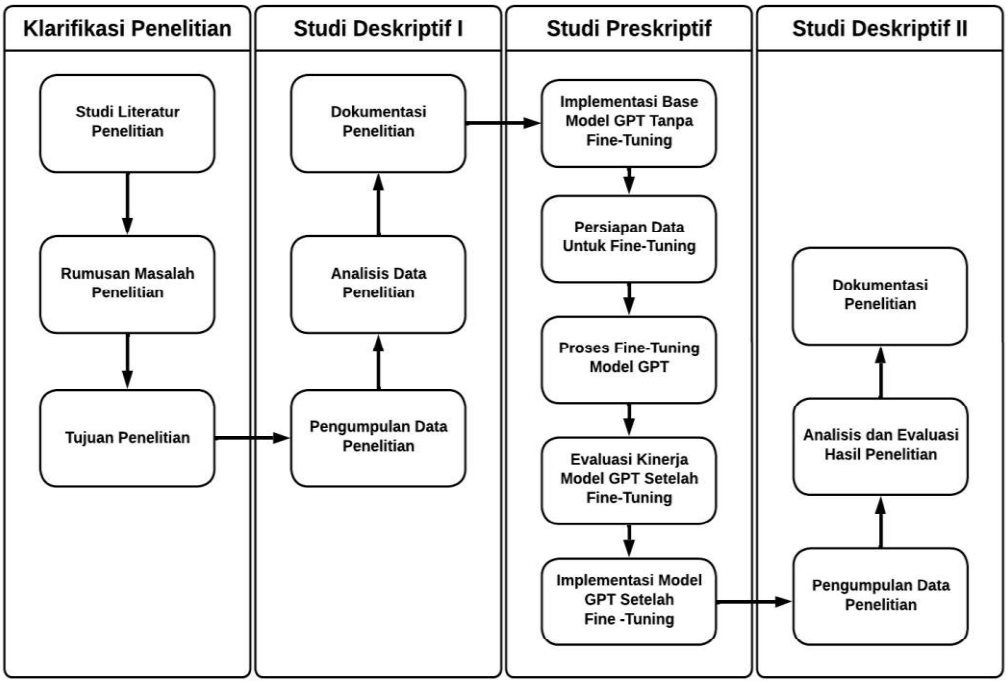
BAB III
METODE PENELITIAN

3.1 Desain Penelitian

Penelitian ini menggunakan desain penelitian *Design Research Methodology* (DRM). *Design Research Methodology* (DRM) merupakan pendekatan yang berisi metode yang digunakan sebagai kerangka kerja dalam melakukan desain penelitian (Calderon, M. L, 2010). Menurut Blessing dan Charabakti, Penelitian dengan menggunakan *Design Research Methodology* (DRM) dapat mendukung pendekatan yang lebih terstruktur dengan membantu merencanakan dan melaksanakan desain penelitian. Berikut pada merupakan desain penelitian ini.



Gambar 3.1 Design Research Methodology (DRM)

Pada Gambar 3.1 merupakan desain penelitian ini yang menggunakan *Design Research Methodology* (DRM) dengan empat tahapan utama yaitu Klasifikasi Penelitian, Studi Deskriptif I, Studi Preskriptif, dan Studi Deskriptif II yang mana setiap tahapan memiliki peran dan tujuan yang spesifik dalam mendukung proses penelitian.

3.1.1 Klarifikasi Penelitian

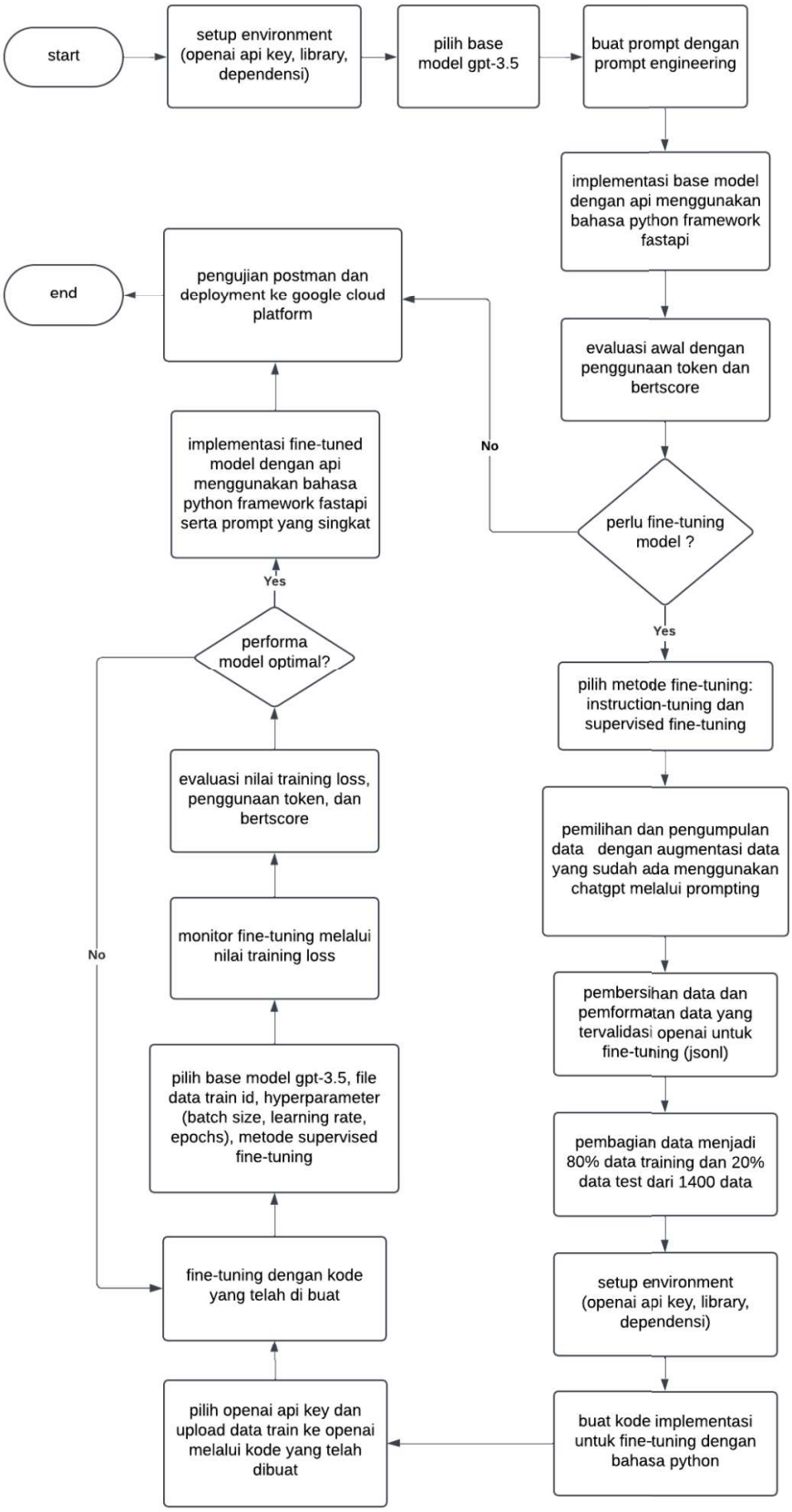
Pada tahap Klarifikasi Penelitian dilakukan beberapa hal penting untuk memastikan tujuan dan fokus penelitian yang jelas dan terstruktur (Blessing et al., 2009). Tahapan ini mencakup studi literatur untuk mengidentifikasi dan memahami penelitian sebelumnya dengan mengumpulkan data awal dari sumber-sumber yang relevan dengan topik penelitian seperti jurnal, buku, dan artikel tentang implementasi *fine-tuning* dan *base* model GPT-3.5 dalam menghasilkan intervensi afektif pada pembelajaran daring sinkronis. Selanjutnya, perumusan masalah penelitian yang dilakukan berdasarkan hasil studi literatur. Rumusan masalah penelitian dikembangkan untuk memberikan fokus yang spesifik pada tujuan penelitian. Terakhir, menetapkan tujuan penelitian yang jelas dan terukur berdasarkan hasil studi literatur dan rumusan masalah.

