TUGAS AKHIR – ER234403

KLASIFIKASI PERAN KOMPONEN ARSITEKTUR PERANGKAT LUNAK MENGGUNAKAN MODEL CODEBERT

MUHAMMAD BUDHI SALMANJANNAH

NRP 5025201084

Dosen Pembimbing

Rizky Januar Akbar, S.Kom., M.Eng.

NIP 198701032014041001

Program Studi Sarjana Teknik Informatika

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2026

TUGAS AKHIR – ER234403

KLASIFIKASI PERAN KOMPONEN ARSITEKTUR PERANGKAT LUNAK MENGGUNAKAN MODEL CODEBERT

MUHAMMAD BUDHI SALMANJANNAH

NRP 5025201084

Dosen Pembimbing

Rizky Januar Akbar, S.Kom., M.Eng.

NIP 198701032014041001

Program Studi Sarjana Teknik Informatika

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2026

FINAL PROJECT – ER234403

SOFTWARE ARCHITECTURE COMPONENT ROLE CLASSIFICATION USING CODEBERT MODEL

MUHAMMAD BUDHI SALMANJANNAH

NRP 5025201084

Advisor

Rizky Januar Akbar, S.Kom., M.Eng.

NIP 198701032014041001

Study Program Informatics Engineering Bachelor

Department of Informatics Engineering

Faculty of Intelligent Electrical and Informatics Technology

Institut Teknologi Sepuluh Nopember

Surabaya

2026

LEMBAR PENGESAHAN

KLASIFIKASI PERAN KOMPONEN ARSITEKTUR PERANGKAT LUNAK MENGGUNAKAN MODEL CODEBERT

TUGAS AKHIR

Diajukan untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Teknik pada
Program Studi S-1 Sarjana Teknik Informatika
Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh : MUHAMMAD BUDHI SALMANJANNAH
NRP. 5025201084

Disetujui oleh Tim Penguji Tugas Akhir :

1. Rizky Januar Akbar, S.Kom., M.Eng. Pembimbing 
2. Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. Penguji 
3. Bintang Nuralamsyah, S.Kom., M.Kom. Penguji 

SURABAYA

Juni, 2026

[Halaman ini sengaja dikosongkan]

PERNYATAAN ORISINALITAS

Yang bertanda tangan di bawah ini:

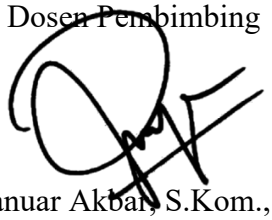
Nama mahasiswa/NRP	:	Muhammad Budhi Salmanjannah / 5025201084
Program studi	:	S-1 Sarjana Teknik Informatika
Dosen Pembimbing/NIP	:	Rizky Januar Akbar, S.Kom., M.Eng. / 198701032014041001

dengan ini menyatakan bahwa Tugas Akhir dengan judul "KLASIFIKASI PERAN KOMPONEN ARSITEKTUR PERANGKAT LUNAK MENGGUNAKAN MODEL CODEBERT" adalah hasil karya sendiri, bersifat orisinal, dan ditulis dengan mengikuti kaidah penulisan ilmiah.

Bilamana di kemudian hari ditemukan ketidaksesuaian dengan pernyataan ini, maka saya bersedia menerima sanksi sesuai dengan ketentuan yang berlaku di Institut Teknologi Sepuluh Nopember.

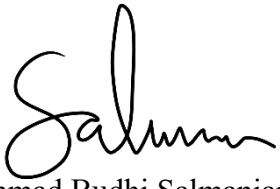
Surabaya, 15 Juni 2026

Mengetahui
Dosen Pembimbing



Rizky Januar Akbar, S.Kom., M.Eng.
NIP. 198701032014041001

Mahasiswa



Muhammad Budhi Salmanjannah
NRP. 5025201084

[Halaman ini sengaja dikosongkan]

ABSTRAK

KLASIFIKASI PERAN KOMPONEN ARSITEKTUR PERANGKAT LUNAK MENGGUNAKAN MODEL CODEBERT

Nama Mahasiswa / NRP	**:**	**Muhammad Budhi Salmanjannah / 5025201084**
Departemen	**:**	**Teknik Informatika FTEIC - ITS**
Dosen Pembimbing	**:**	**Rizky Januar Akbar, S.Kom., M.Eng.**

Abstrak

Clean Architecture (CA) merupakan gaya arsitektural yang memisahkan logika bisnis dari detail teknis untuk meningkatkan pemeliharaan (*maintainability*). Namun, modernisasi perangkat lunak *legacy* menjadi CA melalui migrasi manual memerlukan waktu dan usaha yang signifikan. Penelitian ini bertujuan membangun model otomatis untuk mengklasifikasikan 10 peran komponen arsitektur (seperti *Entity, Repository, Controller*, dll.) menggunakan model pembelajaran representasi kode CodeBERT. Eksperimen membandingkan dua skenario: model CodeBERT Standar dan EL-CodeBERT yang menggunakan mekanisme *Weighted Aggregated Embedding* (WAE) untuk mengakses informasi dari seluruh 12 lapisan transformer. Dataset yang digunakan terdiri dari 1525 sampel kode Java dari 19 proyek *open-source*. Hasil penelitian menunjukkan bahwa model CodeBERT Standar mencapai performa optimal dengan akurasi 72,20% dan *F1-Score* 0,72, mengungguli EL-CodeBERT yang memperoleh akurasi 65,81%. Model *pre-trained* tanpa *fine-tuning* mengalami *Output Degeneracy* dengan akurasi hanya 13,10%. Pada pengujian sistem *legacy*, model standar menunjukkan generalisasi yang baik dengan akurasi 64,29%. Performa EL-CodeBERT yang lebih rendah disebabkan oleh keterbatasan data (*Data Scarcity*) dan *noise* (bising) sintaksis dari lapisan awal yang mengaburkan fitur semantik (*Feature Dilution*). Kesimpulan penelitian ini menunjukkan bahwa pada dataset terbatas, model yang lebih sederhana (CodeBERT Standar) lebih kokoh sesuai prinsip *Occam's Razor*.

Kata kunci: ***Clean Architecture, Machine Learning, CodeBERT, Java, Klasifikasi Peran***

[Halaman ini sengaja dikosongkan]

ABSTRACT

SOFTWARE ARCHITECTURE COMPONENT ROLE CLASSIFICATION USING CODEBERT MODEL

Student Name / NRP	**:**	**Muhammad Budhi Salmanjannah / 5025201084**
Department	**:**	**Teknik Informatika FTEIC - ITS**
Advisor	**:**	**Rizky Januar Akbar, S.Kom., M.Eng.s**

Abstract

Clean Architecture (CA) is an architectural style that separates business logic from technical details to enhance software maintainability. However, modernizing legacy software into CA through manual migration requires significant time and effort. This research aims to build an automated model to classify 10 architectural component roles (e.g., Entity, Repository, Controller, etc.) using the CodeBERT code representation learning model. The experiment compares two scenarios: the Standard CodeBERT model and EL-CodeBERT, which employs a Weighted Aggregated Embedding (WAE) mechanism to access information from all 12 transformer layers. The dataset consists of 1,525 Java code samples from 19 open-source projects. The results show that the Standard CodeBERT model achieved optimal performance with 72.20% accuracy and a 0.72 F1-Score, outperforming EL-CodeBERT, which obtained 65.81% accuracy. Pre-trained models without fine-tuning suffered from Output Degeneracy, with only 13.10% accuracy. In legacy system testing, the standard model demonstrated good generalization with 64.29% accuracy. The lower performance of EL-CodeBERT was attributed to Data Scarcity and syntactic noise from early layers that obscured semantic features (Feature Dilution). This study concludes that on limited datasets, simpler models (Standard CodeBERT) are more robust, consistent with the Occam's Razor principle.

Keywords: ***Clean Architecture, Machine Learning, CodeBERT, Java, Role Classification***.

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT karena atas rahmat, hidayah, dan karunia-Nya, penulis dapat menyelesaikan laporan Tugas Akhir ini dengan baik. Tugas Akhir yang berjudul "KLASIFIKASI PERAN KOMPONEN ARSITEKTUR PERANGKAT LUNAK MENGGUNAKAN MODEL CODEBERT" ini disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer (S.Kom.) pada Departemen Teknik Informatika, Fakultas Teknologi Elektro dan Informatika Cerdas (FTEIC), Institut Teknologi Sepuluh Nopember (ITS).

Dalam proses pelaksanaan hingga penyusunan laporan Tugas Akhir ini, penulis mendapatkan banyak bimbingan, dukungan, serta bantuan dari berbagai pihak. Oleh karena itu, pada kesempatan ini penulis ingin menyampaikan rasa terima kasih yang sebesar-besarnya kepada:

1. Kepala Departemen Teknik Informatika dan Dosen Wali, Bapak Prof. Ir. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D, IPM., atas arahan dan dukungan selama masa perkuliahan dan dalam penulisan Tugas Akhir.
2. Dosen Pembimbing Penulisan Tugas Akhir, Bapak Rizky Januar Akbar, S.Kom., M.Eng. atas semua arahan, masukan, saran, dan bimbingan yang diberikan selama penulisan Tugas Akhir berlangsung hingga dapat selesai.
3. Seluruh dosen dan staf pengajar di Departemen Teknik Informatika ITS yang telah membekali penulis dengan ilmu pengetahuan selama masa perkuliahan.
4. Kedua orang tua dan keluarga penulis. Terima kasih atas seluruh doa, dukungan, semangat, dan nasihat yang tiada henti diberikan kepada penulis. Terima kasih telah menjadi pemacu dan pemberi motivasi penulis.
5. Pasangan penulis, Atalia Kirei Annisa. Terima kasih telah mendorong penulis menjadi lebih baik, membantu penulis berkembang, dan menjadi sumber motivasi dan inspirasi penulis. Terima kasih atas doa, dukungan, pengertian, dan kesabarannya dengan penulis selama penulisan Tugas Akhir.
6. Kawan SMA "Kakaroo", Alief Kukuh, Alief Aditya, Kevin Oktoaria, Belva Aprilliano, dan Amsyar Raftan. Terima kasih atas bantuan, dorongan, nasihat, dan pertemanan selama masa SMA hingga kuliah penulis.
7. Kawan ITS Muaythai Association, Miko, Richie, Mas Fathur, Mas Mike, Purnomo, Gabby, Aaron, Krisna, Ais, Zefanya, Mima, Aryan, Hanna, dan teman-teman saya yang lainnya yang belum sempat saya sebut. Terima kasih atas nasihat, dukungan, dan doa yang diberikan.
8. Rekan-rekan Departemen Teknik Informatika serta seluruh pihak yang tidak dapat disebutkan satu per satu. Terima kasih atas bantuan, diskusi, dan kebersamaannya selama ini.

Penulis menyadari bahwa laporan Tugas Akhir ini masih jauh dari kesempurnaan karena keterbatasan pengetahuan dan pengalaman yang dimiliki. Oleh karena itu, penulis sangat mengharapkan kritik dan saran yang membangun dari pembaca demi perbaikan di masa mendatang.

Surabaya, Juni 2026
Penulis

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Clean Architecture (CA) adalah gaya arsitektural yang membagi kode menjadi beberapa lapisan yang terstruktur. Tujuan dari penerapan CA adalah untuk meningkatkan pemeliharaan (*maintainability*) dan meningkatkan produktivitas selagi mengurangi atau mempertahankan sumber daya yang dibutuhkan dari sebuah perangkat lunak (Martin, 2019).

Berdasarkan Robert C. Martin (2019), domain merujuk pada inti dari aplikasi yang mencakup logika bisnis dan peraturan yang tidak dipengaruhi oleh kerangka eksternal. Infrastruktur sebuah perangkat lunak mengikuti peraturan dan logika bisnis yang ada di dalam domain. Oleh karena itu, penting bagi developer untuk mengerti perbedaan antara domain dan infrastruktur suatu perangkat lunak sebelum mulai mengembangkan perangkat lunak tersebut.

Perangkat lunak *legacy* mengacu pada perangkat lunak yang masih menggunakan sistem yang sudah ketinggalan zaman namun masih digunakan oleh berbagai macam bisnis. Dari sekian banyak perangkat lunak yang ada, masih banyak yang memiliki arsitektur seperti *big ball of mud* dan *fat controller*. Untuk melakukan *maintenance* maupun menambahkan fitur baru pada perangkat lunak tersebut menjadi lebih sulit. Oleh karena itu, perpindahan atau migrasi desain struktur suatu perangkat lunak dapat dilakukan untuk meningkatkan *maintainability* dan mengurangi biaya dalam jangka waktu panjang.

Migrasi struktur perangkat lunak perlu usaha manual oleh developer untuk memelajari *source code* lama dan menulis kode baru (*code rewrite*) menggunakan gaya arsitektur CA secara manual. Oleh karena itu, dalam tugas akhir ini berusaha memecahkan salah satu *task* dalam migrasi. Task tersebut adalah mengidentifikasi potongan kode (*code snippet*) pada *source code legacy* dan mengklasifikasikannya ke dalam peran komponen CA seperti *repository*, *service*, *command*, *query*, *entity,* dan sebagainya untuk mempermudah *developer* melakukan proses migrasi. Identifikasi secara manual merupakan hal yang tidak mudah dan membutuhkan banyak sumber daya. Oleh karena itu, tugas akhir ini memanfaatkan *Machine Learning* (ML) untuk melakukan identifikasi peran tersebut.

CodeBERT merupakan sebuah model ML yang dapat meringkas kode dan melakukan klasifikasi kode (Feng et al., 2020). CodeBERT bekerja dengan cara mengubah kode-kode menjadi token, yang pada akhirnya akan melewati 12 lapisan. Vektor yang didapat pada lapisan terakhir mengandung informasi penting terkait kode tersebut termasuk hubungan semantik dan kegunaan kode tersebut. Fitur klasifikasi kode inilah yang dimanfaatkan dalam mengidentifikasi dan mengklasifikasikan suatu kode ke dalam peran komponen CA dengan cara melakukan *fine-tuning*.

Tujuan dari penggunaan CodeBERT pada tugas akhir ini adalah untuk melatih dan melakukan *fine-tuning* model CodeBERT agar dapat melakukan klasifikasi kode berdasarkan peran komponen CA.

1.2 Rumusan Masalah

Rumusan masalah yang dibahas dalam tugas akhir ini adalah:

1. Bagaimana membangun model klasifikasi peran komponen arsitektur perangkat lunak berbasis model CodeBERT?
2. Bagaimana akurasi model dalam mengklasifikasikan bagian kode pada *source code legacy* yang menggunakan gaya arsitektur non CA?

