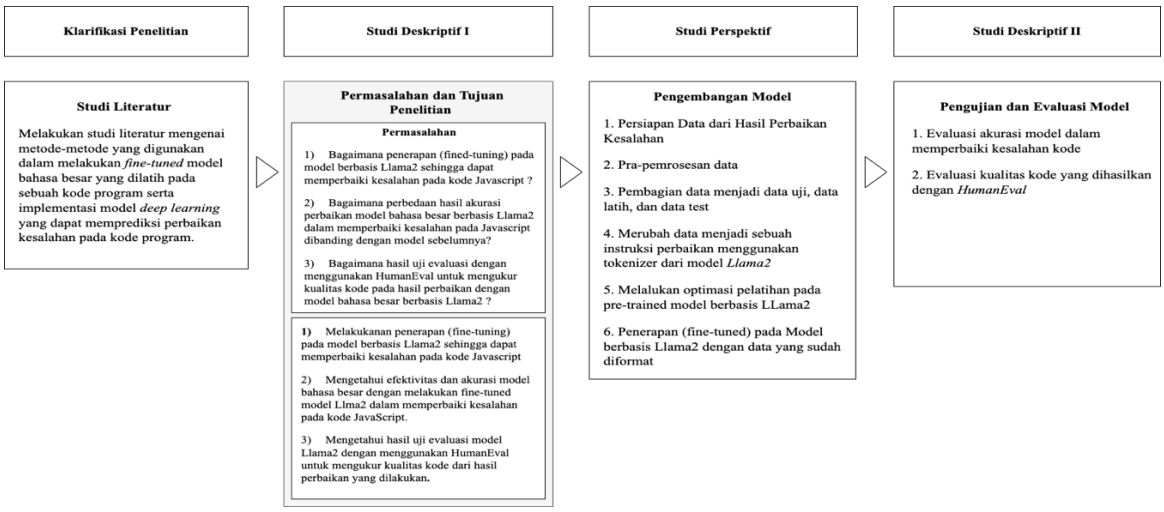
BAB III
METODE PENELITIAN

3.1 Desain Penelitian

Untuk mencapai tujuan penelitian dengan cara yang valid dan efisien, diperlukan sebuah Desain Penelitian. Desain Penelitian merupakan sebuah kerangka atau rencana sistematis yang digunakan untuk mengarahkan dan mengorganisasi sebuah studi atau penelitian. *Design Research Methodology* (DRM) merupakan salah satu pendekatan desain penelitian yang berfokus pada perumusan, validasi penelitian, dan teori tentang bagaimana desain penelitian berjalan (Blessing dan Chakrabarti, 2009). DRM mengembangkan dan memeriksa berbagai pengetahuan, metode yang digunakan, dan alat yang bisa digunakan berdasarkan teori-teori tersebut untuk membuat proses desain menjadi lebih baik. Fokus utama dari penelitian ini adalah mengembangkan dan menguji metode baru dalam memprediksi perbaikan error kode program. Karena memiliki fokus dan tujuan yang sama, DRM digunakan sebagai desain penelitian pada penelitian ini.

Berdasarkan (Blessing dan Chakrabarti, 2009) *Design Research Methodology* (DRM) memiliki enam tahap penelitian yang diantaranya Klarifikasi Penelitian, Studi Deskriptif I, Studi Preskriptif dan Studi Deskriptif II. Pada Gambar 3.1 diperlihatkan penerapan tahapan DRM pada penelitian ini.



Gambar 3.1 Desain Penelitian

3.1.1 Klarifikasi Penelitian

Tahap awal dalam penelitian ini adalah Melakukan studi literatur mengenai metode-metode yang digunakan dalam melakukan *fine-tuning* terhadap *Large Language Model (*LLM) yang dilatih pada sebuah kode program serta implementasi model *deep learning* yang dapat memprediksi perbaikan kesalahan pada kode program. Hasil dan saran dari Penelitian terdahulu, dapat dijadikan untuk bahan rujukan dalam pembuatan model prediksi perbaikan kesalahan penulisan sintaks kode pada penelitian ini.