3.1.2 Studi Deskriptif I

Pada tahap Studi Deskriptif dilakukan pengumpulan data awal melalui observasi untuk memahami data deskriptif terkait implementasi *fine-tuning* dan *base* model GPT-3.5 dalam menghasilkan intervensi afektif pada pembelajaran daring sinkronis sehingga memberikan gambaran menyeluruh tentang masalah yang diteliti. Data yang terkumpul dianalisis untuk mengidentifikasi pola, tantangan, dan kebutuhan yang ada sebelum solusi dirancang. Data yang sudah dianalisis didokumentasikan secara detail. Tahapan ini bertujuan untuk memberikan dasar pengembangan solusi yang relevan dan efektif serta memastikan penelitian ini didasarkan pada pemahaman yang komprehensif tentang masalah yang dihadapi (Blessing et al., 2009).

3.1.3 Studi Preskriptif

Pada tahap Studi Preskriptif dilakukan pengembangan solusi berdasarkan hasil dari Studi Deskriptif I (Blessing et al., 2009).

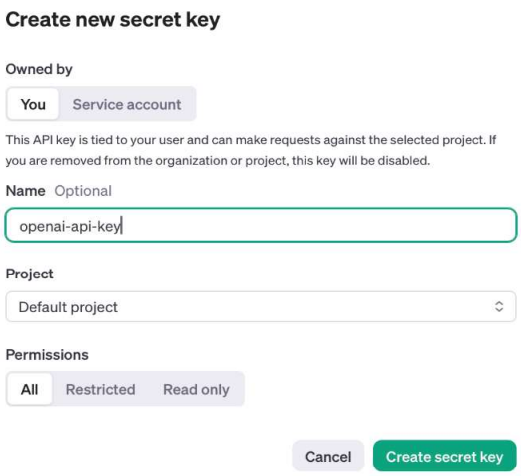


Gambar 3.2 Diagram Flowchart Pengembangan Model

Dalam konteks penelitian ini, pengembangan solusi menggunakan model Generative Pre-trained Transformer (GPT) dari OpenAI yang melibatkan serangkaian tahapan sistematis sesuai dengan dokumentasi resmi dari OpenAI. Berikut merupakan tahapan pengembangan model dalam penelitian ini yang diilustrasikan melalui diagram flowchart pada Gambar 3.2 yang terdiri dari detail tahapan Implementasi Base Model GPT tanpa Fine-Tuning, Persiapan Data untuk Fine-Tuning, Proses Fine-Tuning Model GPT, Evaluasi Kinerja Model GPT setelah Fine-Tuning, dan Implementasi Model GPT setelah Fine-Tuning.

3.1.3.1 Implementasi Base Model GPT Tanpa Fine-Tuning

Dalam penelitian ini, tahapan pertama yang dilakukan adalah implementasi *base* model GPT-3.5 dari OpenAI dengan mengikuti tahapan yang disediakan oleh OpenAI sehingga memahami kapasitas *base* model sebelum tahap *fine-tuning*. Berdasarkan dokumentasi resmi yang disediakan oleh OpenAI, tahapan ini diawali dengan membuat OpenAI API key melalui situs resmi dari OpenAI, seperti terlihat pada Gambar 3.3. API key ini berfungsi sebagai alat autentikasi dan otorisasi yang memungkinkan pengguna mengakses *base* model GPT serta mengelola penggunaan sumber daya sesuai dengan kebijakan keamanan dan kuota yang ditetapkan oleh OpenAI.



Gambar 3.3 Membuat OpenAI API Key

Langkah berikutnya adalah mengimplementasikan model melalu*i Application Programming Interface* (API) yang memungkinkan model untuk menerima, memproses, dan merespons permintaan dari sistem atau layanan eksternal lain. Implementasi pengembangan API menggunakan bahasa pemrograman Python dengan *framework* FastAPI yang sesuai untuk membuat API yang cepat dan efisien dengan beberapa tahapan pengembangan. Selanjutnya, untuk pemilihan model GPT-3.5 didasarkan pada hasil penelitian sebelumnya yang dilakukan oleh Meyer et al. (2024) dan Hajipoor et al. (2025) yang menunjukkan bahwa model ini memiliki kemampuan dalam menghasilkan teks berkualitas serta beragam terutama dalam konteks spesifik seperti pendidikan dan psikologi. Penelitian-penelitian tersebut membuktikan bahwa GPT-3.5 mampu memahami berbagai konteks dengan baik sehingga memberikan respons yang relevan.

Gambar 3.4 Library dan Konfigurasi

Dalam pengembangan API ini, penggunaan beberapa *library* Python diperlukan untuk menjamin fungsionalitas dan keamanan sistem, seperti yang terlihat pada Gambar 3.4. Pertama, modul os digunakan untuk mengakses fungsi-fungsi sistem operasi khususnya untuk interaksi dengan variabel lingkungan. Library dotenv melalui fungsi load_dotenv diimplementasikan untuk memuat konfigurasi dari file .env yang tidak termasuk dalam kode sumber untuk meningkatkan keamanan sistem dengan menyembunyikan data sensitif. Uvicorn sebagai server *Asynchronous Server Gateway Interface* (ASGI) digunakan untuk menjalankan sistem FastAPI dengan efisiensi tinggi

sehingga mendukung permintaan asinkron. Modul openai digunakan untuk memfasilitasi interaksi langsung dengan API OpenAI sehingga dapat menggunakan model baik yang telah di *fine-tuning* maupun *base* model. Terakhir, FastAPI dipilih sebagai *framework* utama untuk mengembangkan API karena kecepatan dan skalabilitas sedangkan HTTPException memungkinkan penanganan kesalahan HTTP yang lebih efektif.