1.3 Batasan Masalah

Batasan masalah yang ada di tugas akhir ini adalah

1. Dataset latih (*training*) dan uji diambil dari proyek-proyek *open source* yang telah menerapkan CA.
2. Dataset inferensi diambil dari proyek *legacy open source*.
3. Fokus training dan inferensi hanya pada melakukan klasifikasi komponen.
4. Fokus klasifikasi pada komponen yang ada pada gaya arsitektural CA: *Entity*, *Value Object*, *Repository*, *Domain Service*, *Application Service*, *Infrastructure*, dan *Controller*.
5. Model yang dibangun berbasis model CodeBERT.
6. Dataset menggunakan bahasa pemrograman Java.

1.4 Tujuan

Tugas akhir ini bertujuan untuk mengembangkan model klasifikasi peran komponen arsitektur berbasis pembelajaran representasi kode dan dapat menerapkan model klasifikasi peran komponen arsitektur untuk mengidentifikasi dan memberi label bagian-bagian kode dari proyek *legacy*.

1.5 Manfaat

Manfaat dari penelitian tugas akhir ini adalah menghasilkan metode berbasis AI untuk membantu migrasi sistem *legacy* menjadi lebih terstruktur dengan gaya arsitektur CA. Tugas akhir ini juga dapat menjadi landasan untuk penelitian selanjutnya terkait rekomendasi migrasi gaya arsitektur perangkat lunak secara otomatis.

BAB 2 TINJAUAN PUSTAKA

2.1 Hasil Penelitian Terdahulu

2.1.1 Pattern Detection

Berdasarkan dari penelitian yang dilakukan oleh Allamanis pada tahun 2018 mengenai *Machine Learning* (ML) *for Big Code* atau pembelajaran mesin untuk kode sumber, penelitian mengenai klasifikasi peran komponen arsitektu masuk dalam bidang penelitian yang dilakukan oleh Allamanis. Hal ini didasari oleh hipotesis kealamian (*Natural Hypothesis*), yang menyatakan bahwa meskipun bahasa pemrograman bersifat formal dan kaku, kode yang ditulis oleh manusia sebenarnya merupakan bentuk komunikasi yang memiliki pola statistik berulang, mirip dengan bahasa alami (Allamanis et al., 2018). Pengembang perangkat lunak cenderung menulis kode yang mempermudah pemeliharaan perangkat lunak, sehingga ada sebuah keteraturan yang dapat diprediksi oleh model probabilitas (Allamanis et al., 2018).

Dalam sistematika pembelajaran mesin untuk kode, tugas untuk mengenali peran komponen arsitektur secara otomatis dikategorikan dalam Model Penambangan Pola (*Pattern Mining Model*) (Allamanis et al., 2018). Model ini bertujuan untuk menyimpulkan peran fungsional di dalam kode sumber dengan mempelajari keteraturan statistik dari ribuan proyek perangkat lunak yang sudah ada.

2.1.2 Code2Vec

Code2Vec merupakan *neural* model dalam pembelajaran representasi kode (*code representation learning*) yang memperkenalkan metode untuk merepresentasikan potongan kode sebagai vektor (*embedding*) (Alon et al., 2019). Tujuan utama dari model ini adalah memetakan kode sumber ke dalam ruang vektor sedemikian rupa sehingga potongan kode yang memiliki kemiripan semantik akan berada pada posisi yang berdekatan dalam ruang tersebut.

Code2Vec tidak melakukan pendekatan *Natural Language Processing* (NLP) yang memperlakukan kode sebagai urutan token linier, namun model ini memanfaatkan sifat terstruktur dari bahasa pemrograman. Model ini membedah *source code* ke dalam bentuk *Abstract Syntax Tree* (AST) dan mengekstraksi jalur-jalur sintaksis yang menghubungkan antar *terminal node* (nilai atau token) dalam pohon tersebut. Code2Vec menggunakan mekanisme *soft-attention* untuk mempelajari tingkat kepentingan relatif dari setiap jalur dalam sebuah potongan kode. Model secara otomatis memberikan bobot lebih tinggi pada jalur yang paling relevan untuk memprediksi properti semantik, seperti nama sebuah metode atau fungsi (Alon et al., 2019).

Dalam konteks penelitian ini, Code2Vec menjadi pembanding bagi model berbasis Transformer seperti model yang digunakan dalam penelitian ini, CodeBERT. Code2Vec sangat bergantung pada AST untuk menangkap hubungan antar elemen kode, sedangkan CodeBERT memproses kode sebagai urutan token linier dan menangkap hubungan antar elemen melalui mekanisme *self-attention* pada arsitektur Transformer bidirectional.

2.1.3 CodeBERT

CodeBERT merupakan sebuah bi-modal model yang menggunakan *Natural Language* (NL) dan *Programming Language* (PL) (Feng et al., 2020). Model ini didesain untuk melakukan *task* seperti pencarian kode, peringkasan kode dan klasifikasi. Model ini dapat menghasilkan representasi kontekstual yang mampu menangkap hubungan semantik mendalam antara deskripsi teks (seperti dokumentasi atau komentar) dan implementasi kode sumber.

CodeBERT mengadopsi struktur Transformer bidirectional multi-lapis yang serupa dengan konfigurasi RoBERTa-base. Model ini terdiri dari 12 lapisan blok transformer, di mana setiap lapisannya memiliki 12 *attention heads* dan dimensi *hidden state* sebesar 768 (Feng et al., 2020).

Input, yang berupa NL dan PL, dimasukkan ke dalam model sebagai satu urutan linier dengan format sebagai berikut: CLS yang merupakan representasi vektornya pada lapisan akhir digunakan sebagai "ringkasan" dari seluruh *input* untuk tugas klasifikasi, urutan kata NL, SEP yang merupakan token pemisah antara segmen teks penjelasan (NL) dan segmen kode sumber (PL), urutan token PL, dan EOS yang merupakan penanda akhir dari seluruh rangkaian *input*. Teks NL diproses menggunakan algoritma WordPiece dan PL dipecah menjadi urutan token berdasarkan sintaksis bahasa pemrograman Java.

Output yang dihasilkan oleh CodeBERT dapat dibagi menjadi dua jenis utama, yaitu *Contextual Token Embeddings* dan Vektor Representasi Agregat (CLS). Model menghasilkan vektor representasi kontekstual yang menangkap hubungan semantik antara satu token dengan token lainnya dalam satu urutan kode (*Contextual Token Embeddings*). Representasi tersembunyi (*hidden representation*) dari token CLS dianggap sebagai kumpulan informasi semantik fungsional dari seluruh potongan kode yang dimasukkan. Model hanya mengambil vektor [CLS] dari lapisan ke-12 (lapisan terakhir) untuk diteruskan ke bagian klasifikasi (*classifier head*). Setelah model mengeluarkan vektor representasi, vektor tersebut masuk ke lapisan *linear* yang memetakan fitur-fitur abstrak menjadi nilai-nilai numerik. Model kemudian menghasilkan nilai mentah yang disebut *logits* untuk masing-masing dari 10 peran komponen arsitektur, dan akan menggunakan fungsi *argmax* untuk mengambil indeks dengan nilai *logit* tertinggi sebagai prediksi final.

2.1.4 EL-CodeBERT

EL-CodeBERT menggunakan model CodeBERT dan meningkatkan kemampuan dari model tersebut. Arsitektur yang digunakan masih sama dengan CodeBERT, yaitu Transformer bidirectional multi-lapis, namun ada peningkatan kemampuan CodeBERT untuk menangkap semua informasi yang didapat dari lapisan *transformer* (Liu et al., 2022). CodeBERT hanya mengambil informasi dan kesimpulan berdasarkan lapisan ke-12 (Feng et al., 2020), sehingga informasi yang didapat pada lapisan awal bisa hilang setelah melewati lapisan-lapisan pada tahapan selanjutnya. EL-CodeBERT menggunakan informasi yang didapat dari setiap lapisan yang mulai dari fitur permukaan (*surface*) di lapisan bawah, fitur sintaksis di lapisan menengah, dan fitur semantik di lapisan atas (Liu et al., 2022).

Titik perbedaan utama antara keduanya terletak pada fase ekstraksi representasi vektor untuk kebutuhan klasifikasi. EL-CodeBERT melakukan intervensi dengan mengakses seluruh *hidden states* dari lapisan 0 hingga 12 secara bersamaan. Dalam penelitian ini, modifikasi tersebut diimplementasikan melalui mekanisme *Weighted Aggregated Embedding* (WAE). WAE memperkenalkan parameter tambahan berupa *layer_weights*, yang mengajarkan model

untuk memberikan bobot kontribusi numerik pada setiap lapisan sebelum digabungkan menggunakan operasi *weighted sum*. Adaptasi ini digunakan dalam penelitian ini untuk memastikan model tetap mampu mempelajari signifikansi tiap lapisan meskipun bekerja dengan volume dataset yang terbatas.

2.2 Dasar Teori

2.2.1 Arsitektur Transformer

Arsitektur sekuensial seperti R*ecurrent Neural Networks* dan *Long Short-Term Memory* memproses data secara berurutan, yang dapat mengakibatkan *dependencies* jarak jauh susah untuk dimodel. *Transformers* tidak memiliki kekurangan tersebut karena ia memungkinkan model untuk langsung melihat seluruh *input sequence* secara langsung dan dapat menggunakan mekanisme *self-attention* untuk menentukan bagian apa yang paling penting untuk setiap token (Vaswani et al., 2017).

Transformers terdiri atas beberapa lapisan dan tiap lapisannya memiliki komponen-komponen penting. Komponen tersebut adalah *input embedding* dan *encoding* posisional, *multi-head self-attention*, *Feed Forward Network* (FFN), dan *add* dan *norm*.

Cara kerja setiap lapisan adalah *input embedding* & *encoding* posisional akan mengubah setiap token menjadi vektor (*embedding*) dan *embedding* akan diberi *encoding* posisional agar *embedding* tersebut mengerti urutan token dan posisi mereka. Setiap *embedding* token ditransformasi menjadi tiga vektor yaitu *query* (Q), *key* (K) dan *value* (V). *Self-attention* akan menghitung produk dot dari Q dan K untuk menentukan seberapa banyak tiap token harus memperhatikan *token* lainnya; *Multi-head* akan melakukan *self-attention* secara paralel, yang akan dilakukan berulang kali, dengan parameter yang berbeda setiap perulangannya untuk memelajari hubungan antar *token*. Setelah melewati mekanisme *attention*, hasilnya akan melewati dua lapisan FFN.

Tujuan dari lapisan-lapisan ini adalah untuk membantu token memelajari konteks mereka dan menghasilkan vektor yang penuh informasi dari sebuah token atau seluruh inputnya.

2.2.2 Self-Attention dan Multi-Head Attention

Self-attention merupakan sebuah mekanisme yang memungkinkan sebuah token dalam *sequence* "melihat" token lain untuk dalam *sequence* yang sama untuk menentukan seberapa berpengaruhnya mereka pada iterasi selanjutnya.

Setiap token akan ditransformasikan menjadi tiga vektor, yaitu *Query* (Q), *Key* (K) dan *Value* (V). Q akan digunakan untuk dibandingkan dengan K untuk menentukan apa yang perlu dilakukan token tersebut. K digunakan sebagai pembanding Q untuk menentukan relevansi vektor. V merupakan data yang digunakan untuk komputasi representasi token yang terkait di iterasi selanjutnya (Vaswani et al., 2017).

Multi-head attention merupakan sebuah mekanisme yang memanfaatkan sejumlah self-attention yang dijalankan secara paralel, yang tiap mekanismenya memelajari hal yang berbeda dari sequence tersebut. Setiap *self-attention* atau *heads* memiliki perhitungan Q, K, V yang berbeda dari satu sama lain. Penggunaan sejumlah *head* ini bertujuan untuk menangkap informasi yang berbeda dari masing-masing *head* untuk mendapatkan pengertian secara

kontekstual yang dalam (Vaswani et al., 2017). Karena itu *multi-head attention* memungkinkan model menangkap informasi dari ruang representasi yang berbeda secara bersamaan.

2.2.3 Model CodeBERT

CodeBERT mengadopsi arsitektur *Transformer encoder* yang identik dengan RoBERTa-base dan dapat memproses *Natural Language* (NL) dan *Programming Language* (PL) (Feng et al., 2020). Pada awal menjalankan modelnya, token CLS akan dibuat, yang tujuannya untuk meringkas *input*. Kemudian *input* akan dipotong menjadi beberapa token (*tokenization*). Token-token tersebut akan melewati 12 lapisan *transformer*. Jika sudah melewati semua lapisannya, setiap token akan mendapatkan *embedding* kontekstual yang menangkap maksud token itu dalam konteksnya dan vektor CLS akan digunakan untuk meringkas inputnya untuk digunakan dalam *task* seperti klasifikasi, *code matching* dan seterusnya.

2.2.4 Occam's Razor

Occam's Razor merupakan sebuah prinsip heuristik dasar dalam sains yang menyatakan bahwa di antara beberapa teori atau model yang memiliki kemampuan penjelasan yang setara, model dengan penjelasan yang paling sederhana adalah yang harus diutamakan (Sun et al., 2025). Dalam ML, prinsip ini dipelajari secara formal melalui teori *Minimum Description Length* (MDL), yang berargumen bahwa model yang paling ringkas dalam mengomunikasikan atau merangkum data observasi adalah model yang memiliki kemampuan generalisasi terbaik. Model yang terlalu kompleks sering kali cenderung belajar menghafal bising (*noise*) pada data latih alih-alih menangkap pola fungsional yang sebenarnya, konsep ini disebut sebagai *overfitting* (Sun et al., 2025).

Dalam penelitian ini, ada kaitan terhadap prinsip Occam's Razor dengan performa model yang terletak pada aspek kekokohan (*Robustness*). Model dapat disebut sebagai kokoh (*robust*) jika apabila performa prediksinya tetap stabil meskipun dihadapkan pada variasi bising (*noise*), ketidakteraturan struktur data, maupun data yang belum pernah dilihat sebelumnya (Allamanis et al., 2018). Maka dapat diambil kesimpulan bahwa model yang sederhana sesuai prinsip Occam's Razor cenderung lebih kokoh karena memiliki ruang parameter yang lebih ringkas, sehingga lebih sedikit dipengaruhi oleh fluktuasi pada dataset yang terbatas.

Dalam konteks pengujian pada sistem *legacy*, *robustness* menjadi indikator yang penting untuk mengevaluasi apakah model berhasil membangun representasi vektor yang fungsional. Sesuai dengan prinsip Occam's Razor, pada kondisi keterbatasan data, model yang lebih sederhana (CodeBERT) secara teori lebih diunggulkan karena parameter-parameternya lebih stabil dibandingkan model yang lebih kompleks (EL-CodeBERT) yang memerlukan volume data jauh lebih besar untuk menstabilkan parameter bobot antar-lapisannya.