3.1.2 Studi Deskriptif I

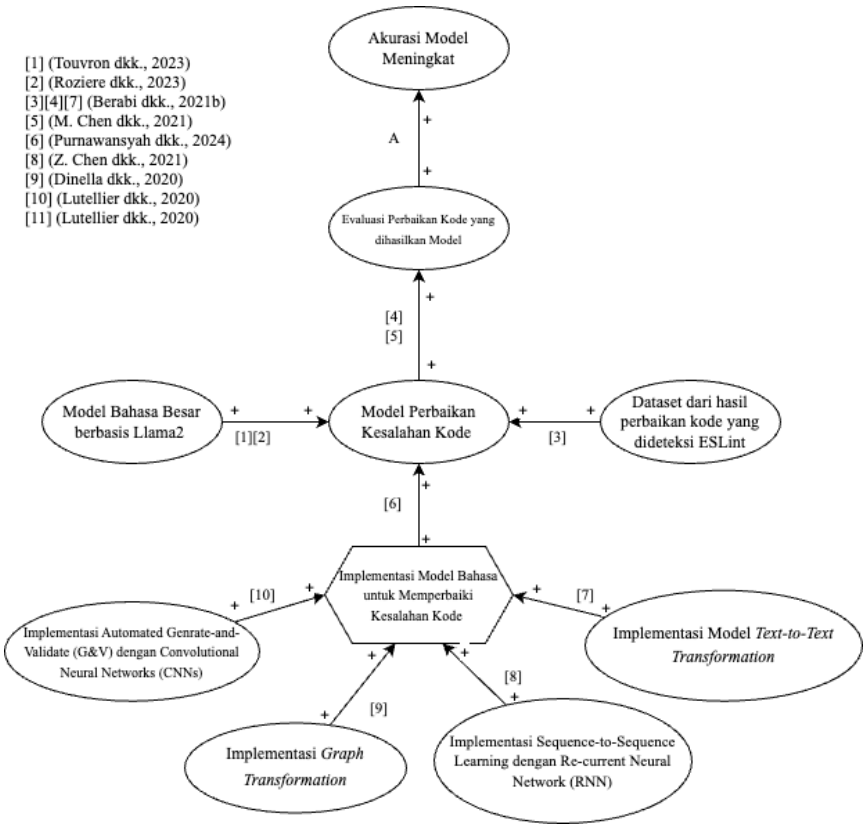
Tahapan kedua yang dilakukan adalah menjelaskan permasalahan dan tujuan penelitian yang mengikuti hasil studi literatur yang dilakukan sebelumnya. Hasil dari tahapan ini merupakan fondasi dalam membangun solusi dari permasalahan penelitian ini.

3.1.3 Studi Perspektif

Tahapan selanjutnya, adalah prosedur pembuatan *Large Language Model* (LLM) yang berbasis *Llama2*. Langkah-langkah dan prosedur dalam pembuatan model mengacu pada penelitian terdahulu yang kemudian disesuaikan dengan objek penelitian yang baru untuk meningkatkan performa dari model yang sedang dikembangkan.