app > fine tuning > openai_api.py

```
7   load_dotenv()
8   openai.api_key = os.getenv("OPENAI_API_KEY")
9   openai_model = os.getenv("OPENAI_MODEL")
10
11  required_vars = [openai.api_key, openai_model]
12  missing_vars = [var for var in required_vars if var is None]
13
14  if missing_vars:
15      print(f"missing the following variables in .env file: {', '.join(missing_vars)}")
16      exit()
```

Gambar 3.5 Pemeriksaan dan Pengelolaan Variabel Lingkungan

Setelah memuat *library* yang diperlukan, pada Gambar 3.5 konfigurasi sistem dimulai dengan fungsi load_dotenv() yang memungkinkan pengambilan konfigurasi, seperti OPENAI_API_KEY dan OPENAI_MODEL dari file .env. Hal ini dilakukan untuk memastikan bahwa semua informasi sensitif yang diperlukan tersimpan aman dan tidak terpapar dalam kode. Variabel required_vars menyimpan variabel penting yang dibutuhkan sistem untuk beroperasi. Pengecekan dilakukan untuk memastikan tidak ada variabel yang hilang. Jika ada variabel yang tidak terdefinisi, sistem akan memberikan pemberitahuan kesalahan dan menghentikan eksekusi untuk menghindari operasi tanpa konfigurasi yang memadai.

app > fine tuning > openai_api.py

```
18  async def read_prompt_file():
19      try:
20          with open('prompt.txt', 'r') as file:
21              data = file.read()
22          return data
23      except Exception as e:
24          raise Exception(f"error reading prompt file: {str(e)}")
```

Gambar 3.6 Fungsi untuk Membaca Data dari File Prompt

Pada Gambar 3.6 fungsi read_prompt_file digunakan untuk membaca isi file prompt.txt. Fungsi ini beroperasi secara asinkron dengan mencoba membuka file dalam mode baca ('r'), membaca seluruh isinya, dan mengembalikannya sebagai string. Pendekatan ini memungkinkan pemisahan antara kode utama dan isi prompt sehingga lebih fleksibel dalam pengelolaan dan pembaruan tanpa perlu mengubah kode sumber.

app > utils > file > prompt.txt

```
Instruksi
- Berikan intervensi afektif berupa kalimat motivasi untuk meningkatkan kesejahteraan dan performa pelajar
  selama pembelajaran online di Google Meet berdasarkan informasi input dan output.
- Jangan berasumsi apapun terkait sumber emosinya.
- Informasi tambahan:
  - ARCS Model
    - Attention: Gunakan strategi untuk menarik perhatian, seperti kejutan, humor, atau aktivitas yang
                 membangkitkan rasa ingin tahu agar mereka fokus dan tertarik.
    - Relevance: Hubungkan pembelajaran dengan kebutuhan atau tujuan agar mereka merasa materi tersebut
                 penting dan berguna.
    - Confidence: Bangun kepercayaan diri dengan memastikan mereka merasakan kemajuan dan yakin dapat
                  berhasil.
    - Satisfaction: Ciptakan rasa puas dengan memberikan umpan balik positif atau penghargaan atas
                    pencapaian mereka agar termotivasi untuk terus belajar.
  - Reinforcement
    - Positive: Berikan sesuatu yang menyenangkan setelah perilaku positif untuk memperkuat perilaku tersebut.
    - Negative: Hilangkan hal yang tidak menyenangkan setelah perilaku positif terjadi untuk mendorong perilaku
                tersebut terulang di masa depan.

Format Input
- Emotion: emotion
- Reinforcement: reinforcement
- ARCSModel: arcsmodel
- Username: username
- Classname: classname
- Subject: subject

Format Output
- Bahasa Indonesia yang tidak formal.
- Sertakan emoji.
- Maksimal 25 kata.
```

Gambar 3.7 File Prompt dengan Prompt Engineering

Selanjutnya Gambar 3.7 menunjukkan isi dari file prompt.txt, yang dirancang menggunakan teknik *prompt engineering*. File ini berisi instruksi spesifik untuk menghasilkan intervensi afektif dengan mempertimbangkan model ARCS (*Attention, Relevance, Confidence, Satisfaction*) serta strategi *Reinforcement Positive atau Negative*. Selain itu, format input dan output telah ditentukan, seperti penggunaan bahasa informal, penyertaan emoji, dan batasan maksimal 25 kata, untuk memastikan bahwa respons yang dihasilkan sesuai dengan kebutuhan.

app > fine tuning > openai_api.py

```python
async def get_openai_response(username, emotion, reinforcement, arcsmodel, classname, subject, prompt):
    system_content = prompt
    user_content = (
        f"Username: {username}\n"
        f"Emotion: {emotion}\n"
        f"Reinforcement: {reinforcement}\n"
        f"ARCSModel: {arcsmodel}\n"
        f"Classname: {classname}\n"
        f"Subject: {subject}"
    )

    try:
        response = await openai.ChatCompletion.create(
            model=openai_model,
            messages=[
                {"role": "system", "content": system_content},
                {"role": "user", "content": user_content}
            ],
            max_tokens=150,
            temperature=0.7,
        )
        return response.choices[0].message.content.strip()

    except Exception as e:
        raise Exception(f"error generating response from OpenAI: {str(e)}")
```

Gambar 3.8 Fungsi untuk Mengambil Respons dari OpenAI

Proses berikutnya terlihat pada Gambar 3.8 melibatkan fungsi get_openai_response untuk menghasilkan respons yang dipersonalisasi. Fungsi ini mengambil beberapa parameter, termasuk username, emotion, reinforcement, arcs model, classname, subject, dan prompt yang diintegrasikan ke dalam user_content dan system_content. user_content merupakan content dari role user merepresentasikan pengguna yang berinteraksi dengan model dengan memberikan input berupa instruksi atau konteks tertentu, sedangkan system_content merupakan content dari role system sebagai pengatur perilaku model dalam memberikan respons dengan memberikan instruksi atau panduan awal kepada model agar mengikuti gaya atau aturan tertentu. Model yang digunakan pada implementasi ini adalah Generative Pre-trained Transformer 3.5 (GPT-3.5). max_token adalah menentukan jumlah token maksimal yang dapat dihasilkan dalam respons dari model maka respons yang dihasilkan tidak akan melebihi 150 token. Sementara itu, temperature digunakan untuk mengatur tingkat kreativitas dan variasi dalam respons yang dihasilkan oleh model. Nilai temperature biasanya berkisar antara 0 hingga 1. Pada implementasi ini digunakan temperature dengan nilai 0.7 memungkinkan model untuk menghasilkan respons yang lebih kreatif dan beragam.

Terakhir dalam tahapan pengembangan, implementasi menggunakan FastAPI untuk interaksi dengan OpenAI seperti pada Gambar 3.8 menunjukkan bagaimana endpoint API dirancang untuk menangani permintaan. Endpoint tersebut didefinisikan dengan path /openai menggunakan @app.get("/openai") dan fungsi openai_response bertanggung jawab untuk memproses parameter masukan seperti username, emotion, reinforcement, arcsmodel, classname, subject, dan prompt.