2.2.5 Output Degeneracy dan Representation Collapse

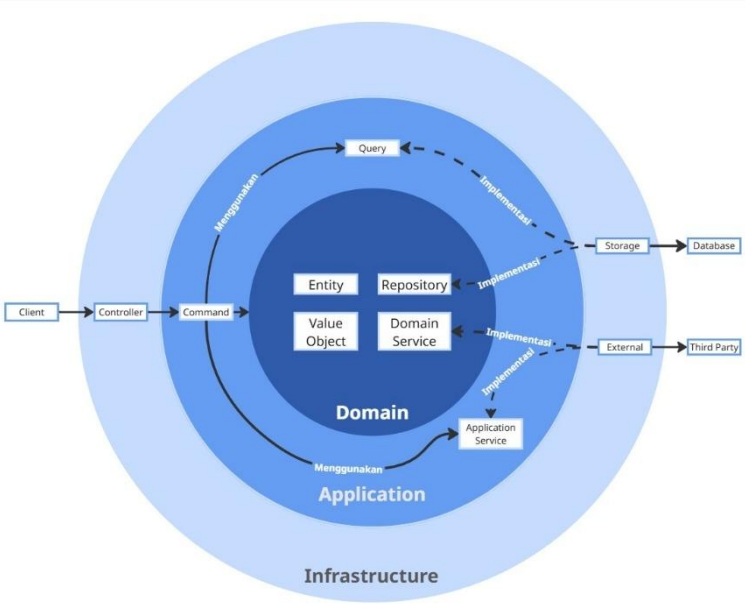
Output Degeneracy ditandai dengan adanya bias pengulangan yang kuat (*strong repetition bias*), pada fenomena tersebut model cenderung menghasilkan keluaran yang seragam, repetitif, atau bias pada pola tertentu terlepas dari variasi input yang diberikan (Gopalani & Hu, 2025). Dalam konteks klasifikasi, hal ini menyebabkan model kehilangan kemampuan klasifikasinya dan mereduksi ruang prediksi menjadi sempit, dan sering kali mengarah pada prediksi kelas mayoritas secara terus-menerus.

Hal ini merupakan manifestasi eksternal dari masalah *Representation Collapse*, yaitu kondisi yang disebabkan oleh keberagaman representasi internal (*hidden states*) dalam model menghilang, sehingga fitur-fitur yang dihasilkan menjadi tidak informatif. Penyebabnya adalah tingkat kemiripan kosinus (*cosine similarity*) yang sangat tinggi antar representasi token di setiap lapisan (Gopalani & Hu, 2025). Kondisi ini menyebabkan geometri representasi di dalam arsitektur Transformer menjadi hancur (*degenerate*), yang akhirnya membatasi kapasitas representasi model untuk membedakan karakteristik unik dari setiap sampel data.

2.2.6 Clean Architecture

Clean Architecture (CA) merupakan sebuah pola desain suatu perangkat lunak yang mempunyai fokus pada konsep *Separation of Concern* yang bermaksud untuk memisahkan logika bisnis dari detail teknis (Martin, 2019). Konsep ini bermaksud untuk membagi sebuah perangkat lunak menjadi beberapa bagian atau lapis untuk mempermudah proses pembangunan perangkat lunak. Setiap bagiannya memiliki atau bobot yang berbeda, dan menangani suatu masalah spesifik.

Lapisan mencakup lapisan inti untuk peraturan dan logika inti bisnis, dan lapisan luar untuk pengguna dan antarmuka sistem. Lapisan inti merupakan peraturan dari sistem tersebut, sedangkan lapisan luar merupakan mekanisme yang menjalankan sistem berdasarkan peraturan dari lapisan inti sistem (Martin, 2019). Lapisan inti ini disebut *domain* dan lapisan luar disebut *infrastructure*.



Gambar 2.1 Diagram Clean Architecture

Diagram di atas dibuat dengan tujuan untuk visualisasi pembagian struktur berdasarkan CA. Tiap lapisan dan komponen dalam lapisan tersebut bertanggung jawab atas hal yang berbeda.

2.2.6.1 Entity

Entity adalah sebuah objek yang didefinisikan oleh *identifier* uniknya yang bertahan selama siklus hidupnya. *Entity* memiliki utilitas atau peraturan yang unik (peraturan bisnis) yang penting terhadap jalannya bisnis di dalamnya. *Entity* bersifat *mutable*, yang artinya atribut-atribut yang ada di dalamnya dapat diubah jika memenuhi peraturan atau setelah melewati proses utilitas yang ada di dalam *Entity* tersebut (Martin, 2019). Kode Sumber 2.1 merupakan contoh kode sumber *Entity*.

```
1.  public class Order {
2.      private final String orderId;
3.      private String status;
4.      private List<OrderItem> items;
5.
6.      public Order(String orderId) {
7.          this.orderId = orderId;
8.          this.status = "PENDING";
9.          this.items = new ArrayList<>();
10.     }
11.     public void confirmPayment() {
12.         if (items.isEmpty()) {
13.             throw new IllegalStateException("Order tidak memiliki item.");
14.         }
15.         this.status = "PAID";
16.     }
17.
18.     public String getOrderId() { return orderId; }
19.     public String getStatus() { return status; }
20. }
```

Kode Sumber 2.1 Contoh Kode Sumber Entity

2.2.6.2 Value Object

Value Object (VO) adalah sebuah komponen yang berfungsi sebagai pendeskripsi, pengukur, atau pemberi kuantitas suatu konsep bisnis, dan tidak memiliki identitas unik. VO didefinisikan oleh atributnya, bersifat *immutable*, dan hanya dapat diganti dengan VO baru yang memiliki *value* yang sudah diperbarui (Vernon, 2013). Data VO yang dimiliki merupakan atribut *Entity*, dan secara umum tersimpan dalam tempat yang sama sehingga VO tidak memerlukan identitas unik dalam suatu sistem. Kode Sumber 2.2 merupakan contoh kode sumber VO.

```
1.  public final class Address {
2.      private final String street;
3.      private final String city;
```

```
4.      private final String zipCode;
5.
6.      public Address(String street, String city, String zipCode) {
7.          if (zipCode == null || zipCode.length() != 5) {
8.              throw new IllegalArgumentException("Zip code tidak valid.");
9.          }
10.         this.street = street;
11.         this.city = city;
12.         this.zipCode = zipCode;
13.     }
14.     public String getStreet() { return street; }
15.     public String getCity() { return city; }
16.     public String getZipCode() { return zipCode; }
17.
18.     @Override
19.     public boolean equals(Object o) {
20.         if (this == o) return true;
21.         if (!(o instanceof Address)) return false;
22.         Address address = (Address) o;
23.         return Objects.equals(street, address.street) &&
24.                 Objects.equals(city, address.city) &&
25.                 Objects.equals(zipCode, address.zipCode);
26.     }
27. }
28.
```

Kode Sumber 2.2 Contoh Kode Sumber Value Object

2.2.6.3 Repository

Repository adalah suatu komponen yang berfungsi sebagai penghubung antara *Storage* dan *Domain*. *Repository* bertanggung jawab atas pendefinisian apa yang dilakukan oleh sistem dengan data yang diambil dari *Database* melalui *Storage* menggunakan bahasa bisnis. Dengan begitu, *Repository* memisahkan logika bisnis dengan detail teknis sehingga mekanisme teknis tersebut terisolasi dari komponen/layer lainnya (Millett, 2015). Kode Sumber 2.3 merupakan contoh kode sumber *Repository*.

```
1.  public interface OrderRepository {
2.      // Menggunakan bahasa bisnis, bukan bahasa SQL [9, 18, 19]
3.      void save(Order order);
4.      Optional<Order> findById(String orderId);
5.      // Spesifik untuk kebutuhan bisnis e-commerce [20, 21]
```

```
6.      List<Order> findAllByCustomerId(String customerId);
7.      void remove(Order order);
8.  }
```

Kode Sumber 2.3 Contoh Kode Sumber Repository

2.2.6.4 Domain Service

Domain Service (DS) adalah komponen operator yang bersifat *stateless* (tanpa status), tidak memiliki identitas, dan bertanggung jawab atas peraturan bisnis yang membutuhkan kooperasi antar komponen *Domain* (Vernon, 2013). DS biasa digunakan jika suatu operasi membutuhkan beberapa *Entity* atau *Agreggate* (grup yang berisi dengan sekumpulan *Entity*) atau jika suatu operasi membutuhkan *input* yang banyak dari komponen domain. Kode Sumber 2.4 merupakan contoh kode sumber DS.

```
1.  public class ShoppingListItemNameComparator implements Comparator<ShoppingListItem> {
2.      @Override
3.      public int compare(ShoppingListItem o1, ShoppingListItem o2) {
4.          return o1.getName().compareTo(o2.getName());
5.      }
6.  }
```

Kode Sumber 2.4 Contoh Kode Sumber Domain Service

2.2.6.5 Application Service

Application Service (AS) adalah komponen operator yang tidak memiliki peraturan bisnis. AS tidak memiliki identitas dan bertanggung jawab mengkoordinasikan jalannya *use case* yang membutuhkan komponen dari lapisan luar hingga *Domain* (Millett, 2015). AS biasa digunakan jika *client* membutuhkan data dari *database*, atau jika *client* menjalankan suatu operasi yang membutuhkan komponen dari *Domain*. Kode Sumber 2.5 merupakan contoh kode sumber AS.

```
1.  class LoadShoppingListUseCase {
2.      private final FirebaseShoppingListRepository repository;
3.
4.      @Inject
5.      LoadShoppingListUseCase(FirebaseShoppingListRepository repository) {
6.          this.repository = repository;
7.      }
8.      Observable<ShoppingList> loadShoppingList(String id) {
9.          ItemByIdSpecification byIdSpecification = new ItemByIdSpecification(id);
10.         return repository.getItem(byIdSpecification);
11.     }
12. }
```

Kode Sumber 2.5 Contoh Kode Sumber Application Service

2.2.6.6 Command

Command adalah komponen yang bertanggung jawab atas instruksi atau niat pengguna dalam menjalankan suatu *use case* yang akan mengubah status suatu sistem (Millet, 2015). *Command* hanya melakukan suatu aksi dan hanya dapat mengembalikan nilai sukses/gagal jika perlu. *Command* dapat dilewati jika *Controller* hanya membutuhkan operasi yang sederhana, sehingga AS bisa langsung digunakan. Kode Sumber 2.6 merupakan contoh kode sumber *Command*.

```
1.  public interface NavigationCommand {
2.    void navigate();
3.  }
```

Kode Sumber 2.6 Contoh Kode Sumber Command

2.2.6.7 Query

Query adalah komponen yang bertanggung jawab atas pengambilan data (Vernon, 2013). *Query* bersifat *read-only* dan tidak mengubah data apapun. Karena *Query* tidak memiliki aturan bisnis yang kompleks, *Query* dapat mengakses *Storage* tanpa harus melewati *Domain*. Data yang dikembalikan *Query* biasanya mengembalikan data struktur yang sederhana seperti *Data Transfer Object* (DTO). Kode Sumber 2.7 merupakan contoh kode sumber *Query*.

```
1.  public class GetHomeTimelineException extends Exception {
2.    public GetHomeTimelineException(){}
3.
4.    public GetHomeTimelineException(Exception reason){
5.      super(reason);
6.    }
7.    public GetHomeTimelineException(String s) {
8.      super(s);
9.    }
10. }
```

Kode Sumber 2.7 Contoh Kode Sumber Query

2.2.6.8 Controller

Controller (HTTP *Controller*) adalah komponen *stateless* yang bertanggung jawab atas mengubah permintaan *client* menjadi data struktur yang sederhana seperti DTO yang dapat diolah oleh AS. Setelah *Controller* mengolah permintaan *client*, *Controller* akan mengarahkan DTO tersebut ke komponen AS yang relevan. Controller juga bertanggung jawab atas menentukan format *output* terhadap client, seperti JSON/HTML ataupun menunjukkan HTTP status *code* yang tepat (Martin, 2019). Kode Sumber 2.8 merupakan contoh kode sumber *Controller*.

```
1.  public class BeakActivityBase extends AppCompatActivity {
2.
```

```
3.    private boolean homeAsBack = true;
4.
5.    @Override public boolean onOptionsItemSelected(MenuItem item) {
6.      if (item.getItemId() == android.R.id.home) {
7.        if (isHomeAsBack()) {
8.          onBackPressed(); // Behave as a back button
9.        }
10.       return true;
11.     }
12.     return super.onOptionsItemSelected(item);
13.   }
14.   public boolean isHomeAsBack() {
15.     return homeAsBack;
16.   }
17.   public void setHomeAsBack(boolean homeAsBack) {
18.     this.homeAsBack = homeAsBack;
19.   }
20. }
```

Kode Sumber 2.8 Contoh Kode Sumber Controller

2.2.6.9 Storage

Storage adalah komponen yang bertanggung jawab atas pengambilan data dari *Database* atas permintaan *Repository* dan mengolah data tersebut menjadi suatu *Entity*. Pada umumnya, *Storage* akan memiliki detail teknis yang tidak akan dilihat oleh komponen lainnya, terutama komponen *Repository* (Martin, 2019). Kode Sumber 2.9 merupakan contoh kode sumber *Storage*.

```
1.  public class StubShoppingListRepository implements Repository<ShoppingList> {
2.
3.      private final Set<ShoppingList> shoppingLists;
4.
5.      public StubShoppingListRepository() {
6.          this.shoppingLists = new HashSet<>();
7.          initializeShoppingLists();
8.      }
9.
10.     @Override
11.     public Single<List<ShoppingList>> getItems(Specification specification) {
12.         //ItemsSpecification itemsSpecification = (ItemsSpecification) specification;
13.         return Single.just(new ArrayList<>(shoppingLists));
```

```
14.     }
15.
16.     @Override
17.     public Single<ShoppingList> add(ShoppingList item) {
18.         shoppingLists.add(item); // add or update given shopping list
19.         return Single.just(item);
20.     }
21.
22.     @Override
23.     public Single<ShoppingList> update(ShoppingList item) {
24.         shoppingLists.add(item); // add or update given shopping list
25.         return Single.just(item);
26.     }
27. }
```

Kode Sumber 2.9 Contoh Kode Sumber Storage

2.2.6.10 External

External adalah komponen yang bertanggung jawab atas menyambungkan aplikasi/bisnis dengan API pihak ketiga (Martin, 2019). Bertugas sebagai *Gateway*, *External* akan mengubah data yang diambil dari API pihak ketiga menjadi data yang sesuai dengan format yang sudah ditentukan oleh aplikasi/bisnis. Kode Sumber 2.10 merupakan contoh kode sumber *External*.

```
1.  public interface TownshipService {
2.      @GET(""bins/yy0wl"")
3.      Observable<List<TownshipEntity>> getTownships();
4.  }
```