3.1.3.1 Model Dampak

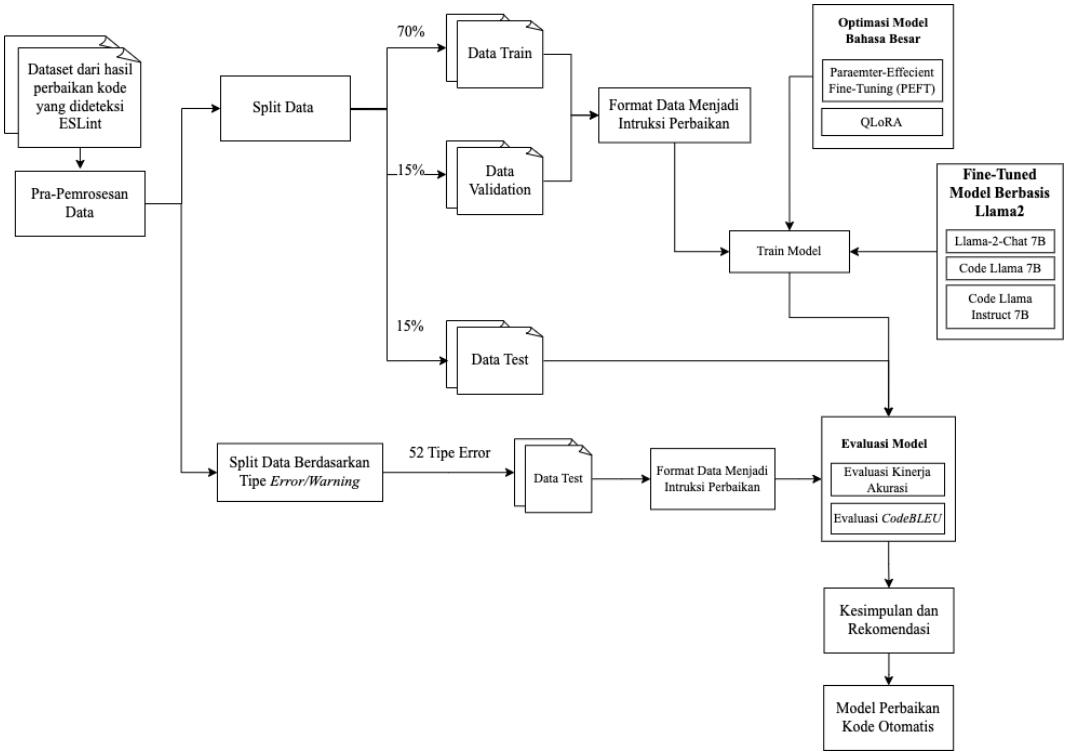
Model dampak pada penelitian ini digambarkan Pada Gambar 3.2. Model Dampak merupakan sketsa atau visual yang menggambarkan alur dari pengembangan model yang akan dibuat dengan mengambil rujukan dari penelitian sebelumnya terkait metode *deep learning* khususnya model bahasa besar yang dilatih pada dataset sebuah kode. Model Dampak merepresentasikan visual yang disusun oleh penulis dan bertujuan sebagai panduan dalam mencapai hasil sesuai dengan visi misi penulis.



Gambar 3.2 Model Dampak

3.1.3.2 Pengembangan Model Perbaikan Kesalahan Kode Program

Tahapan pengembangan model digambarkan menjadi diagram alur yang dapat dilihat pada Gambar 3.3. Pada tahap ini menjelaskan pengerjaan *Large Language Model* (LLM) berbasis *Llama2* yang dimana model tersebut sudah dilatih dengan korpus data yang besar agar dapat meningkatkan akurasi dari prediksi sebuah teks. Model berbasis *Llama2* yang digunakan diantaranya *Llama2 Chat*, *Code Llama*, dan *Code Llama Instruct* dengan varian 7B parameter. Model-model tersebut dilakukan penyesuaian atau *fine-tuned* dengan dataset yang telah disesuaikan untuk tugas yang lebih spesifik dalam melakukan perbaikan kesalahan penulisan sintaks kode JavaScript.



Gambar 3.3 Diagram Alur Pengembangan Model Prediksi Perbaikan Kesalahan Penulisan Sintaks Kode JavaScript