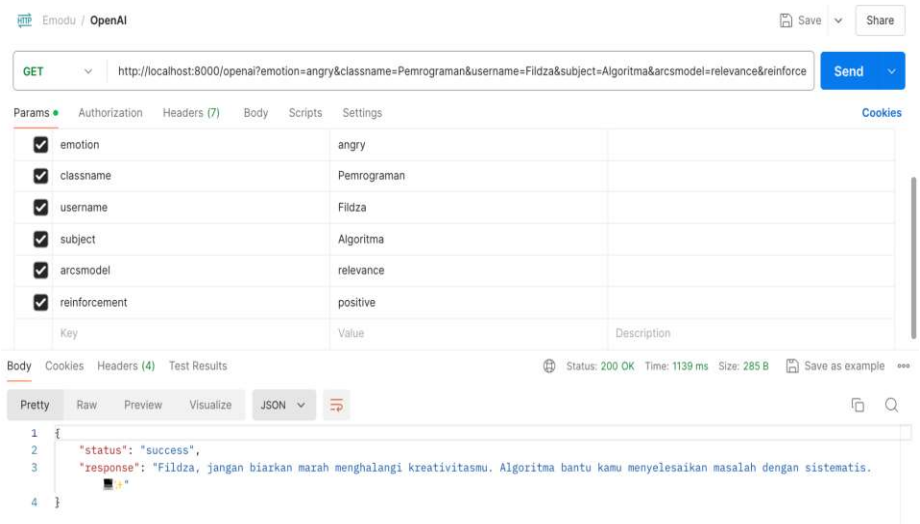
app > fine tuning > openai_api.py

```python
app = FastAPI()

@app.get("/openai")
async def openai_response(
    username: str,
    emotion: str,
    reinforcement: str,
    arcsmodel: str,
    classname: str,
    subject: str
):
    try:
        prompt = await read_prompt_file()
        response = await get_openai_response(
            username=username,
            emotion=emotion,
            reinforcement=reinforcement,
            arcsmodel=arcsmodel,
            classname=classname,
            subject=subject,
            prompt=prompt
        )

        return {
            "status": "success",
            "response": response
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Gambar 3.9 Implementasi API dengan FastAPI untuk Interaksi OpenAI

Parameter ini kemudian digunakan untuk membuat permintaan ke layanan OpenAI melalui fungsi get_openai_response. Proses diawali dengan membaca konten file prompt melalui fungsi read_prompt_file yang menyediakan dasar bagi konteks sistem. Fungsi

get_openai_response memanfaatkan data parameter yang diterima untuk mengirimkan masukan ke layanan OpenAI sehingga menghasilkan respons yang relevan berdasarkan konteks yang diberikan. Jika proses berhasil maka API akan mengembalikan hasilnya dalam format JSON dengan status *success* dan respons yang dihasilkan.



Gambar 3.10 Testing API di Postman

Setelah pengembangan selesai, dilakukan pengujian API menggunakan Postman, seperti pada Gambar 3.10 untuk memastikan fungsionalitas berjalan sesuai yang diharapkan. Pengujian ini meliputi simulasi permintaan dari klien untuk memverifikasi bagaimana API menangani berbagai skenario input dan memastikan output yang dihasilkan sesuai dengan harapan. Pengujian ini meliputi permintaan dari klien dengan memasukkan query parameter, seperti emotion, classname, username, subject, arcsmodel, dan reinforcement. Permintaan dikirimkan melalui metode GET ke endpoint http://localhost:8000/openai dengan parameter yang sesuai. Setelah permintaan diproses oleh API, respons dikembalikan dalam format JSON dengan status 200 OK yang menunjukkan bahwa permintaan berhasil diproses.

Tujuan dari tahap ini adalah untuk memahami kemampuan dasar dari model GPT yang mana pada penelitian ini menggunakan model

dasar GPT-3.5 sebelum dilakukan fine-tuning lebih lanjut. Evaluasi awal menggunakan evaluasi efisiensi penggunaan token dan relevansi output dengan BERTScore. Data penggunaan token diperoleh melalui proses logging pada 280 interaksi dengan base model GPT-3.5. Terdapat tiga jenis token yang mana prompt_tokens adalah jumlah token yang digunakan untuk menyusun prompt yang dikirim ke model, completion_tokens adalah jumlah token yang dihasilkan oleh model sebagai respons terhadap prompt, sedangkan total_tokens adalah jumlah gabungan dari prompt_tokens dan completion_tokens. Sementara itu, data BERTScore terdiri dari metrik evaluasi *Precision*, *Recall*, dan *F1-Score* yang didapatkan melalui evaluasi dengan *library* BERTScore. Langkah selanjutnya adalah merencanakan penyesuaian lanjutan yang mencakup rencana untuk *fine-tuning* model dengan dataset yang lebih spesifik. Dengan demikian, tahap implementasi model GPT dasar tanpa *fine-tuning* memberikan dasar yang kuat untuk pengembangan lebih lanjut, memungkinkan penelitian untuk mengidentifikasi potensi dan keterbatasan model GPT dasar sebelum melangkah ke tahapan optimisasi yang lebih kompleks.

3.1.3.2 Persiapan Data Untuk Fine-Tuning

Setelah mengidentifikasi potensi dan keterbatasan dari *base* model GPT tanpa *fine-tuning*, tahapan selanjutnya adalah mempersiapkan data untuk *fine-tuning*. Persiapan data merupakan langkah krusial karena menggunakan metode instruction tuning dan supervised fine-tuning sehingga dataset harus dapat disesuaikan terhadap kebutuhan khusus pada permasalahan penelitian ini. Proses seleksi dan pengumpulan data dilakukan dengan teknik augmentasi data teks menggunakan ChatGPT dengan model o1 dan o1-mini melalui strategi prompting yang didukung kajian literatur terkait konteks data. Pendekatan ini dilakukan untuk memastikan bahwa augmentasi data dilakukan secara sistematis dan relevan, sehingga data yang dihasilkan memiliki kualitas yang sesuai dengan tujuan penelitian. Teknik augmentasi data ini peningkatan volume dan keragaman data.

Augmentasi Data Teks mencakup metode seperti *paraphrasing*, *synonym replacement*, *random insertion*, *random swap*, dan *back translation* dari dataset yang tersedia dari penelitian Fatharani (2024) untuk dijadikan dasar augmentasi yang memungkinkan penciptaan variasi teks yang lebih beragam. Dataset yang tersedia berjumlah 160 data yang terverifikasi oleh ahli di bidang Psikologi dan Pendidikan.

Gambar 3.11 Format JavaScript Object Notation Lines

Setelah proses seleksi dan pengumpulan data, proses pembersihan data dilakukan untuk menghapus elemen yang tidak relevan dengan melakukan normalisasi teks, dan menyesuaikan format data yang divalidasi oleh OpenAI untuk fine-tuning model. Format data yang divalidasi dalam format JavaScript Object Notation Lines (JSONL) , seperti pada Gambar 3.11 yang memungkinkan setiap baris file berisi satu objek JSON yang mewakili satu messages atau pesan dalam dialog, seperti {"messages": [{"role": "system", "content": "..."}, {"role": "user", "content": "..."}, {"role": "assistant", "content": "..."}]}. Dalam setiap objek JSON terdapat *role* yang menandai peran pesan, seperti *system*, *user*, atau *assistant*. Dalam penelitian ini, *role system* menyediakan instruksi spesifik mengenai *content* yang harus dihasilkan oleh *assistant*, yaitu intervensi afektif dengan batasan tertentu.

Selanjutnya, *role user* memberikan informasi spesifik tentang situasi pelajar, termasuk jenis emosi, jenis *reinforcement*, jenis model ARCS, nama kelas, nama pelajaran, dan nama pelajar yang dapat

dijadikan label untuk metode *Supervised-Fine-Tuning*. Terakhir *role assistant* menghasilkan respons berdasarkan instruksi dari *role system* dan informasi dari *role user*. Sementara itu, *content* adalah isi teks pesan dari setiap *role*. Hal tersebut krusial untuk fine-tuning dengan metode *Instruction-Tuning* yang mana model dilatih menggunakan dataset yang berisi pasangan prompt (instruksi) dan response (jawaban) agar model lebih memahami konteks dan maksudnya.

Data yang telah dibersihkan dan disiapkan dibagi menjadi set pelatihan dan pengujian. Penelitian menunjukkan bahwa penggunaan 80% data untuk pelatihan dan 20% untuk pengujian menghasilkan hasil yang optimal (Gholamy et al., 2018). Pembagian ini penting untuk menguji efektivitas model secara objektif. Set pelatihan digunakan untuk proses *fine-tuning* dan set pengujian untuk evaluasi kinerja model yang mandiri setelah pelatihan selesai.

3.1.3.3 Proses Fine-Tuning Model GPT

Setelah persiapan data selesai, langkah berikutnya adalah proses *fine-tuning* model dengan menggunakan *base* model GPT-3.5 dengan metode "instruction-tuning" " sesuai dokumentasi resmi OpenAI. Proses ini dilakukan untuk mengadaptasi model pra-latih dari OpenAI sehingga lebih sesuai dengan kebutuhan spesifik dan konteks sistem yang ditargetkan. *Fine-tuning* menggunakan panduan dokumentasi resmi yang disediakan oleh OpenAI. Pada dokumentasi tersebut, terdapat beberapa tahapan utama yang mana terdiri dari persiapan dan *upload* set data untuk *fine-tuning*, *training* model *fine-tuning*, evaluasi hasil *fine-tuning*, dan implementasi model *fine-tuning*. Pada tahapan sebelumnya, telah dilakukan persiapan data untuk *fine-tuning* sehingga bisa ke tahapan berikutnya yang mana adalah *upload* set data untuk *fine-tuning* ke OpenAI.

app > fine tuning > upload_dataset.py