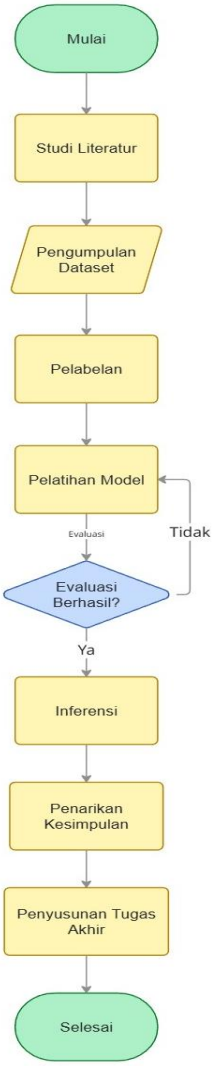
Kode Sumber 2.10 Contoh Kode Sumber External

[Halaman ini sengaja dikosongkan]

BAB 3 METODOLOGI

3.1 Metode Penelitian

Penelitian ini dilakukan dalam 8 tahapan, yaitu Studi Literatur, Pengumpulan Dataset, Pelabelan, Model *Training*, Inferensi, Evaluasi, Penarikan Kesimpulan dan Penyusunan Tugas Akhir. Berikut merupakan *flowchart* tahapan pelaksanaan penelitian:



Gambar 3.1 Flowchart Penelitian

3.1.1 Studi Literatur

Studi literatur dilakukan untuk mencari dan memelajari berbagai macam referensi dan literatur untuk menyelesaikan tugas akhir ini. Literatur dapat berupa jurnal, *conference*, maupun sumber – sumber lainnya. Pada tugas akhir ini akan dibahas hal – hal mengenai :

a) CodeBERT
b) Clean Architecture
c) Self-attention dan Multi-Head
d) Transformers Architecture

3.1.2 Pengumpulan Dataset

Pada tahap ini, peneliti akan mengambil proyek *open source* berbahasa Java yang menggunakan prinsip CA dalam perangkat lunaknya sebagai data latih. Pengumpulan proyek-proyek ini dilakukan di platform Github, dan akan dipilah oleh peneliti untuk memastikan jika sesuai dengan gaya arsitektural CA atau tidak sebelum lanjut ke tahap selanjutnya. Target dataset yang digunakan adalah 20 proyek *open source*.

3.1.3 Pelabelan

Setelah melakukan proses pemilihan dataset, akan dilakukan *labeling* atau pelabelan dengan bantuan *Large Language Model* (LLM) melalui *prompting* yang diakses melalui *OpenRouter* yang kemudian akan ditinjau ulang oleh peneliti. Pelabelan dengan LLM akan dilakukan dengan menggunakan *zero-shot prompting* pada 70% dataset menggunakan model Claude Opus 4.8, dan *few-shot prompting* 30% menggunakan Deepseek V4 Pro.

3.1.3.1 Pemilihan LLM

Pemilihan Claude Opus 4.8 didasari oleh evaluasi LLM yang dilakukan *Artificial Analysis Intelligence Index*, Claude Opus 4.8 mencapai peringkat 2 dalam pengujian penalaran dan pemahaman bahasa dengan nilai sebesar 55,7 dan *Coding Index* sebesar 74,3 (Artificial Analysis, 2026). Claude Opus 4.8 memiliki kapabilitas tinggi dalam menangkap logika semantik suatu *snippet* tanpa diberikan contoh di awal *prompting* (*zero-shot*). Oleh karena itu hasil dari pelabelan yang dilakukan Claude Opus 4.8, setelah melewati proses pemeriksaan secara manual oleh peneliti, akan digunakan sebagai *ground truth* dalam pelabelan menggunakan model LLM lainnya.

Pemilihan Deepseek V4 Pro didasari oleh optimasi sumber daya penelitian, model ini memiliki biaya yang jauh lebih murah dengan performa yang kompetitif dengan nilai *Artificial Analysis Intelligence Index* sebesar 44,3 dan *Coding Index* sebesar 59,4. Berdasarkan pertimbangan efisiensi dan performa tersebut, peneliti memilih model ini untuk melakukan pelabelan terhadap 30% dataset pada penelitian ini, sekaligus memproyeksikannya sebagai instrumen pelabelan data untuk pengembangan penelitian di masa mendatang dengan menggunakan *ground truth* yang sudah ditetapkan sebagai contoh.

Dilakukannya pelabelan ini adalah untuk mengajarkan model yang digunakan untuk mencari kode dengan struktur atau hubungan semantik yang serupa agar pada saat inferensi model ini dapat mencari struktur kode yang serupa dan mengklasifikasikan komponen tersebut sesuai perannya dalam CA.

3.1.3.2 Ekstraksi Data dan Pelabelan

Tahap ini mencakup dua proses yang meliputi ekstraksi *source code* dan transformasi *source code* tersebut menjadi format dataset, dan memberikan label komponen peran CA dengan menggunakan strategi hibrida yaitu pelabelan dengan memanfaatkan AI dan validasi manual.

3.1.3.2.1 Ekstraksi Data

Data mentah yang berupa *source code* dikumpulkan dari repositori GitHub dan diproses menggunakan skrip “data_extractor.py”. Skrip ini bertujuan untuk mengisolasi logika fungsional kode. Skrip ini akan memindai secara otomatis terhadap direktori proyek dan

mengambil file dengan ekstensi *Java*. Kemudian untuk mengurangi kebisingan (*noise*) pada model, akan dihapus deklarasi *package*, pernyataan *import*, serta header lisensi yang umumnya berada di awal file. Terakhir, potongan kode (*snippet*) yang sudah bebas dari *noise* akan dimasukkan ke dalam file CSV (unlabelled.csv) yang memuat informasi nama proyek, jalur file, nama file, dan isi kode.

```
1.  INPUT: Direktori Repositori GitHub (raw_dir)
2.  OUTPUT: File CSV Dataset mentah (unlabelled.csv)
3.
4.  INISIALISASI list data_kosong
5.  UNTUK repositori DALAM raw_dir:
6.  UNTUK file_kode (.java) DALAM repositori:
7.  BACA isi_file
8.  HAPUS komentar lisensi (multiline comments di awal file)
9.  HAPUS deklarasi 'package'
10. HAPUS pernyataan 'import'
11. CARI baris pertama deklarasi (class/interface/enum)
12. POTONG teks mulai dari deklarasi tersebut hingga akhir file
13. SAVE snippet, nama_proyek, dan jalur_file KE data_kosong
14. EXPORT data_kosong KE unlabelled.csv
```

Kode Sumber 3.1 Pseudo-Code Ekstraksi Data

3.1.3.2.2 Pelabelan Hibrida

Tahap selanjutnya adalah pelabelan yang memanfaatkan dua model LLM melalui API OpenRouter dan melakukan validasi label secara manual. Untuk menjaga kualitas dan konsistensi label, strategi yang digunakan adalah pertama untuk menggunakan model Claude Opus 4.8 sebagai pelabel utama dengan menggunakan teknik *zero-shot prompting*. Setelah Claude Opus 4.8 melakukan pelabelan pada 70% dataset, dilakukan validasi oleh peneliti secara manual untuk memastikan akurasi pelabelan komponen, yang kemudian dataset tersebut menjadi *ground-truth* penelitian. Kemudian, dilakukan penggantian model menjadi Deepseek V4 Pro, yang menggunakan teknik *Few-shot prompting*, yang artinya model diberikan definisi dan contoh kode untuk setiap komponen dengan memanfaatkan dataset *ground-truth* yang sudah ditetapkan pada sesi pelabelan sebelumnya. Terakhir, dilakukan validasi ulang oleh peneliti secara manual untuk memastikan akurasi pelabelan komponen.

```
1.  INPUT: unlabelled.csv, Definisi Peran Komponen
2.  OUTPUT: final_dataset.csv (Ground Truth)
3.
4.  // Fase 1: Zero-Shot (70% data)
5.  UNTUK 70% data DALAM unlabelled.csv:
6.    KIRIM snippet ke Claude Opus 4.8 via API
7.    PROMPT: "Klasifikasikan peran kode ini (10 peran) berdasarkan definisi yang diberikan
```

```
    tanpa contoh"
  8.    SIMPAN hasil
  9.  // validasi manual oleh peneliti terhadap hasil_label_sementara
  10. TETAPKAN data tervalidasi sebagai Ground Truth (GT)
  11.
  12. // Fase 2: Few-Shot (30% data sisanya)
  13. UNTUK 30% data sisanya:
  14.   AMBIL contoh snippet dan label dari GT
  15.   KIRIM snippet + contoh GT ke Deepseek V4 Pro
  16.   SIMPAN hasil
  17. // validasi manual oleh peneliti
  18. // gabung manual seluruh data KE final_dataset.csv
```

Kode Sumber 3.2 Pseudo-Code Pelabelan Data

3.1.4 Model Latih

Akan dilakukan *fine-tuning* model klasifikasi CodeBERT dan model klasifikasi CodeBERT berbasis pengimplementasian EL-CodeBERT, untuk mengidentifikasi peran kode dalam konteks CA. Untuk melatih model ini digunakan 80% dataset yang telah diperoleh dari tahap pelabelan menggunakan *Fixed Random Seed,* dalam bentuk *train_test_split* yang merupakan fungsi dari *library* scikit-learn, sebagai pengacak dataset agar pembagian data bersifat deterministik dan data uji tidak terlihat oleh model.

Pada tahap ini, dilakukan adaptasi bobot internal agar mampu mengklasifikasikan 10 peran komponen arsitektur secara mandiri. Proses pelatihan model dilakukan dengan metode *fine-tuning* pada model *pre-trained* microsoft/codebert-base. Untuk Model CodeBERT, konfigurasi dilakukan dengan mengambil vektor representasi dari token [CLS] pada lapisan terakhir (lapisan ke-12). Untuk Model EL-CodeBERT, diterapkan modifikasi arsitektur menggunakan *Weighted Aggregated Embedding* (WAE) yang mengakses seluruh lapisan transformer (0-12). Parameter *layer_weights* diinisialisasi secara acak dan dilatih untuk menentukan kontribusi relatif tiap lapisan.

Pelatihan menggunakan API Trainer dari library HuggingFace dengan parameter: *learning rate* sebesar 5e-5, *warmup steps* sebanyak 50, dan dijalankan selama 5 epoch. Digunakan *Cross-Entropy Loss* sebagai fungsi kerugian untuk meminimalkan selisih antara prediksi model dengan *ground truth* dari dataset latih.

```
  1.  INPUT: final_dataset.csv, Pre-trained Model Weight
  2.  OUTPUT: fine_tuned_weights.bin
  3.
  4.  LOAD final_dataset.csv
  5.  SPLIT data (80% Latih, 20% Uji) dengan Fixed Random Seed
  6.  TOKENISASI snippet menggunakan WordPiece (microsoft/codebert-base)
```

```
7.  IF menggunakan EL-CodeBERT:
8.    AKTIFKAN output_hidden_states=True
9.    INISIALISASI trainable layer_weights (Lapisan 0-12)
10. UNTUK SETIAP epoch (1 s/d 5):
11.   FORWARD PASS:
12.     HITUNG representasi token [CLS]
13.     IF EL: Hitung Weighted Sum dari 13 lapisan [CLS]
14.     ELSE: Ambil [CLS] dari lapisan ke-12 saja
15.   HITUNG Cross-Entropy Loss antara prediksi dan label aktual
16.   BACKPROPAGATION: Update bobot model dan layer_weights
17. SIMPAN model_terbaik (berdasarkan F1-Score)
```

Kode Sumber 3.3 Pseudo-Code Pelatihan Model

3.1.4.1 CodeBERT

Dalam pelatihan model ini, dilakukan konfigurasi untuk hanya mengambil informasi dari lapisan transformer terakhir (lapisan ke-12). Vektor representasi dari token CLS pada lapisan ke-12 digunakan sebagai ringkasan informasi semantik dari seluruh urutan kode Java yang dimasukkan. Vektor tersebut akan diteruskan ke sebuah *classifier head* yang memetakan fitur ke dalam 10 kategori peran arsitektur.

3.1.4.2 EL-CodeBERT

EL-CodeBERT merupakan sebuah modifikasi arsitektur yang dirancang untuk menangkap informasi secara lebih komprehensif dari struktur internal transformer. EL-CodeBERT dikonfigurasi untuk mengakses informasi dari seluruh lapisan transformer (lapisan 0 – 12).

Tahap ini menerapkan implementasi *Weighted Aggregated Embedding* (WAE) berupa *layered_weight*, yang bersifat *trainable* dan berfungsi untuk memberikan bobot kontribusi pada setiap lapisan. WAE merupakan bentuk efisiensi adaptasi implementasi EL-CodeBERT karena implementasi *Attention Mechanism* akan menambahkan banyak parameter baru yang membutuhkan jumlah data yang melewati batasan tugas akhir ini. Implementasi ini didasari oleh teori lapisan awal cenderung menangkap sintaksis kode, sedangkan lapisan akhir menangkap fitur semantik fungsional (Liu et al., 2022).

Setelah itu, token CLS dari setiap lapisan diambil dan digabungkan menggunakan operasi *weighted sum* untuk menghasilkan satu *Aggregate Embedding* yang lebih kaya informasi sebelum masuk ke tahap klasifikasi akhir.

3.1.5 Evaluasi dan Inferensi

Setelah melakukan model training, evaluasi dan inferensi dilakukan menggunakan standar evaluasi pada riset CodeBERT dan EL-CodeBERT yang menggunakan metrik *Accuracy*, *Precision*, *Recall*, dan F1-Score.

Accuracy adalah rasio prediksi yang benar (baik positif maupun negatif) dibandingkan dengan keseluruhan jumlah data (Liu et al., 2022).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.1)$$

Precision adalah rasio prediksi positif yang benar dibandingkan dengan keseluruhan hasil yang diprediksi sebagai positif (Liu et al., 2022).

$$Precision = \frac{TP}{TP + TN} \quad (3.2)$$

Recall adalah rasio prediksi positif yang benar dibandingkan dengan keseluruhan jumlah data yang sebenarnya positif (Liu et al., 2022).

$$Recall = \frac{TP}{TP + FN} \quad (3.3)$$

F1-Score merupakan nilai rata-rata antara *precision* dan *recall* untuk mendapatkan keseimbangan antara keduanya (Liu et al., 2022).

$$F1 = \frac{2 \cdot P \cdot R}{P + R} \quad (3.4)$$

Rumus-rumus tersebut didasari oleh perhitungan TP (*True Positive*) yaitu sampel positif yang diprediksi benar sebagai kelas positif, TN (*True Negative*) yaitu sampel negatif yang diprediksi benar sebagai kelas negatif, FP (*False Positive*) yaitu sampel negatif yang diprediksi salah sebagai kelas positif, FN (*False Negative*) yaitu sampel positif yang diprediksi salah sebagai kelas negatif (Liu et al., 2022).