3.1.3.3 Pengumpulan Data

Pada tahap pengumpulan data, dataset yang digunakan pada penelitian bersumber pada penelitian (Berabi dkk., 2021b). Data tersebut berisi 104.804 pasang kode error yang dideteksi oleh ESLint dan perbaikan dari kode tersebut berdasarkan commit pada Github. Dataset yang digunakan merupakan hasil yang sudah diekstraksi dari hasil laporan error ataupun warning yang dideteksi oleh alat analisis statis *ESLint*. Dataset berisi informasi mengenai file sumber dari kode yang belum diperbaiki dan kode yang sudah diperbaiki. Pada penelitian (Berabi dkk., 2021b) mengklaim dataset ini memiliki kualitas yang lebih baik dibandingkan dengan dataset yang dibuat secara sintesis dalam mendeteksi sebuah *bugs* (Vasic, Kanade, Maniatis, Bieber, dan Singh, 2019). Pada Tabel 3.1 menampilkan deskripsi data dari file json dataset yang digunakan pada. Deskripsi secara keseluruhan ditampilkan pada Lampiran 1.

Tabel 3.1

Deskripsi Dataset yang Digunakan

Kolom	Deskripsi
linter_report	Berisi informasi detail terkait laporan kesalahan atau peringatan pada kode, termasuk pesan yang disajikan kepada pengguna dan jenis kesalahan yang teridentifikasi berdasarkan laporan ESLint.
repo	Informasi mengenai *repository* path file pada Github.
source_changeid	Berisi terkait id perubahan kode yang bermasalah pada commit Github
source_code	Kode yang bermasalah
source_file	Menampilkan keseluruhan isi file pada Kode yang bermasalah
source_filename	Nama dan path dari file kode yang bermasalah
target_changeid	Berisi terkait id perubahan kode yang sudah diperbaiki pada commit Github
target_code	Kode yang sudah diperbaiki

3.1.3.4 Pra-Pemrosesan Data

Pada tahap ini hal yang pertama dilakukan mengubah dataset yang berbentuk json kedalam sebuah kelas *Datasets* pada library *python* yang berfungsi untuk mempermudah dalam melakukan penyesuaian data. Hanya beberapa kolom data yang diambil pada data json tersebut. Diantaranya laporan adanya kesalahan dari *ESLint*, kode yang perlu diperbaiki dan kode yang sudah diperbaiki. Dengan Menggunakan bantuan *package python* yang dinamakan *Datasets* dengan versi yang digunakan v2.20.0, mempermudah dalam melakukan split data train dengan data validation. Merujuk pada penelitian (Tehrani, Calvello, Liu, Zhang, dan Lacasse, 2022; Uc-Castillo, Marín-Celestino, Martínez-Cruz, Tuxpan-Vargas, dan Ramos-Leal, 2023), Jika sebuah data memiliki jumlah yang sangat banyak, Datasets

dapat dibagi kedalam kelompok *train*, *validation*, dan *test* dengan data train memiliki total sebesar 70% dari jumlah data dan data validation berjumlah 15% dari total data. Sisa dari jumlah yakni 15%, dapat digunakan sebagai data test yang nantinya akan dimanfaatkan pada perhitungan evaluasi keakuratan model dalam memperbaiki kesalahan penulisan sintaks *kode* JavaScript. Pada data test sendiri selain mengambil 15% dari total data, data test akan mengambil semua data yang ada yakni sekitar 104.804 data dengan membaginya berdasarkan 52 aturan atau tipe kesalahan yang terdaftar pada ESLint. Hal ini dilakukan untuk membandingkan hasil evaluasi model dengan model sebelumnya yakni *TFix* (Berabi dkk., 2021b).

3.1.3.5 Format dan Tokenisasi Perintah (*Prompt*)

Agar mencapai hasil yang maksimal, data diubah menjadi sebuah format instruksi. Hal ini bertujuan untuk melakukan optimasi saat pelatihan dengan pendekatan *Supervised Fine-Tuning*. Pendekatan *Supervised Fine-Tuning* melakukan *tokenizing* terhadap label yang nantinya memiliki nilai yang sama dengan input yang di tokenisasi. Format yang dibuat seperti melakukan perintah pada sebuah sistem robot untuk melakukan task spesifik dalam memperbaiki kesalahan pada kode program. Data yang dibutuhkan untuk melakukan format *prompting* ini adalah laporan linter yang memiliki data terkait tipe error atau warning yang dideteksi oleh *ESLint* dan pesan dari laporan linter tersebut. Dua data tersebut dijadikan sebuah Instruksi agar model mengetahui apa yang akan dikerjakan dan dapat mengelompokan tugas perbaikan kode program sesuai dengan tipe error yang diberikan.