```python
import openai
import os
from dotenv import load_dotenv

load_dotenv()
openai.api_key = os.getenv("OPENAI_API_KEY")

if openai.api_key is None:
    print("openai api key is missing. please check your .env file.")
    exit()

try:
    with open("data_train.jsonl", "rb") as file:
        response = openai.File.create(file=file, purpose="fine-tune")
        print(f"file uploaded successfully. file id: {response['id']}")
except Exception as e:
    print(f"error uploading file: {e}")

```

Gambar 3.12 Upload Dataset Fine-Tuning

Pada tahapan *upload* set data untuk *fine-tuning* seperti pada gambar 3.12, terlihat proses mengunggah dataset ke OpenAI untuk *fine-tuning* model GPT-3.5. Proses ini diawali dengan impor modul openai untuk interaksi API, os untuk akses sistem operasi, dan dotenv untuk mengelola informasi dari file .env. Fungsi load_dotenv() digunakan untuk memuat variabel lingkungan sebelum memeriksa keberadaan OPENAI_API_KEY menggunakan os.getenv(). Jika kunci API tidak ditemukan, program akan menghentikan eksekusi. Pengunggahan file dataset dilakukan dengan memanfaatkan fungsi openai.File.create(), yang diawali dengan membuka file dalam mode baca biner ('rb'). Setelah file terbuka, parameter purpose diatur ke nilai "fine-tune" untuk memastikan bahwa file yang diunggah dikenali sebagai dataset yang akan digunakan dalam proses pelatihan model. Keberhasilan proses pengunggahan ini dikonfirmasi dengan pencetakan ID file yang diberikan oleh OpenAI setelah unggahan berhasil. ID file ini menjadi referensi utama yang akan digunakan pada tahapan selanjutnya dalam fine-tuning, memastikan bahwa dataset telah tersedia dalam sistem OpenAI untuk langkah selanjutnya dalam *fine-tuning* model.

app > fine tuning > fine_tuning_model.py

```python
import openai
import os
from dotenv import load_dotenv

load_dotenv()
openai.api_key = os.getenv("OPENAI_API_KEY")
model = os.getenv("OPENAI_MODEL")
training_file_id = os.getenv("OPENAI_TRAINING_FILE_ID")
method = os.getenv("OPENAI_METHOD")

required_vars = [openai.api_key, model, training_file_id, validation_file_id, method]
missing_vars = [var for var in required_vars if var is None]

if missing_vars:
    print(f"missing the following variables in .env file: {', '.join(missing_vars)}")
    exit()

# Hyperparameters
batch_size = None
learning_rate_multiplier = None
n_epochs = None

try:
    fine_tune_response = openai.FineTuningJob.create(
        training_file=training_file_id,
        model=model,
        method=method,
        n_epochs=n_epochs,
        batch_size=batch_size,
        learning_rate_multiplier=learning_rate_multiplier
    )
    print(f"fine-tune job created. job id: {fine_tune_response['id']}")