Evaluasi dan inferensi dilakukan pada empat jenis model, yaitu *Pre-Trained* CodeBERT, *Pre-Trained* EL-CodeBERT, CodeBERT, dan EL-CodeBERT. Varian *Pre-Trained* mengacu pada model yang belum dilatih atau *fine-tune*, sedangkan varian lainnya adalah model yang sudah dilatih atau *fine-tuned* menggunakan dataset pelatihan. Tujuan penggunaan dalam evaluasi dan inferensi varian *Pre-Trained* adalah sebagai titik acuan perbandingan dengan varian *fine-tuned*.

Evaluasi dilakukan pada 20% dataset yang tidak pernah terlihat oleh model berdasarkan *Fixed Random Seed* dalam bentuk *train_test_split* yang merupakan fungsi dari *library* scikit-learn.

Inferensi dilakukan pada proyek *Java open source* untuk mengklasifikasikan peran komponen berdasarkan CA. Jika hasil evaluasi klasifikasi yang didapat masih belum sesuai kebutuhan (F1-Score atau akurasi rendah), maka dilakukan iterasi ulang pada tahap penelitian dengan menambah data, atau menyesuaikan parameter pelatihan dan sebagainya. Inferensi dilakukan untuk menguji kemampuan model dalam mengklasifikasikan data baru (sistem *legacy*) ke dalam 10 peran arsitektur.

Alur teknis inferensi yang diterapkan adalah sistem memuat bobot model hasil *fine-tuning* beserta berkas *label_mapping.json*. Berkas *JSON* ini sangat penting untuk menerjemahkan kembali ID numerik hasil prediksi model menjadi label peran arsitektur. Kemudian potongan kode dari sistem *legacy* dibersihkan dari *noise* dan ditokenisasi menggunakan *AutoTokenizer* agar sesuai dengan format input CodeBERT. Model menghasilkan nilai *logits* untuk masing-masing dari 10 kelas. Prediksi akhir diambil dari indeks dengan nilai *logits* tertinggi

menggunakan fungsi *argmax*. Performa model diukur menggunakan metrik *Accuracy*, *Precision*, *Recall*, dan *Weighted* F1-Score untuk memastikan model tidak hanya akurat secara keseluruhan tetapi juga stabil dalam mengenali kelas-kelas minoritas.

```
1.  LOAD fine_tuned_weights DAN label_mapping.json
2.  TOKENISASI snippet sistem legacy
3.  FORWARD PASS:
4.    AMBIL nilai logits untuk 10 kelas
5.  PREDIKSI = argmax(logits) // Ambil indeks probabilitas tertinggi
6.  LABEL = mapping(PREDIKSI) // Ubah ID numerik ke teks (e.g., "Entity")
7.  HITUNG METRIK (UNTUK SEMUA SAMPEL):
8.    Accuracy = (TP + TN) / Total Data
9.    Weighted F1-Score = 2 * (Prec * Rec) / (Prec + Rec)
10. CETAK Classification Report
```

Kode Sumber 3.4 Pseudo-Code Evaluasi dan Inferensi

3.2 Lingkungan Penelitian

Spesifikasi perangkat keras yang digunakan untuk penelitian ini adalah sebagai berikut:

- CPU : Ryzen 5 2600
- GPU : NVIDIA GTX 1660 SUPER – 6 GB VRAM
- RAM : 2 × 8 GB Colorful BattleAx DDR4-3200

Spesifikasi perangkat lunak yang digunakan untuk penelitian ini adalah sebagai berikut:

- Sistem Operasi : Windows 11
- Bahasa Program : Python 3.13.13
- PyTorch : 2.12.1+cu130
- Transformers : 5.11.0
- Scikit-Learn : 1.9.0
- Pandas : 3.0.3
- OpenAI : 2.41.1
- Python-Dotenv : 1.2.2
- NVIDIA CUDA Toolkit

Spesifikasi CodeBERT yang digunakan untuk penelitian ini adalah sebagai berikut:

- Versi Model : Pre-trained microsoft/codebert-base

[Halaman ini sengaja dikosongkan]

BAB 4 HASIL DAN PEMBAHASAN

4.1 Hasil penelitian

Fokus utama pada subbab ini adalah menyajikan fakta teknis terkait lingkungan penelitian, karakteristik dataset dari 19 repositori Java yang digunakan, nilai *loss* dan akurasi selama proses pelatihan, serta hasil evaluasi performa klasifikasi yang diukur melalui metrik *Precision*, *Recall*, dan *F1-Score*.

4.1.1 Analisis Dataset

Dataset yang digunakan dalam penelitian ini dikumpulkan dari repositori publik di GitHub yang menerapkan prinsip CA pada bahasa pemrograman Java. Proses pengumpulan menghasilkan total 1525 sampel kode (*snippet*) yang telah melalui tahap pelabelan manual dan bantuan LLM.

4.1.1.1 Daftar Repositori

Data diambil dari berbagai domain aplikasi untuk memastikan model memiliki variasi semantik yang luas. Berikut adalah daftar 19 repositori utama yang menjadi sumber data dalam iterasi penelitian ini:

Tabel 4.1 Daftar Repositori Latih dan Uji

No	Nama Repositori	Domain Aplikasi
1	Beak-master	Twitter Client
2	CleanArchitecture-master	Post & Social App Template
3	Earthquakes-master	Earthquake Information
4	flight-booking-clean-architecture	Travel & Booking
5	Kamus-Inggris-Indonesia-master	Education/Dictionary
6	LittleLight-master	Game Companion
7	Mediateka-master	Movie & Actor Information
8	Music-App-master	Music Streaming
9	QuizAlarmClock-master	Productivity/Utility
10	RecipePuppy-master	Food & Cooking
11	MovieGuide-master	Movie Discovery
12	Movies-Finder-master	Search Engine
13	musicweather-api-clean-architecture-master	API Service / Music & Weather Integration
14	Neuronizer-master	Productivity
15	pet-visits-service-main	Veterinary Management
16	protohipster-master	Location Prototype
17	Reedly-master	Information / RSS News Reader
18	RestaurantBooker-master	Reservation System
19	ShoppingList-master	Grocery Shopping List

4.1.1.2 Distribusi Data Berdasarkan Peran Komponen

Penyebaran label dalam dataset sangat memengaruhi kemampuan model CodeBERT dalam mengenali "sinyal" spesifik dari setiap komponen. Dengan memanfaatkan *train_test_split*, dataset terbagi menjadi dua bagian, yaitu 80% sebagai data pelatihan dan 20% sebagai data uji.

Tabel di bawah ini menunjukkan distribusi jumlah data untuk masing-masing dari 10 peran komponen arsitektur:

Tabel 4.2 Distribusi Sampel Berdasarkan Peran Komponen

Komponen	Jumlah Sampel	Persentase (%)
Controller	442	29,0%
External	312	20,4%
Entity	185	12,1%
Value Object	178	11,7%
Application Service	145	9,5%
Storage	112	7,3%
Query	65	4,3%
Repository	48	3,1%
Domain Service	23	1,5%
Command	15	1,0%
Data Latih	1212	80%
Data Uji	313	20%
Total	1525	100%

4.1.1.3 Analisis Karakteristik Data

Sesuai dengan Tabel 4.2, dapat diambil kesimpulan bahwa komponen seperti *Controller* dan *External* memiliki jumlah sampel terbanyak. Hal ini dikarenakan aplikasi Android/Java yang dikumpulkan memiliki banyak interaksi *UI* dan integrasi *API* pihak ketiga. Kedua, terdapat perbedaan jumlah yang signifikan antara komponen *Controller* (442) dengan *Command* (15). Perbedaan yang signifikan ini memungkinkan model untuk menjadi lebih akurat dalam menebak suatu komponen yang memiliki data latih yang lebih banyak dibandingkan dengan komponen yang tidak memiliki banyak data latih, sehingga masalah ini menjadi salah satu tantangan dalam proses *fine-tuning* model.

4.1.2 Hasil Pelatihan Model

Setelah melalui tahap pengolahan data dan pelabelan, tahap selanjutnya yang dilakukan adalah *fine-tuning* model. Model yang dilatih adalah model CodeBERT dan model CodeBERT dengan implementasi EL-CodeBERT.

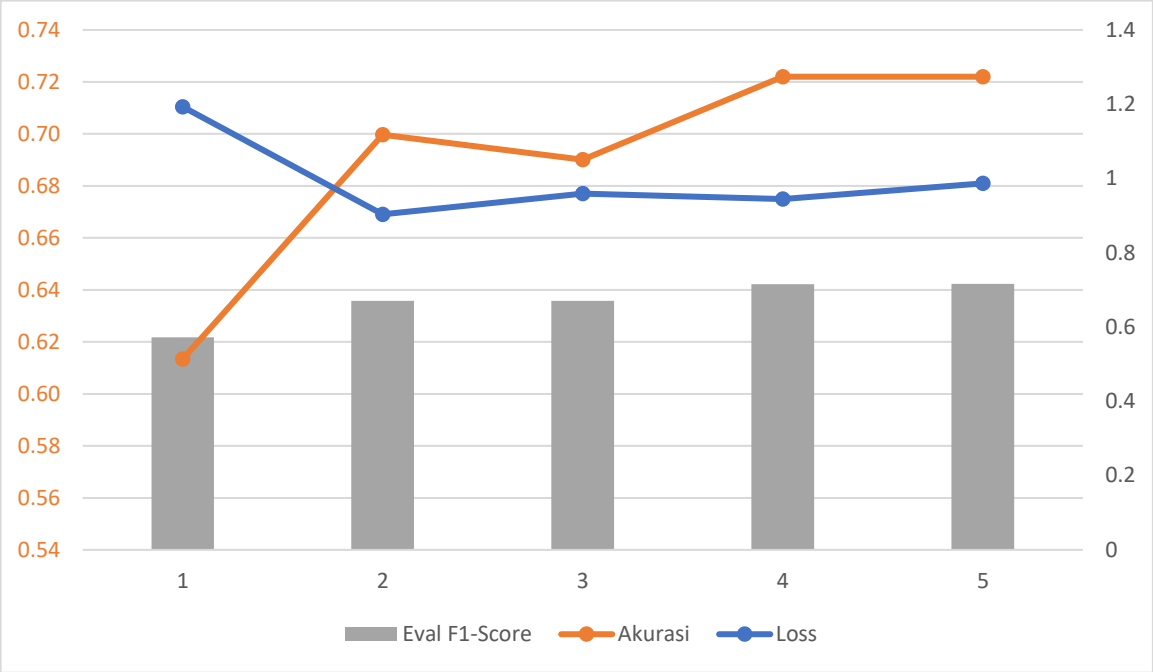
4.1.2.1 Hasil Pelatihan Model Fine-tuned CodeBERT

Pelatihan model CodeBERT memakan waktu sebanyak 14 menit dan menggunakan perangkat GPU. Berikut merupakan tabel *Loss* dan Akurasi dari pelatihan model:

Tabel 4.3 *Loss* dan Akurasi Pelatihan Model CodeBERT

Epoch	Loss	Akurasi	Eval F1-Score
1	1,193	61,34%	0,5719
2	0,9035	69,97%	0,6709
3	0,9594	69,01%	0,6705
4	0,9447	72,20%	0,7151
5	0,9868	72,20%	0,7160

Dan berikut merupakan grafik yang merepresentasikan Tabel 4.3:



Gambar 4.1 Grafik *Loss* dan Akurasi Pelatihan Model CodeBERT

Sesuai dengan grafik dan tabel yang telah disajikan, dapat ditarik kesimpulan bahwa akurasi mengalami peningkatan yang cukup signifikan dari *epoch* pertama menuju *epoch* ke-2, mencapai 8,63% dari nilai akurasi awal. Sedangkan dari *epoch* ke-2 hingga *epoch* ke-5 hanya meningkat dengan total 2,23%. Ini menunjukkan model mencapai titik stabil (*convergence*). Sebaliknya, *loss* berkurang secara signifikan dari *epoch* pertama menuju *epoch* ke-2, yang mencapai 0,2895 atau 24,27% dari nilai *loss* awal. Hal ini mengindikasikan bahwa model CodeBERT mampu mengenali pola dasar komponen arsitektur dengan cepat pada iterasi awal sebelum akhirnya mengalami perlambatan peningkatan performa saat mendekati titik optimal sebesar 72,20% di *epoch* ke-5.

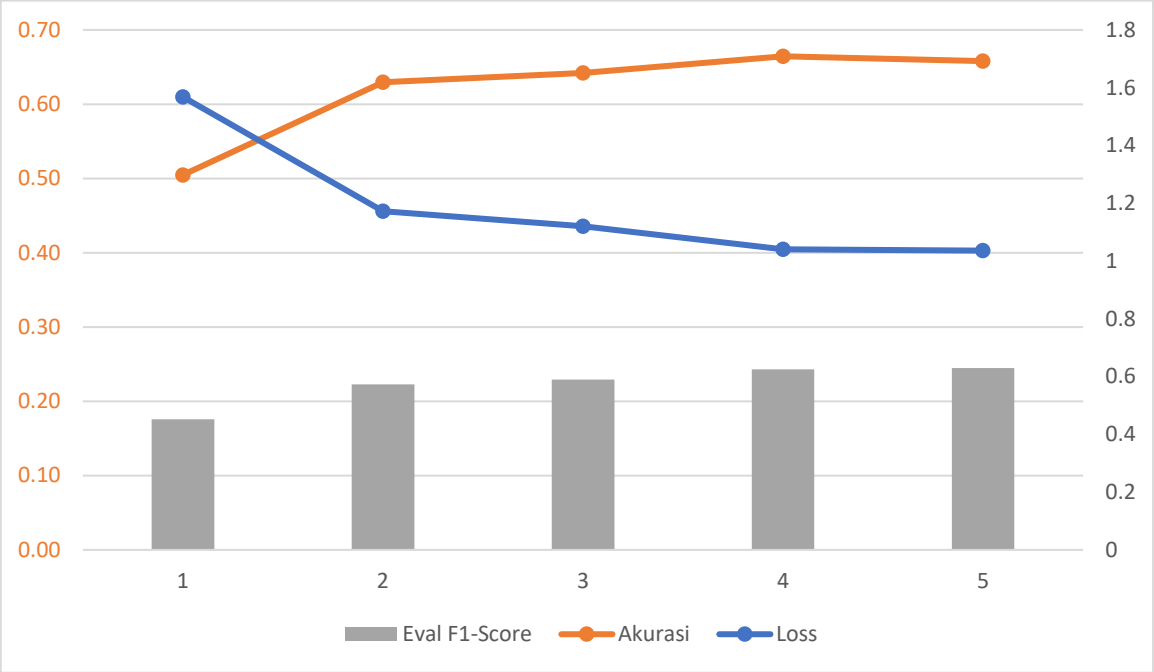
4.1.2.2 Hasil Pelatihan Model Fine-tuned EL-CodeBERT

Pelatihan model CodeBERT memakan waktu sebanyak 16 menit dan menggunakan perangkat GPU. Berikut merupakan tabel *Loss* dan Akurasi dari pelatihan model:

Tabel 4.4 *Loss* dan Akurasi Pelatihan Model EL-CodeBERT

Epoch	Loss	Akurasi	Eval F1-Score
1	1,568	50,48%	0,4524
2	1,172	62,94%	0,5728
3	1,120	64,22%	0,5890
4	1,041	66,45%	0,6246
5	**1,036**	**65,81%**	**0,6297**

Dan berikut merupakan grafik yang merepresentasikan Tabel 4.4:



Gambar 4.2 Grafik *Loss* dan Akurasi Pelatihan Model EL-CodeBERT

Sesuai dengan grafik dan tabel yang telah disajikan, dapat ditarik kesimpulan bahwa akurasi mengalami peningkatan yang cukup signifikan dari *epoch* pertama menuju *epoch* ke-2, mencapai 12,46% dari nilai akurasi awal. Namun dari *epoch* ke-2 hingga *epoch* ke-5 hanya meningkat dengan total 2,87%, yang kemudian pada *epoch* 4 menuju *epoch* 5 turun 0,64%. Hal ini menunjukkan kurva pembelajaran yang lebih kompleks pada arsitektur *Aggregated* Embedding. Sejalan dengan hal tersebut, nilai *loss* berkurang secara konsisten dari *epoch* pertama ke *epoch* ke-2 sebesar 0,396 atau 25,25%. Meskipun menunjukkan progres yang stabil, performa EL-CodeBERT masih berada di bawah model *default*, mengindikasikan bahwa model dengan parameter bobot lapisan (*layer weights*) ini memerlukan volume data yang lebih besar untuk melampaui titik optimal *baseline* saat ini.