3.1.3.6 Perbaikan Kode dengan *Large Language Model* (LLM)

Data yang dilakukan format perintah instruksi perbaikan dan tokenisasi, kemudian dilatih dengan *pre-trained* model berbasi *Llama2* yang diantaranya *Llama2 Chat*, *Code Llama*, *Code Llama Instructs* dengan pilihan 7B parameter. Karena model tersebut sudah dilatih sebelumnya untuk tugas prediksi teks, maka kita hanya perlu melakukan penyesuaian (*fine-tuned*) pada model yang dibuat. Sebelum proses pelatihan dilakukan, model akan dilakukan optimasi sebelum

pelatihan dengan menggunakan *Parameter-Efficient Fine-Tuning* (Houlsby dkk., 2019) dan melakukan konfigurasi kuantitasi dengan QLoRA (Dettmers dkk., 2024). Dengan melakukan optimasi sebelum pelatihan, diharapkan dapat mempercepat proses pelatihan model bahasa besar dan tidak akan mengurangi akurasi prediksi model. Proses pelatihan model akan dilakukan pada 400 steps, batch size 128, dengan latihan per batch nya sekitar 32 dan learning rate 3e-4. Optimasi yang digunakan "adamw_torch" yang sudah disediakan library *transformers*.

3.1.4 Studi Deskriptif II

Pada tahap ini, setelah pengembangan model bahasa besar berbasis *Llama2* dilakukan penyesuaian pada dataset kode untuk melakukan perbaikan kesalahan kode JavaScript, Selanjutnya dilakukan evaluasi dan pengujian model untuk kemudian dijadikan kesimpulan dari penelitian ini. Untuk mengukur kinerja dan validasi kesesuaian hasil perbaikan kode pada model berbasis *Llama2*, menggunakan tiga tahapan evaluasi. Tahap pertama mengevaluasi model dengan data test yang diberikan dengan jumlah 15% dari keseluruhan data. Dilanjutkan dengan Evaluasi perbaikan berdasarkan tipe error atau warning yang terdaftar pada alat analisis statis ESLint dengan prosedur yang sama seperti pada pengujian *TFix* (Berabi dkk., 2021b). Terakhir mengevaluasi hasil model berbasis *Llama2* dengan metrik BLEU dan CodeBLEU untuk mengukur kemiripan dan kesamaan fungsionalitas dari referensi kode manusia yang sudah diperbaiki sebelumnya. Evaluasi model menggunakan library *python* tambahan yang Bernama *evaluate* untuk mempermudah dalam mengevaluasi model

3.1.4.1 Evaluasi Akurasi Model

Skor akurasi perbaikan pada model didapatkan pada perhitungan dasar dalam mendapatkan akurasi. Berdasarkan (Ehrlinger dan Wöß, 2022) skor akurasi didapatkan dengan menjumlahkan prediksi yang akurat atau benar dan membaginya dengan jumlah total dari hasil prediksi. Berikut notasi perhitungan skor akurasi yang disesuaikan dalam perhitungan akurasi perbaikan model bahasa besar.

$$Accuracy = \frac{\text{Jumlah Prediksi Akurat}}{Panjang\ Label\ Hasil\ Prediksi} \quad (10)$$

Jumlah prediksi yang akurat didapatkan dari membandingkan kesamaan antara hasil label prediksi dengan referensi dari label target perbaikan kode. Label dibagi menjadi sebuah token dan token tersebut yang dibandingkan. Jika antar token memiliki kesamaan maka akan dijumlahkan dan didapatkan akurasi perbaikan dari hasil model yang didapatkan. Perhitungan skor akurasi perbaikan dibantu dengan library *evaluate* v0.4.2 yang ada pada pemrograman python.