except Exception as e:
    print(f"error creating fine-tune job: {e}")
```

Gambar 3.13 Fine-Tuning Model GPT-3.5

Pada Gambar 3.13 terlihat langkah selanjutnya dalam proses *fine-tuning* model GPT-3.5. Hal ini meliputi beberapa tahap yang dilakukan setelah persiapan dan *upload* set data. Pada tahap ini menggunakan load_dotenv() untuk memuat variabel lingkungan, yang mencakup OPENAI_API_KEY, OPENAI_MODEL, OPENAI_METHOD, dan OPENAI_TRAINING_FILE_ID. Untuk *fine-tuning* ini, menggunakan metode openai 'supervised' karena memungkinkan pelatihan model secara terstruktur dengan dataset yang memiliki label yang jelas sehingga memastikan model dapat belajar dari data yang spesifik sesuai dengan tujuan yang diinginkan. Variabel tersebut penting untuk memastikan akses ke API dan menentukan parameter *fine-tuning* yang akan digunakan. Selanjutnya, melakukan pengecekan untuk memastikan semua variabel yang diperlukan tersedia. Jika ada variabel yang hilang, program akan mencetak variabel yang hilang dan menghentikan eksekusi untuk menghindari kesalahan selanjutnya

dalam proses *fine-tuning*. Hyperparameter seperti batch_size, learning_rate_multiplier, dan n_epochs diatur sebagai None atau secara *default* yang berarti akan menggunakan nilai default OpenAI yang sesuai kebutuhan *fine-tuning*. Dengan openai.FineTuningJob.create(), job *fine-tuning* diinisiasi menggunakan file training, model, metode, dan *hyperparameter* yang telah ditentukan. Metode *fine-tuning* yang digunakan adalah *supervised* yang menekankan pembelajaran dari dataset berlabel untuk meningkatkan spesifikasi respons model terhadap situasi yang ditargetkan. Fungsi ini mengirimkan permintaan ke OpenAI untuk memulai proses *fine-tuning* dengan konfigurasi yang ditentukan. Jika job berhasil dibuat, ID job akan ditampilkan sehingga dapat memantau status job *fine-tuning*.

3.1.3.4 Evaluasi Kinerja Model GPT Setelah Fine-Tuning

Setelah proses *fine-tuning base* model GPT-3.5, tahap berikutnya adalah evaluasi kinerja untuk memastikan bahwa *fine-tuning* yang telah dilakukan berhasil meningkatkan kemampuan *base* model sesuai dengan kebutuhan sistem yang ditargetkan. Evaluasi ini dilakukan dengan beberapa metrik salah satunya menggunakan metrik yang dihasilkan setelah fine-tuning yang mana memberikan gambaran tentang seberapa baik model bekerja dengan data baru. Metrik *training loss* yang mana mengukur seberapa baik model memprediksi data selama fase pelatihan. *Training loss* yang semakin rendah menunjukkan bahwa model dengan akurat mempelajari detail dari set pelatihan.

Selain itu, evaluasi dilakukan dengan menganalisis hasil penggunaan token pada model GPT-3.5 yang telah di *fine-tuning*. Data penggunaan token diperoleh melalui proses logging selama 280 interaksi dengan model GPT-3.5 fine-tuned dari 280 interaksi. Terdapat tiga jenis token seperti dijelaskan sebelumnya di bagian implementasi *base* model GPT-3.5 tanpa *fine-tuning*, prompt_tokens adalah jumlah token yang digunakan untuk menyusun prompt yang dikirim ke model, completion_tokens adalah jumlah token yang dihasilkan oleh model

sebagai respons terhadap prompt, sedangkan total_tokens adalah jumlah gabungan dari prompt_tokens dan completion_tokens. Setelah dilakukan evaluasi melalui analisis penggunaan token baik dari *base* model GPT-3.5 maupun model GPT-3.5 yang telah di *fine-tuning* maka dilakukan perbandingan penggunaan token antara kedua model tersebut. Menurut dokumentasi resmi OpenAI, salah satu manfaat dari *fine-tuning* model adalah penghematan penggunaan token yang mana akan menghemat sumber daya yang digunakan juga. Hal ini terjadi karena *prompt* yang digunakan pada model yang telah di *fine-tuning* lebih singkat dibandingkan dengan *base* model. Model yang telah di *fine-tuning* sudah dilatih untuk memahami konteks khusus lebih efisien sehingga *prompt* yang diperlukan menjadi langsung ke inti kebutuhan. Dengan demikian, jumlah token yang digunakan untuk mendapatkan respons yang diinginkan dapat berkurang signifikan.

```
from bert_score import score

references = [
    "reference sentence",
    "reference sentence",
    ...
    "reference sentence"
]
candidates = [
    "candidate sentence",
    "candidate sentence",
    ...
    "candidate sentence"
]

P, R, F1 = score(candidates, references, lang="id")
print("Precision:", P)
print("Recall:", R)
print("F1 Score:", F1)
```

Gambar 3.14 Evaluasi dengan BERTScore

Terakhir, model dievaluasi dengan set pengujian yang belum pernah "dilihat" oleh model menggunakan metrik evaluasi BERTScore. Hal ini memberikan pengujian yang objektif terhadap kinerja model dalam kondisi nyata. Pada Gambar 3.14 menunjukkan evaluasi dengan BERTScore membandingkan kemiripan semantik antara kalimat yang

dihasilkan oleh model (*candidates*) dengan kalimat referensi yang mana kalimat dianggap benar atau standar (*references*) dengan *library* BERTScore yang menghitung tiga metrik evaluasi utama yaitu *Precision*, *Recall*, dan *F-1 Score*. *Precision* mengukur seberapa banyak kata-kata dalam kalimat yang dihasilkan model (*candidates*) yang relevan terhadap kalimat referensi (*references*). *Precision* tinggi menunjukkan bahwa kebanyakan kata di kalimat candidates memang relevan dengan kalimat references. *Recall* mengukur seberapa banyak kata-kata dalam kalimat referensi (*references*) yang berhasil ditangkap oleh kalimat yang dihasilkan model (*candidates*). *Recall* yang tinggi menunjukkan bahwa kalimat yang dihasilkan oleh model (*candidates*) berhasil mencakup atau menangkap banyak informasi yang relevan dari kalimat referensi (*references*). *F1-Score* merupakan *harmonic mean* dari *Precision* dan *Recall* yang memberikan keseimbangan antara kedua metrik tersebut. *F1-Score* tinggi menunjukkan bahwa model berhasil menjaga keseimbangan antara relevansi dan kelengkapan. Dalam BERTScore, nilai *Precision*, *Recall*, dan *F1-Score* berada dalam rentang 0 hingga 1, di mana 0 menunjukkan tidak ada kesamaan antara kalimat *candidates* dan *references*. Sementara itu, 1 menunjukkan kesamaan sempurna antara kalimat *candidates* dan *references*. Setelah itu, dilakukan perbandingan hasil BERTScore baik base model GPT-3.5 maupun fine-tuned model GPT-3.5 untuk mengetahui perbedaan performa dalam mengahsilkan intervensi afektif yang relevan. Terakhir, hasil dari BERTScore kemudian ditinjau bersama dengan hasil penggunaan token untuk memberikan gambaran terkait performa model dalam menghasilkan output yang relevan dan efisien. Evaluasi komprehensif ini menentukan apakah model sudah memenuhi standar yang ditetapkan dan siap untuk diimplementasikan atau memerlukan *fine-tuning* lebih lanjut untuk meningkatkan kinerjanya.

3.1.3.5 Implementasi Model GPT Setelah di Fine-Tuning

Setelah menyelesaikan proses evaluasi kinerja untuk model *Generative Pre-trained Transformer* 3.5 (GPT-3.5) yang telah di *fine-*

tuning, langkah berikutnya adalah mengimplementasikan model tersebut dalam lingkungan produksi yang nyata. Proses ini melibatkan penerapan model ke dalam *Application Programming Interface* (API) yang memungkinkan model tersebut untuk menerima, memproses, dan merespons permintaan dari sistem atau layanan eksternal lain. Implementasi pengembangan API menggunakan bahasa pemrograman Python dengan *framework* FastAPI yang sesuai untuk membuat API yang cepat dan efisien dengan beberapa tahapan pengembangan yang telah dijelaskan pada sub bab 3.1.3.1 terkait tahapan implementasi *base* model GPT tanpa *fine-tuning*.

Salah satu perbedaan utama dalam implementasi model *fine-tuned* ini terletak pada prompt yang digunakan. Prompt yang digunakan setelah proses fine-tuning cenderung lebih singkat karena model telah melalui tahapan pelatihan tambahan dengan data yang spesifik dan relevan. Proses *fine-tuning* memungkinkan model untuk lebih memahami konteks secara mendalam, sehingga tidak memerlukan instruksi yang terlalu panjang atau terperinci untuk menghasilkan respons yang tepat. Hal ini mengurangi beban penyusunan prompt karena model sudah dilatih untuk mengenali pola-pola dalam data yang disediakan selama proses fine-tuning. Sebagai hasilnya, prompt dapat dirancang dengan lebih efisien, cukup menggunakan kalimat atau frasa kunci yang langsung menggambarkan kebutuhan atau konteks.