4.1.3 Evaluasi Model Klasifikasi

Subbab ini memaparkan evaluasi model CodeBERT *pre-trained*, model CodeBERT *fine-tuning*, dan model CodeBERT *fine-tuning* yang menggunakan pengimplementasian EL-CodeBERT.

4.1.3.1 Evaluasi Model Klasifikasi Pre-Trained CodeBERT

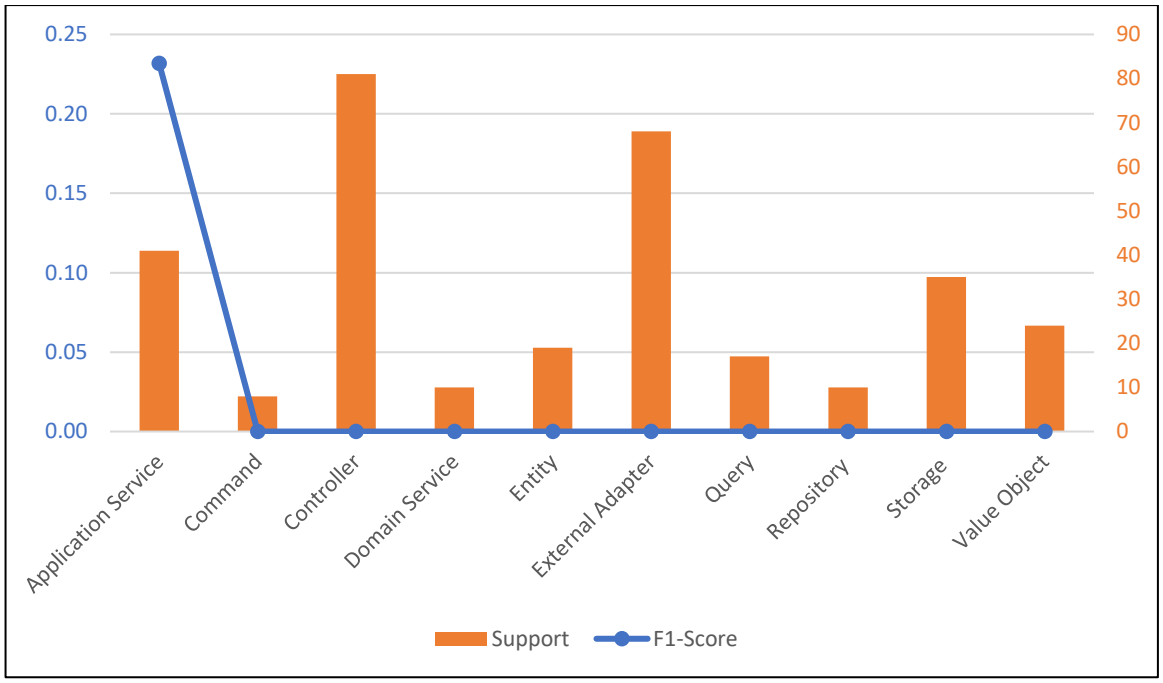
Berikut merupakan tabel evaluasi model yang merepresentasikan hasil dari pelatihan model CodeBERT:

Tabel 4.5 Hasil Evaluasi Model Pre-Trained CodeBERT

Peran Komponen	Precision	Recall	F1-Score	Support (Jumlah Data)
Application Service	0,131	1	0,232	41
Command	0	0	0	8

Controller	0	0	0	81
Domain Service	0	0	0	10
Entity	0	0	0	19
External	0	0	0	68
Query	0	0	0	17
Repository	0	0	0	10
Storage	0	0	0	35
Value Object	0	0	0	24
Weighted Avg F1-Score		0,030342413		313
Akurasi Keseluruhan		13,10%		

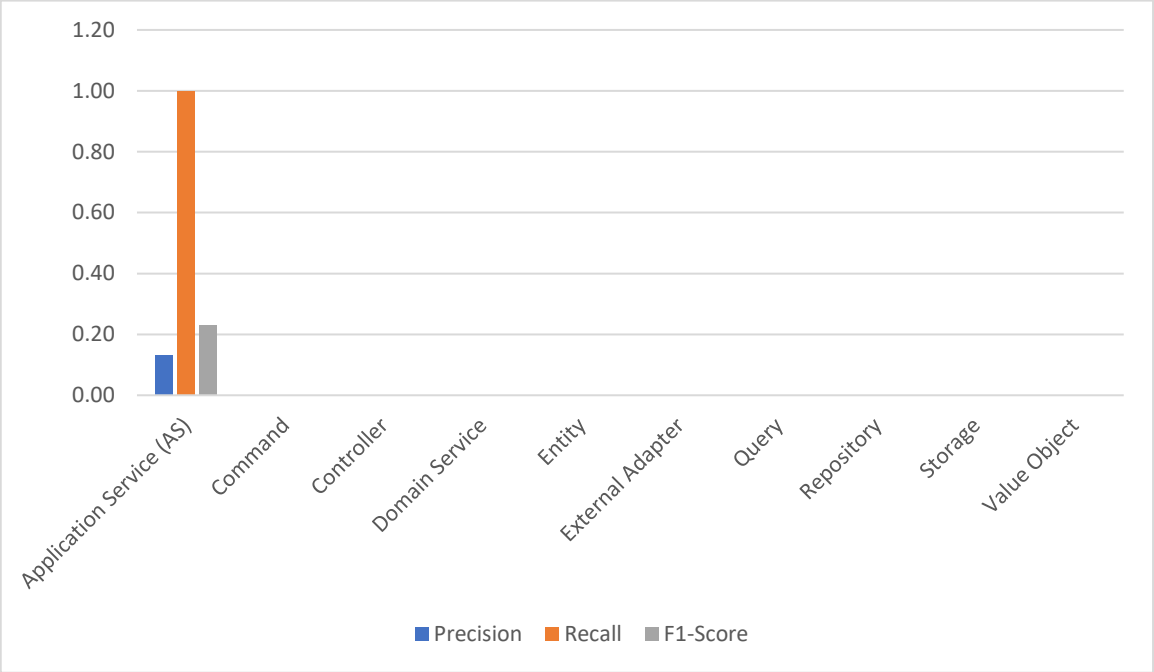
Berikut merupakan grafik yang merepresentasikan perbandingan F1-Score dengan jumlah datanya (Support):



Gambar 4.3 Grafik Perbandingan F1-Score dan Support Model Pre-Trained CodeBERT

Berdasarkan Gambar 4.3, tercatat bahwa jumlah data (*support*) tidak memastikan hasil *F1-Score* memiliki nilai yang tinggi. Kesimpulan ini dapat dilihat dari hasil *Controller* dengan *F1-Score* 0 meskipun *support* mencapai 81. Kejadian ini dapat dilihat pada komponen-komponen lainnya, dengan *Application Service* yang menjadi *outlier* dengan *F1-Score* 0,232 dengan *support* yang hanya mencapai 41.

Dan berikut merupakan grafik yang merepresentasikan perbandingan stabilitas model berdasarkan *Precision*, *Recall*, dan F1-Score:



Gambar 4.4 Grafik Perbandingan Precision, Recall, dan F1-Score Model Pre-Trained CodeBERT

Gambar 4.4 menyajikan perbandingan metrik untuk melihat stabilitas prediksi model. Pada komponen *Application Service*, nilai *Precision* mencapai 0,13 atau sekitar 13% dan *Recall* mencapai angka yang sangat tinggi yaitu 1. Ketidakseimbangan ini menunjukkan bahwa model cenderung sangat agresif dalam melabel dataset uji sebagai komponen *Application Service*. dan menunjukkan bahwa tanpa *fine-tuning*, CodeBERT masih belum bisa melabel komponen peran arsitektur.

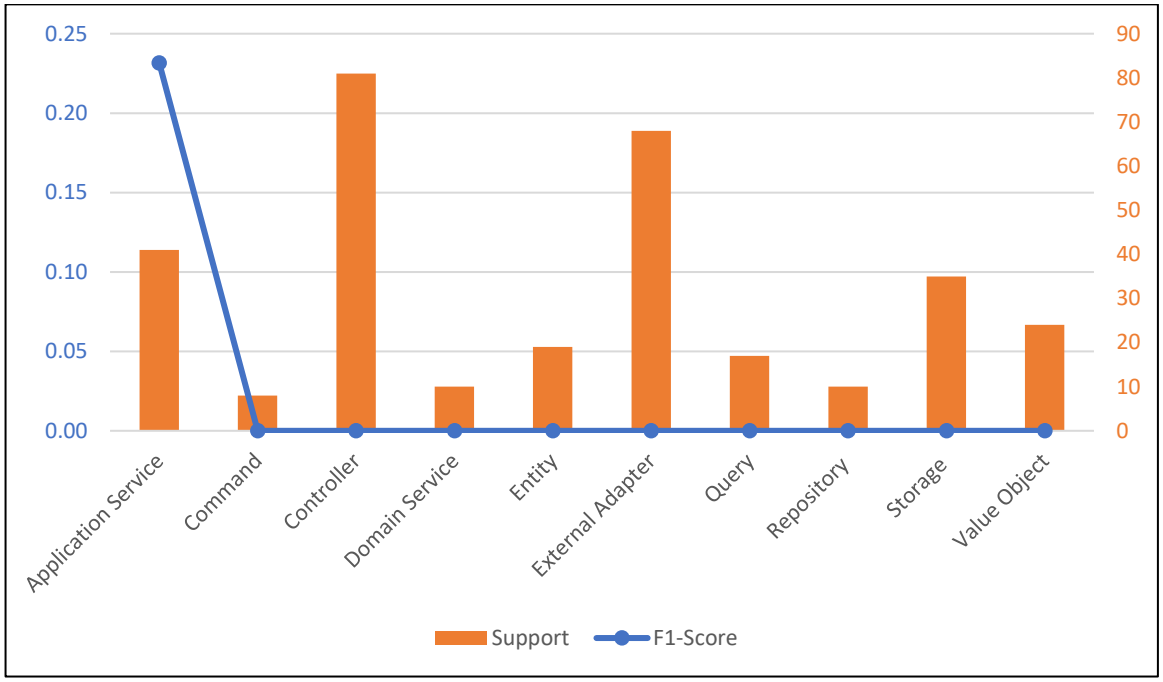
4.1.3.2 Evaluasi Model Klasifikasi Pre-Trained EL-CodeBERT

Berikut merupakan tabel evaluasi model yang merepresentasikan hasil dari pelatihan model CodeBERT:

Tabel 4.6 Hasil Evaluasi Model Pre-Trained EL-CodeBERT

Peran Komponen	Precision	Recall	F1-Score	Support (Jumlah Data)
Application Service	0,13	1	0,23	41
Command	0	0	0	8
Controller	0	0	0	81
Domain Service	0	0	0	10
Entity	0	0	0	19
External	0	0	0	68
Query	0	0	0	17
Repository	0	0	0	10
Storage	0	0	0	35
Value Object	0	0	0	24
Weighted Avg F1-Score		0,030342413		313
Akurasi Keseluruhan		13,10%		

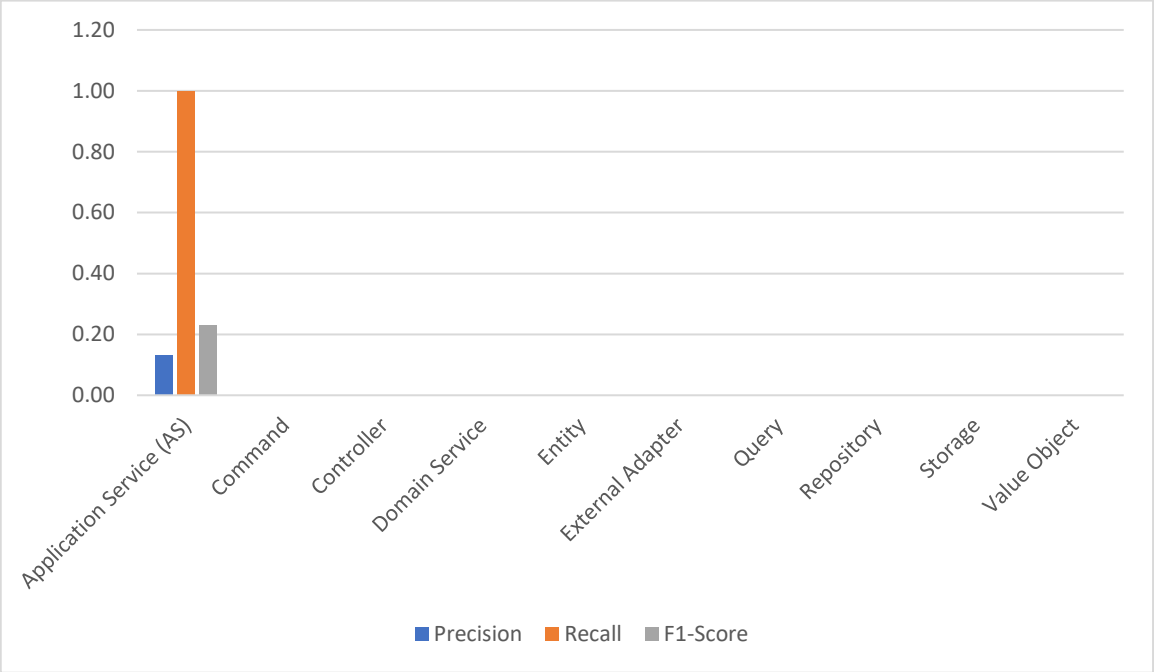
Berikut merupakan grafik yang merepresentasikan perbandingan F1-Score dengan jumlah datanya (*Support*):



Gambar 4.5 Grafik Perbandingan F1-Score dan Support Model Pre-Trained EL-CodeBERT

Kejadian yang sama pada model CodeBERT *Pre-Trained* dapat terlihat pada Gambar 4.5, yaitu jumlah *support* tidak menentukan hasil akhir F1-Score model. Terlihat bahwa *Application Service* yang hanya memiliki *support* 41 memiliki F1-Score 0,23, sedangkan nilai F1-Score komponen lainnya 0.