3.1.4.2 Evaluasi Akurasi Model Berdasarkan Tipe Kesalahan

Untuk melakukan perbandingan dengan model sebelumnya. Model dilakukan uji akurasi yang sama dengan tahapan pengujian yang dilakukan pada model Tfix (Berabi dkk., 2021b) dengan membagi semua data dan mengelompokkannya berdasarkan tipe error atau warning yang dapat dideteksi oleh ESLint. Setelah melakukan pengelompokan dilakukan perhitungan skor akurasi prediksi perbaikan dengan model berbasis *Llama2.* Dari hasil prediksi tersebut dibandingkan dengan model sebelumnya (TFix). Akurasi yang menampilkan data yang sudah dikelompokan 52 aturan yang dilaporkan ESLint.

3.1.4.3 Evaluasi BLEU dan CodeBLEU

Selanjutnya melakukan evaluasi uji dengan menggunakan metrik BLEU dan CodeBLEU untuk mengukur kualitas kode yang dihasilkan dengan membandingkan kesamaan antara label kode perbaikan dengan kode prediksi. Perbedaan dengan menggunakan akurasi, dengan CodeBLEU dapat mengenali suatu fungsi akan dianggap sama meskipun dituliskan berbeda kode hasil prediksi. CodeBLEU dapat mengukur kesamaan secara keseluruhan antara label kode dan prediksi dibanding dengan menggunakan Akurasi. Notasi pada CodeBLEU yang digunakan pada penelitian ini adalah sebagai berikut.

$$CodeBLEU = \alpha \cdot score\ BLEU - \beta \cdot Jumlah\ Token\ Serupa - \quad (11)$$

$$\gamma \cdot Kesamaan\ Token\ pada\ format\ AST - \delta \cdot Kesamaan\ pada\ Flow\ Data$$

Hasil skor BLEU dan CodeBLEU berupa angka persentase antara 0 sampai 100 yang mengukur kesamaan antara hasil prediksi dengan label yang menjadi referensi atau hasil prediksi yang diinginkan, dengan menhitung n-gram pada sebuah hasil prediksi model dengan skor CodeBLEU yang tidak mengabaikan struktur sintaks kode (Fried dkk., 2022a). Berdasarkan (Chatzikoumi, 2020; Lee dkk., 2023) skor BLEU dapt dikelompokan menjadi 7 kategori penilaian. Pada Tabel 3.2 menampilkan 7 kategori penilaian berdasarkan skor BLEU dan CodeBLEU yang telah disesuaikan dengan objek penelitian ini.

Tabel 3.2
Interpretasi Skor BLEU dan CodeBLEU

Skor BLEU & CodeBLEU	Kategori
<10	Menghasilkan kode yang tidak bisa digunakan
10 – 19	Menghasilkan kode yang tidak dapat dibaca
20 – 29	Kode dapat dibaca, akan tetapi masih terdapat kesalahan pada penulisan kode
30 - 40	Menghasilkan kualitas kode cukup baik dan kode dapat dipahami
40 - 50	Menghasilkan kualitas kode baik dan kode dapat dipahami
50 - 60	Penulisan kualitas kode sangat baik dan kode dapat dipahami
> 60	Kualitas penulisan kode lebih baik daripada kualitas kode manusia

3.2 Instrumen Penelitian

Instrumen Penelitian yang terlibat pada penelitian perbandingan model berbasis *Llama2* diantaranya adalah aturan-aturan yang terdaftar pada analisis statis *ESLint*, perangkat keras yang digunakan untuk melatih model dan *library* penting pada *python* untuk pembuatan model bahasa besar dengan detail sebagai berikut :

3.2.1 Aturan-aturan Pada *ESLint*

Pada penelitian ini peraturan yang diterapkan berjumlah 52 tipe yang bersumber dari aturan yang telah dibuat oleh ESLint. Aturan-aturan ini berhubungan dengan kemungkinan kesalahan logika dalam penulisan sintaks kode. Pada Tabel 3.3 menampilkan daftar aturan dan deskripsi yang terdaftar pada ESLint dan digunakan pada pelatihan model. Aturan lainnya ditampilkan pada **Lampiran 2**.