```
Berikan intervensi afektif berupa kalimat motivasi untuk meningkatkan kesejahteraan dan performa pelajar selama pembelajaran online di Google Meet berdasarkan informasi input emotion, reinforcement positive atau negative, arcs model, classname, subject, dan username dengan output bahasa indonesia tidak formal, maksimal 25 kata, dan sertakan emoji.
```

Gambar 3.15 Prompt untuk Fine-Tuned Model GPT-3.5

Berikut merupakan ilustrasi prompt untuk implementasi model fine-tuned pada Gambar 3.15, di mana model menunjukkan peningkatan kemampuan dalam memahami dan merespons instruksi yang lebih padat dan ringkas. Hal ini menunjukkan dampak signifikan dari proses fine-tuning dalam meningkatkan efisiensi interaksi dengan model. Setelah pengembangan selesai, langkah selanjutnya adalah pengujian *Application Programming Interface* menggunakan Postman untuk memastikan fungsionalitas berjalan sesuai yang diharapkan. Pengujian ini meliputi simulasi permintaan dari klien untuk memverifikasi bagaimana *Application Programming Interface* menangani berbagai skenario input dan memastikan output yang dihasilkan sesuai dengan harapan. Setelah pengujian, proses *deployment Application Programming Interface* dilakukan. Deployment ini menggunakan Docker untuk *containerization* dan Google Cloud Platform untuk memanfaatkan infrastruktur *cloud* yang skalabel dan aman. Setelah proses deployment selesai, API dapat diintegrasikan ke dalam sistem Emodu yang menggunakan teknologi sensor pengenalan emosi untuk mendeteksi emosi pelajar. Integrasi ini memungkinkan model GPT-3.5 yang telah di *fine-tuning* untuk menghasilkan intervensi afektif berupa teks dapat disesuaikan dengan emosi pelajar sehingga dapat mendukung kesejahteraan emosional dan meningkatkan hasil pembelajaran. Setelah Integrasi, pemantauan berkelanjutan dilakukan untuk mengevaluasi performa model.

3.1.4 Studi Deskriptif II

Pada tahap Studi Deskriptif II dilakukan pengumpulan data secara langsung untuk mengevaluasi fenomena yang diteliti (Blessing et al., 2009). Setelah data dikumpulkan, tahap berikutnya adalah analisis dan evaluasi hasil penelitian. Pada tahap ini, data yang telah dikumpulkan dianalisis secara mendalam untuk menilai efektivitas sistem yang telah diimplementasikan. Analisis ini mencakup evaluasi terhadap hasil dari beberapa metrik evaluasi yang digunakan. Evaluasi ini bertujuan untuk menentukan seberapa baik sistem memenuhi tujuan penelitian awal dan hipotesis yang telah

dikembangkan serta untuk mengidentifikasi area yang mungkin perlu perbaikan. Hasil dari analisis dan evaluasi ini kemudian didokumentasikan secara menyeluruh. Dokumentasi ini mencakup laporan rinci tentang temuan dari pengumpulan data dan analisis, termasuk grafik, tabel, dan narasi yang menjelaskan bagaimana sistem berfungsi. Dokumentasi yang lengkap ini penting untuk memberikan gambaran komprehensif tentang efektivitas sistem dan untuk menyediakan informasi yang diperlukan untuk perbaikan atau pengembangan lebih lanjut dari sistem.

3.2 Alat dan Bahan Penelitian

Pada penelitian ini beberapa alat dan bahan digunakan untuk mendukung implementasi *Artificial Intelligence* menggunakan OpenAI dalam memberikan intervensi afektif berbasis pengenalan emosi siswa selama pembelajaran daring sinkronis. Berikut ini adalah rincian alat dan bahan yang diperlukan.

1. Perangkat Keras
    - Laptop dengan spesifikasi sebagai berikut.
        - Processor Chip Apple M1 Pro
        - Penyimpanan 1TB SSD
        - Memori 16 GB RAM
        - Kamera FaceTime HD 1080p
        - Sistem Operasi MacOS Sequoia
2. Perangkat Lunak
    - OpenAI
    - Generative Pre-trained Transformer 3.5 (GPT-3.5)
    - Bahasa Pemrograman Python
    - Framework FastAPI
    - BERTScore
    - Visual Studio Code
    - Google Colab
    - Postman
    - Google Cloud Platform
    - GitHub

Dengan penggunaan alat dan bahan tersebut, penelitian ini diharapkan dapat memberikan hasil yang valid dan reliabel dalam mengevaluasi bagaimana implementasi *Artificial Intelligence* menggunakan OpenAI dapat memberikan intervensi afektif yang relevan secara *real-time* berdasarkan pengenalan emosi siswa selama pembelajaran daring sinkronis.

3.3 Teknik Pengumpulan Data Penelitian

Dalam penelitian ini, teknik pengumpulan data dilakukan dengan menggunakan beberapa metode yang terstruktur untuk memastikan data yang diperoleh dapat diandalkan dan relevan dengan tujuan penelitian. Metode yang digunakan mencakup pemanfaatan model Generative Pre-trained Transformer (GPT) yang diterapkan dalam dua bentuk, yaitu model yang telah disesuaikan melalui proses *fine-tuning* dan model dasar tanpa modifikasi. Proses *fine-tuning* dilakukan dengan menggunakan dataset spesifik yang dirancang untuk memenuhi kebutuhan penelitian ini sehingga dapat menghasilkan output yang tepat dan relevan. Sementara itu, model dasar digunakan sebagai pembanding untuk mengamati bagaimana *fine-tuning* pada model mempengaruhi kinerja dalam menghasilkan teks. Selain penggunaan model GPT, penelitian ini juga melibatkan pengembanga dataset dengan teknik text data augmentation menggunakan ChatGPT dengan strategi *prompting* untuk meningkatkan keragaman dan volume data. Text data augmentation mencakup metode seperti paraphrasing, synonym replacement, random insertion, random swap, dan back translation dari dataset yang tersedia yang terverifikasi oleh ahli di bidang Psikologi dan Pendidikan untuk dijadikan dasar augmentasi yang memungkinkan penciptaan variasi teks yang lebih beragam. Hal ini bertujuan untuk membangun set data yang komprehensif dan dapat diandalkan untuk keperluan evaluasi dan analisis lebih lanjut.

3.4 Teknik Analisis Data Penelitian