Dan berikut merupakan grafik yang merepresentasikan perbandingan stabilitas model berdasarkan *Precision*, *Recall*, dan F1-Score:



Gambar 4.6 Grafik Perbandingan Precision, Recall, dan F1-Score Model Pre-Trained EL-CodeBERT

Gambar 4.6 mengindikasikan hal yang sama pada model sebelumnya, yaitu *Application Service* sekali lagi memiliki nilai *precision* yang mencapai 0,13 dan *recall* yang mencapai 1. Nilai tersebut mengindikasikan bahwa model cenderung agresif dalam menebak *snippet* sebagai komponen *Application Service.* Ketidakseimbangan ini mengonfirmasi kesimpulan sebelumnya, bahwa tanpa *fine-tuning*, model masih belum dapat melabel komponen dengan benar.

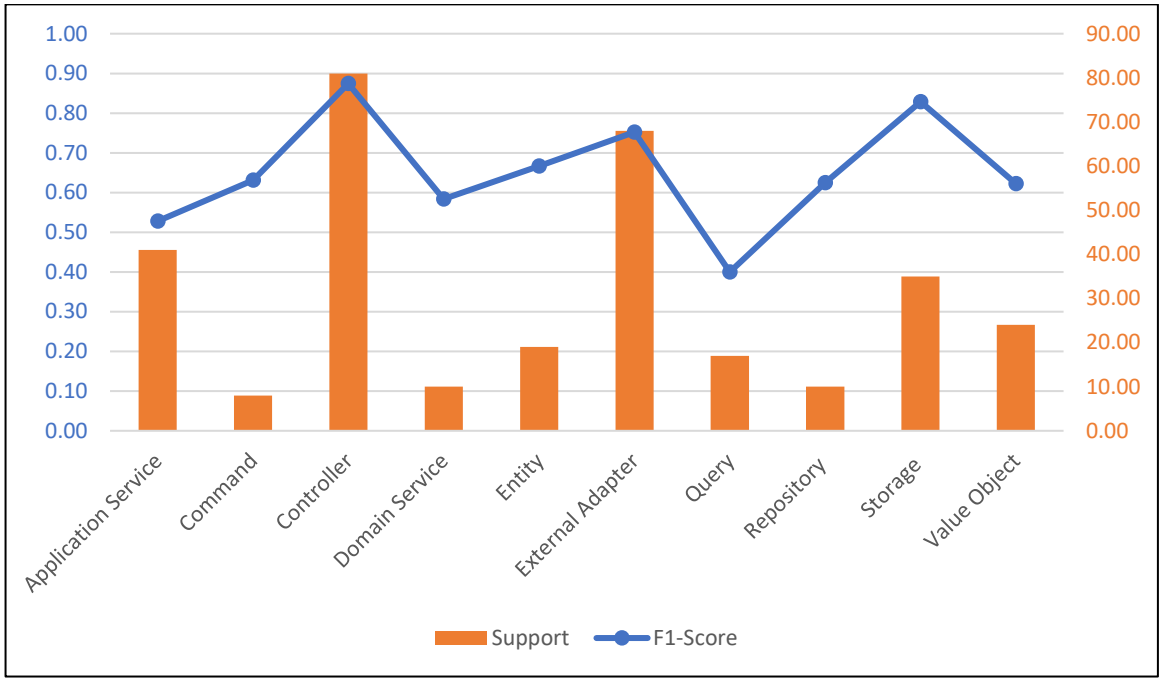
4.1.3.3 Evaluasi Model Klasifikasi Fine-tuned CodeBERT

Berikut merupakan tabel evaluasi model yang merepresentasikan hasil dari pelatihan model CodeBERT:

Tabel 4.7 Hasil Evaluasi Model CodeBERT

Peran Komponen	Precision	Recall	F1-Score	Support (Jumlah Data)
Application Service	0,61	0,46	0,53	41
Command	0,55	0,75	0,63	8
Controller	0,85	0,90	0,87	81
Domain Service	0,50	0,70	0,58	10
Entity	0,61	0,74	0,67	19
External	0,73	0,78	0,75	68
Query	0,46	0,35	0,40	17
Repository	0,83	0,50	0,63	10
Storage	0,83	0,83	0,83	35
Value Object	0,67	0,58	0,62	24
Weighted Avg F1-Score		0.716006		313
Akurasi Keseluruhan		72,20%		

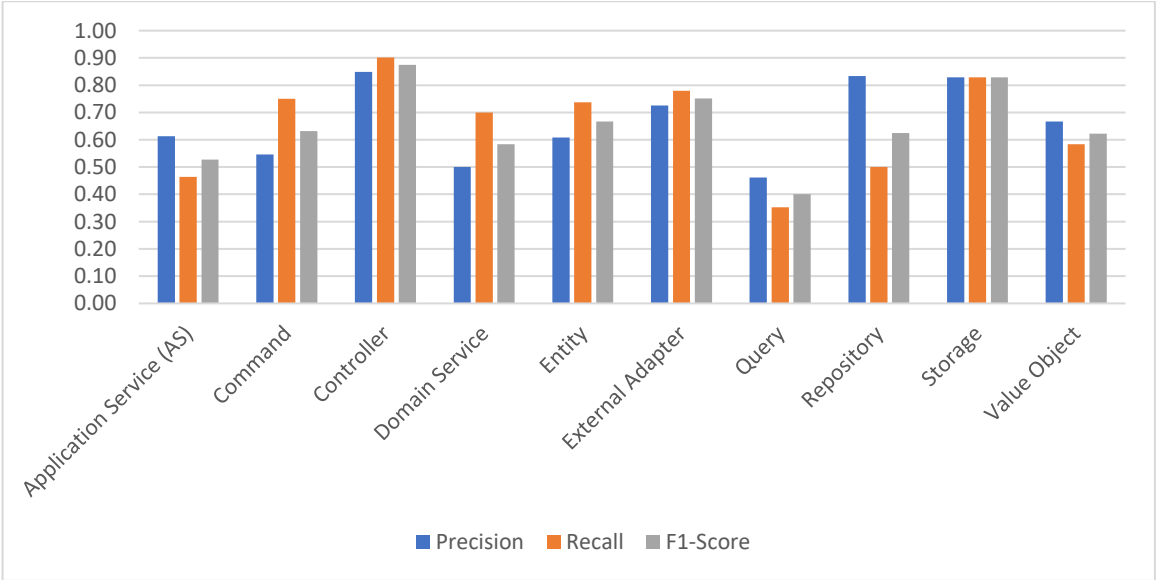
Berikut merupakan grafik yang merepresentasikan perbandingan F1-Score dengan jumlah datanya (*Support*):



Gambar 4.7 Grafik Perbandingan F1-Score dan Support Model CodeBERT

Berdasarkan Gambar 4.7, terlihat adanya perbedaan dengan model CodeBERT *Pre-Trained*. Pada *Controller* dan *External*, terlihat korelasi positif antara *support* dengan nilai F1-Score. *Controller* memiliki F1-Score tertinggi pada 0,87 dengan *support* tertinggi pada 81 data, dan *External* memiliki F1-Score pada 0,75 dengan *support* 68 data. Terdapat *outlier* pada komponen *Storage* yang memiliki F1-Score 0,83 meskipun memiliki *support* yang hanya ada pada 35 data.

Dan berikut merupakan grafik yang merepresentasikan perbandingan stabilitas model berdasarkan *Precision*, *Recall*, dan F1-Score:



Gambar 4.8 Grafik Perbandingan Precision, Recall, dan F1-Score Model CodeBERT

Gambar 4.8 menyajikan perbandingan metrik untuk melihat stabilitas prediksi model. Pada komponen *Storage*, nilai *Precision* dan *Recall* mencapai angka yang seimbang, pada nilai 0,83, menunjukkan bahwa model sangat stabil dalam mengidentifikasi komponen implementasi penyimpanan data. Sementara pada komponen *Command, Domain Service*, dan *Entity* memiliki nilai *Recall* jauh lebih tinggi daripada *Precision*, yang berarti model cenderung sangat agresif dalam melabeli sebuah file sebagai komponen-komponen tersebut, meskipun terdapat beberapa kesalahan klasifikasi di dalamnya. Terakhir, *Controller* memiliki nilai *precision, recall,* dan F1-Score tertinggi. Hal ini mengindikasikan bahwa model sudah memiliki pemahaman yang sangat tinggi terhadap *Controller* dan sudah membangun representasi vektor yang selaras dengan karakteristik aktual *source code*.

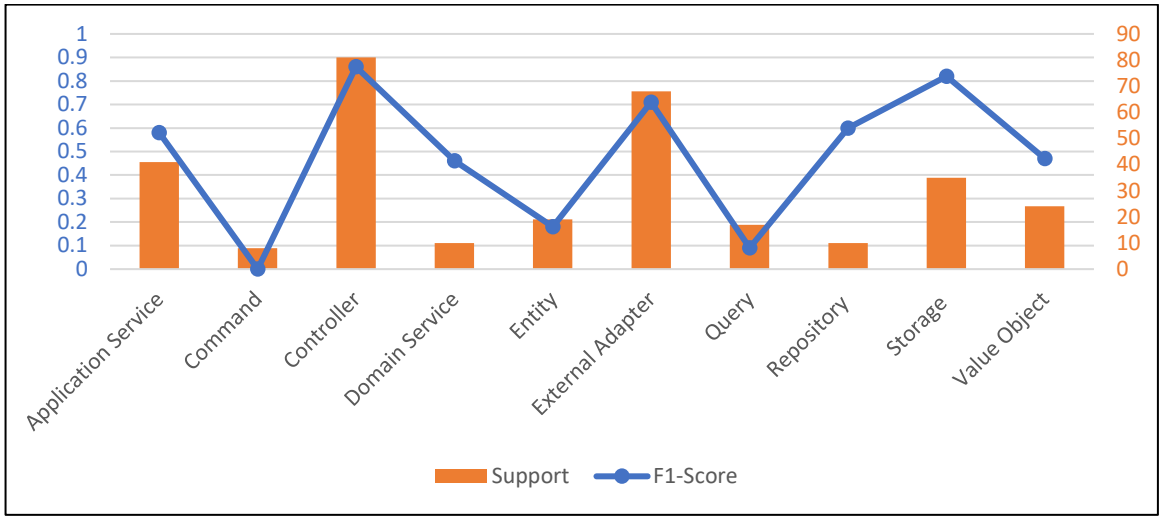
4.1.3.4 Evaluasi Model Klasifikasi Fine-tuned El-CodeBERT

Berikut merupakan tabel evaluasi model yang merepresentasikan hasil dari pelatihan model CodeBERT:

Tabel 4.8 Hasil Evaluasi Model EL-CodeBERT

Peran Komponen	Precision	Recall	F1-Score	Support (Jumlah Data)
Application Service	0,51	0,68	0,58	41
Command	0,00	0,00	0,00	8
Controller	0,84	0,88	0,86	81
Domain Service	1,00	0,30	0,46	10
Entity	0,67	0,11	0,18	19
External	0,68	0,75	0,71	68
Query	0,17	0,06	0,09	17
Repository	0,60	0,60	0,60	10
Storage	0,81	0,83	0,82	35
Value Object	0,38	0,63	0,47	24
Weighted Avg F1-Score		0.629709		313
Akurasi Keseluruhan		65,81%		

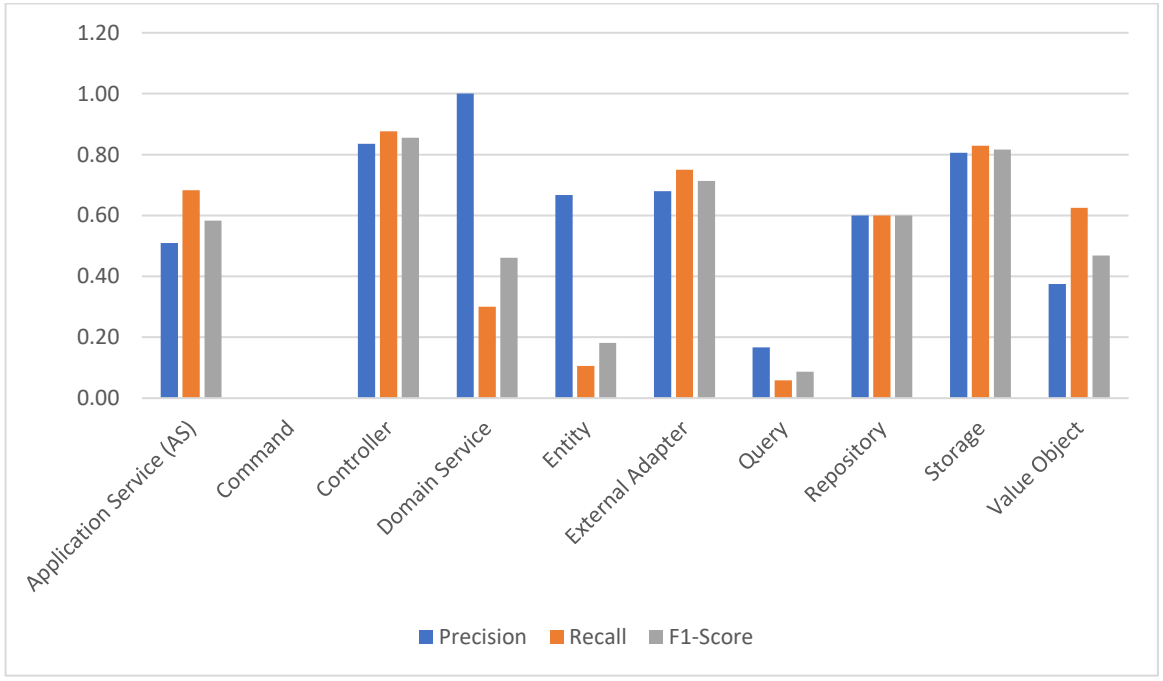
Berikut merupakan grafik yang merepresentasikan perbandingan F1-Score dengan jumlah datanya (Support):



Gambar 4.9 Grafik Perbandingan F1-Score dan Support Model EL-CodeBERT

Berdasarkan Gambar 4.9, fenomena yang dialami oleh model CodeBERT kembali terlihat pada model EL-CodeBERT, yaitu adanya korelasi positif antara *support* dengan nilai F1-Score dan adanya *outlier* pada komponen yang sama. *Controller* dengan *support* 81 data memiliki nilai F1-Score tertinggi pada 0,86 dan *External* dengan *support* 68 data memiliki nilai F1-Score 0,71. Fenomena *outlier* pada komponen *Storage* terjadi lagi dengan F1-Score mencapai 0,82 dengan *support* hanya ada pada di 35 data.