Tabel 3.3
Daftar Aturan pada *ESlint*

Tipe Kesalahan	Deskripsi
no-invalid-this	Mencegah penggunaan `this` di luar dari konteks yang benar dalam JavaScript. Hal ini membantu menghindari kesalahan penggunaan `this` yang dapat mengakibatkan perilaku yang tidak diinginkan atau bug dalam kode. Penggunaan `this` harus terbatas pada metode objek atau fungsi konstruktor di mana objek tersebut dibuat atau digunakan dengan benar.
no-throw-literal	Mencegah melemparkan literal primitif seperti string, angka, boolean, atau objek tanpa ekspresi yang memvalidasi atau menghasilkan error. Ini direkomendasikan untuk memastikan bahwa saat melempar pengecualian (throwing an exception), kode benar-benar memberikan informasi yang berguna tentang alasan kesalahan yang terjadi, daripada hanya melempar nilai primitif yang umumnya tidak memberikan konteks yang cukup.

guard-for-in	Memastikan bahwa loop `for...in` pada JavaScript dilengkapi dengan pernyataan `hasOwnProperty()` untuk memfilter properti-properti yang hanya dimiliki oleh objek itu sendiri, bukan turunan dari prototip objeknya. Hal ini membantu mencegah iterasi yang tidak diinginkan ke properti-properti dari prototip yang dapat menyebabkan masalah dalam logika program, seperti memproses properti yang tidak diharapkan.

3.2.2 Perangkat Keras

Pada penelitian ini perangkat keras digunakan pada tahap pengembangan dan pelatihan model dengan spesifikasi sebagai berikut:

Tabel 3.4

Spesifikasi Perangkat Keras

Spesifikasi	Deskripsi
CPU	24vCPU Cores@MD EPYC 7V12 2.2 GHz
RAM	220GB
Disk	120GB SSD & 960GB SSD NVME
Sistem Operasi	Windows 11 Pro
GPU	Nvidia Tesla A100 80GB VRAM PCIE

3.2.3 Library Python

Penggunaan *Library Python* dimanfaatkan untuk mendukung dalam tahap pengembangan model agar lebih efektif dan efisien. *Library Python* digunakan pada saat pembuatan model, pelatihan model dan evaluasi model. Berikut Daftar *Library Python* yang digunakan pada penelitian ini :

Tabel 3.5

Daftar *Library Python*

Nama *Library*	Versi	Deskripsi Kegunaan
torch	2.2.2	Digunakan untuk memanggil fungsi pada f*ramework* model *Pytorch* yang merupakan alat untuk membangun, melatih, dan menganalisis model *machine learning* dan *deep learning*
transformers	4.39.3	Digunakan untuk mengimplementasikan dan menggunakan model-model bahasa besar berbasis transformer dalam pemrosesan bahasa alami.
peft	0.10.0	Menerapkan metode *Parameter Efficient Fine-Tuning (PEFT)* pada *Large Language Model* (LLM) yang akan melakukan pelatihan. Selain itu library digunakan untuk mengkonfigurasi QLoRA pada model sebelum pelatihan dimulai.
accelerate	0.28.0	Digunakan untuk mempermudah dalam melakukan konfigurasi model pada Multi-GPU, TPU, dan FP16.

3.3 Analisis Data

Pada Analisis data dihasilkan dari evaluasi model yang telah dilakukan sebelumnya. Langkah-langkah dalam proses analisis terdiri dari mereview kembali permasalahan yang telah dirumuskan serta literatur dan teori yang relevan dalam domain machine learning, menentukan metode yang paling cocok berdasarkan karakteristik dataset yang ada, dan akhirnya, mengevaluasi hasil penelitian dan menyimpulkan temuan yang telah ditemukan dalam penelitian ini.