Dalam penelitian ini, teknik analisis data menggunakan beberapa metode evaluasi untuk mengevaluasi kualitas *output* dari model Generative Pre-trained Transformer (GPT) baik hasil *fine-tuning* maupun *base* model. Evaluasi ini dilakukan dengan tujuan untuk memberikan gambaran yang komprehensif tentang seberapa relevan dan efisien model dalam menghasilkan teks yang relevan dan

berkualitas sesuai dengan konteks penelitian. Evaluasi *fine-tuning* juga dilakukan menggunakan metrik yang lebih spesifik untuk menilai proses pelatihan dan kemampuan model dalam menghasilkan teks yang lebih baik seiring dengan proses *fine-tuning*. Metrik yang digunakan untuk mengevaluasi *fine-tuning* meliputi *training loss*. *Training loss* mengukur seberapa besar kesalahan model saat pelatihan pada data pelatihan. Nilai loss yang lebih rendah menunjukkan model yang lebih baik dalam memprediksi hasil yang diinginkan selama pelatihan.

Selanjutnya, evaluasi kinerja hasill *fine-tuning* dengan penggunaan token dari model yang telah di fine-tuning dengan *base* model. Menurut dokumentasi resmi OpenAI, salah satu manfaat dari *fine-tuning* model adalah penghematan penggunaan token yang mana akan menghemat sumber daya yang berupa biaya. Hal ini terjadi karena *prompt* yang digunakan pada model yang telah di *fine-tuning* lebih singkat dibandingkan dengan model dasar. Model yang telah di *fine-tuning* telah dilatih untuk memahami konteks khusus lebih efisien sehingga *prompt* yang diperlukan menjadi langsung ke inti kebutuhan. Dengan demikian, jumlah token yang digunakan untuk mendapatkan respons yang relevan dapat berkurang signifikan.

Sementara itu, Evaluasi kinerja hasil dilakukan dengan metrik evaluasi BERTScore menggunakan set data pengujian yang tidak digunakan dalam proses pelatihan atau *fine-tuning* untuk memastikan bahwa evaluasi yang dilakukan bersifat objektif dan representatif terhadap kinerja model secara keseluruhan. BERTScore merupakan metrik evaluasi utama yang digunakan untuk menilai kualitas teks yang dihasilkan dengan cara menghitung skor kesesuaian antara vektor kata dalam respons yang dihasilkan oleh model dengan teks referensi yang mana teks yang dianggap standar atau benar. Skor ini dihitung berdasarkan *cosine similarity* yang merupakan ukuran kesamaan antara dua vektor dan mengindikasikan seberapa dekat kedua teks tersebut dari segi semantik. BERTScore menggunakan representasi yang diperoleh dari model BERT (Bidirectional Encoder Representations from Transformers) yang telah dilatih pada kumpulan data besar dan mampu menghasilkan representasi kata yang kaya akan konteks. Hasil dari metrik evaluasi BERTScore terdiri dari tiga nilai, yaitu Precision, Recall, dan F1-Score. Precision mengukur sejauh mana kata-kata yang

dihasilkan oleh model relevan dengan kata-kata dalam teks referensi. Precision yang lebih tinggi menunjukkan bahwa model menghasilkan lebih banyak kata yang tepat dan relevan dari teks referensi. Recall mengukur sejauh mana kata-kata dalam teks referensi dapat ditemukan dalam teks yang dihasilkan oleh model. Recall yang lebih tinggi menunjukkan bahwa model mampu menangkap sebagian besar kata-kata penting dalam teks referensi. F1-Score merupakan rata-rata harmonis antara precision dan recall. F1-Score memberikan gambaran umum tentang keseimbangan antara presisi dan sensitivitas model. Nilai F1-Score yang lebih tinggi menunjukkan performa yang lebih baik dalam menghasilkan teks yang baik dari segi keakuratan dan kelengkapan. Dalam BERTScore, nilai Precision, Recall, dan F1-Score berada dalam rentang 0 hingga 1, di mana 0 menunjukkan tidak ada kesamaan antara kalimat candidates dan references (Zhang et al., 2019). Sementara itu, 1 menunjukkan kesamaan sempurna antara kalimat candidates dan references (Zhang et al., 2019).

Dalam analisis data penelitian ini, penggunaan berbagai jenis visualisasi dan tabel statistik deskriptif menjadi untuk memahami dan menginterpretasikan kinerja serta efisiensi model Generative Pre-trained Transformer (GPT) baik versi dasar maupun yang telah di fine-tuning. Visualisasi seperti diagram batang efektif untuk membandingkan hasil BERTScore dan penggunaan token antara *base* model dan model fine-tuned serta hasil Augmentasi Data Teks. Sementara itu, grafik garis dalam memvisualisasikan tren training loss selama proses pelatihan menunjukkan bagaimana model semakin optimal dalam memprediksi atau mengklasifikasikan data seiring waktu. Selain itu, tabel statistik deskriptif yang mencakup jumlah total data, rata-rata, simpangan baku, nilai minimum, dan maksimum, serta persentil, memberikan detail kuantitatif yang mendalam tentang kinerja model. Mean adalah nilai rata-rata yang diperoleh dengan menjumlahkan semua data dan membaginya dengan jumlah data tersebut (Banning, 2020). Median adalah nilai tengah dari data yang telah diurutkan, membagi data menjadi dua bagian yang sama besar (Banning, 2020). Kuartil membagi data terurut menjadi empat bagian yang sama besar, dengan kuartil pertama (Q1) membagi 25% data terendah dan kuartil ketiga (Q3) membagi 75% data terendah (Banning, 2020). Standar deviasi mengukur seberapa

jauh data tersebar dari mean, menunjukkan tingkat variasi atau keragaman dalam data (Banning, 2020).

Dengan demikian, analisis data ini tidak hanya mengukur seberapa baik model Generative Pre-trained Transformer (GPT) baik hasil *fine-tuning* maupun *base* model dalam menghasilkan teks yang semantik sesuai dengan referensi melalui BERTScore dan penggunaan token yang memberikan indikasi langsung tentang efisiensi model dalam memanfaatkan sumber daya ketika berinteraksi dengan model, tetapi juga memberikan pemahaman tentang bagaimana model beradaptasi dan meningkatkan kemampuannya selama proses *fine-tuning* melalui metrik training loss.