Dan berikut merupakan grafik yang merepresentasikan perbandingan stabilitas model berdasarkan *Precision*, *Recall*, dan F1-Score:



Gambar 4.10 Grafik Perbandingan Precision, Recall, dan F1-Score Model EL-CodeBERT

Gambar 4.10 menyajikan perbandingan metrik untuk melihat stabilitas prediksi model. Pada komponen *Repository, Controller,* dan *Storage*, nilai *precision* dan *recall* memiliki rasio yang cukup seimbang, dengan *Repository* memiliki rasio yang sangat stabil pada nilai 0,6 atau rasio 1:1. Ini mengindikasikan bahwa model sudah mulai membangun representasi vektor dengan karakteristik *source code* yang mendekati akurat. Pada sisi lain, *Command* dan *Query* memiliki nilai yang sangat rendah, dengan *Command* menjadi komponen dengan nilai terendah, nilai 0, di seluruh metrik stabilitas model.

4.1.4 Hasil Inferensi pada Proyek Legacy

Dataset yang digunakan dalam penelitian ini dikumpulkan dari repositori publik di GitHub yang tidak menerapkan prinsip CA pada bahasa pemrograman Java.

4.1.4.1 Daftar Repositori

Tabel 4.9 Daftar Repositori Inferensi

No	Nama Repositori	Jumlah Sampel
1	BookStore-master	12
2	BikeCourier-master	30

4.1.4.2 Distribusi Data

Berikut merupakan distribusi dataset berdasarkan peran komponen yang digunakan sebagai dataset inferensi

Tabel 4.10 Distribusi Data Berdasarkan Peran Komponen

Komponen	**Jumlah Sampel**	**Persentase (%)**
Controller	12	28,57%
External	3	7,14%
Entity	7	16,67%
Value Object	2	4,76%
Application Service	1	2,38%
Storage	3	7,14%
Query	0	0%
Repository	1	2,38%
Domain Service	12	28,57%
Command	1	2,38%
Total	**42**	**100%**

Tabel 4.10 menunjukkan rincian distribusi label pada dataset inferensi yang berjumlah 42 file kode. Terdapat konsentrasi data yang tinggi pada komponen **Controller** dan **Domain Service** yang secara kolektif mencakup lebih dari 50% total dataset. Hal ini memberikan gambaran nyata mengenai struktur sistem *legacy* yang sering kali memiliki logika bisnis yang tersebar antara kontroler dan layanan mandiri. Tercatat, *Query* merupakan pengecualian, sehingga pengujian generalisasi pada tahap ini hanya berfokus pada 9 peran lainnya.

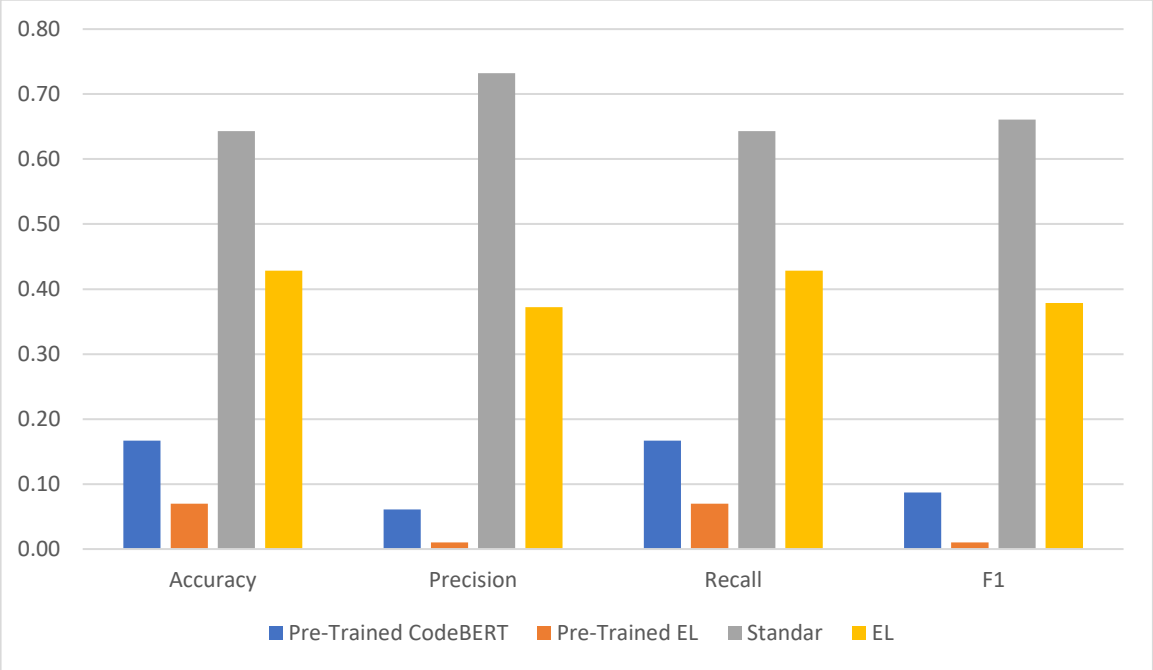
4.1.4.3 Hasil Inferensi Seluruh Model

Berikut merupakan tabel hasil inferensi *legacy* dataset yang dilakukan oleh seluruh model berdasarkan setiap komponennya.

Tabel 4.11 Hasil Inferensi

Model	Accuracy	Precision weighted	Recall weighted	F1 weighted	Total Error
Pre-Trained CodeBERT	0,17	0,06	0,17	0,09	35
Pre-Trained EL-CodeBERT	0,07	0,01	0,07	0,01	39
CodeBERT	0,64	0,73	0,64	0,66	15
EL	0,43	0,37	0,43	0,38	24

Dan berikut merupakan grafik representasi perbandingan hasil tiap model:



Gambar 4.11 Grafik Perbandingan Precision, Recall, dan F1-Score Setiap Model

Gambar 4.11 mengilustrasikan perbandingan performa antara empat varian model dalam mengklasifikasikan peran komponen pada dataset inferensi. Visualisasi ini mempertegas dominasi Model CodeBERT yang mencapai nilai tertinggi pada seluruh metrik, dengan akurasi 64,29% dan *weighted* F1-Score sebesar 0,66.

Sebaliknya, EL-CodeBERT menunjukkan degradasi performa yang signifikan dengan akurasi hanya 42,86%. Hal ini mengonfirmasi bahwa arsitektur agregasi multi-lapis lebih rentan terhadap bising sintaksis dari sistem *legacy*, di mana informasi dari lapisan transformer awal justru mengaburkan fitur semantik fungsional yang matang di lapisan akhir (fenomena *feature dilution*).

4.2 Pembahasan

Setelah menyajikan data hasil penelitian secara objektif, bagian pembahasan ini bertujuan untuk memberikan interpretasi dan analisis terhadap perilaku model dalam mengenali struktur arsitektur perangkat lunak. Selain itu, bagian pembahasan ini menganalisis generalisasi model pada sistem *legacy* dan perbandingan efektivitasnya.

4.2.1 Analisis Performa Model

Berdasarkan Tabel 4.8 Hasil Evaluasi CodeBERT, dapat dilihat bahwa *Controller* memiliki F1-Score yang melampaui peran komponen lainnya dengan F1-Score mencapai 0,87. Hal yang serupa dapat ditemukan pada Tabel 4.9 Hasil Evaluasi EL-CodeBERT, dengan F1-Score mencapai 0,86. Kemudian komponen *Storage* mencapai nilai serupa pada kedua model tersebut dengan F1-Score yang mencapai 0,83 untuk CodeBERT dan 0,82 untuk EL-CodeBERT.

Ada beberapa faktor yang memengaruhi F1-Score *Controller* dan *Storage* hingga dapat mencapai angka tersebut. Faktor pertama adalah karena ada beberapa kata kunci yang spesifik, seperti "*showLoading*", "*hideLoading*", atau penggunaan anotasi UI yang sering muncul pada paket *presentation*. Faktor selanjutnya adalah pola metode yang sangat spesifik sehingga model dapat mengenali pola implementasi yang memiliki struktur khas dibandingkan komponen lainnya.

Sebaliknya, *Command* dan *Query* memiliki F1-Score yang sangat rendah dibandingkan komponen lainnya pada model CodeBERT dan EL-CodeBERT. Beberapa faktor yang memengaruhi *Command* dan *Query* mendapatkan F1-Score rendah adalah karena *Command* hanya memiliki 15 sampel sedangkan *Query* hanya memiliki 65 sampel dari keseluruhan 1525 file. Model masih belum memiliki cukup referensi untuk membedakan *Command* dan *Query* dengan komponen lainnya. Selain itu, sinyal teknis *Command* masih lemah. Jika dibandingkan dengan *Controller*, ia memiliki sinyal yang kuat seperti anotasi "@GET" atau "@Bind", sedangkan implementasi *Command* dan *Query* sering kali hanya berupa kelas Java biasa (*Plain Old Java Object*). Terakhir, terdapat limitasi dalam jumlah *Epoch* yang hanya cukup untuk mengoptimalkan komponen-komponen mayoritas seperti *Controller* dan *External*, dan tidak cukup untuk model untuk memahami fitur-fitur yang membedakan *Command* dengan komponen lainnya.

Berdasarkan hasil evaluasi model-model *Pre-Trained* pada Tabel 4.5 dan Tabel 4.6, kedua varian model *pre-trained*, baik CodeBERT dan EL-CodeBERT, menunjukkan performa yang identik dengan akurasi yang rendah sebesar 13,10%. Tanpa proses *fine-tuning*, lapisan klasifikasi (*classifier head*) yang diinisialisasi secara acak gagal menangkap fitur semantik arsitektur CA. Ketidakmampuan *pre-trained* EL dalam melabeli lebih dari 3 sampel dengan benar menegaskan bahwa arsitektur multi lapis sangat bergantung pada proses *fine-tuning* untuk menentukan relevansi tiap lapisan. Tanpa itu, fitur semantik dari lapisan atas terkubur oleh informasi permukaan dari lapisan-lapisan awal. Hal ini sangat memperkuat analisis bahwa kurangnya *fine-tuning* akan meningkatkan kegagalan model dalam melabel komponen secara benar.

Kesimpulan ini dibuktikan dengan nilai *recall Application Service* sebesar 1.0, namun *precision* hanya 0,13. Data tersebut mengonfirmasi bahwa model hanya menebak seluruh 313 data uji sebagai satu kelas mayoritas (*Application Service*) tanpa kemampuan pembedaan antar komponen. Temuan ini memberikan justifikasi kuat bahwa bobot asli dari model Microsoft

CodeBERT murni memahami sintaksis bahasa pemrograman secara umum, namun memerlukan adaptasi bobot melalui *fine-tuning* untuk memahami peran arsitektural spesifik.

4.2.2 Analisis Kesalahan Model

Subbab ini menganalisis pola kesalahan klasifikasi model melalui data uji yang disajikan dalam bentuk *Confusion Matrix* yang berisi dengan prediksi model dan label aktual *snippet*.

4.2.2.1 Confusion Matrix Pre-Trained CodeBERT dan EL-CodeBERT

Tabel 4.12 Confusion Matrix Pre-Trained CodeBERT dan EL-CodeBERT

Aktual / Prediksi	AS	Command	Controller	DS	Entity	External	Query	Repo	Storage	VO
AS	41	0	0	0	0	0	0	0	0	0
Command	8	0	0	0	0	0	0	0	0	0
Controller	81	0	0	0	0	0	0	0	0	0
DS	10	0	0	0	0	0	0	0	0	0
Entity	19	0	0	0	0	0	0	0	0	0
External	68	0	0	0	0	0	0	0	0	0
Query	17	0	0	0	0	0	0	0	0	0
Repository	10	0	0	0	0	0	0	0	0	0
Storage	35	0	0	0	0	0	0	0	0	0
VO	24	0	0	0	0	0	0	0	0	0

Tercatat pada Tabel *Confusion Matrix* 4.12, bahwa komponen *Application Service* diprediksikan oleh model sebanyak 313 kali, atau dengan kata lain, seluruh dataset uji dilabel sebagai *Application Service*. Hal ini mengindikasikan bahwa model *Pre-Trained* masih tidak dapat melabel komponen peran arsitektur CA. Kedua model tersebut masih kesulitan untuk membedakan atau mengidentifikasi komponen karena model-model tersebut masih belum memiliki konteks yang cukup mengidentifikasikan komponen-komponen yang ada berdasarkan karakteristik komponen yang ada pada *source code*.

4.2.2.2 Confusion Matrix CodeBERT

Tabel 4.13 Confusion Matrix CodeBERT

Aktual / Prediksi	AS	Command	Controller	DS	Entity	External	Query	Repo	Storage	VO

AS	19	4	8	1	0	6	2	0	1	0
Command	1	6	0	0	1	0	0	0	0	0
Controller	2	0	73	1	0	5	0	0	0	0
DS	1	0	0	7	0	0	0	0	1	1
Entity	0	0	1	0	14	1	0	0	2	1
External	4	0	4	1	1	53	1	1	1	2
Query	4	0	0	0	2	2	6	0	0	3
Repository	0	1	0	0	0	1	2	5	1	0
Storage	0	0	0	3	1	2	0	0	29	0
VO	0	0	0	1	4	3	2	0	0	14

Berdasarkan Tabel 4.12, tercatat model sudah mencapai tingkat keberhasilan prediksi yang tinggi pada komponen yang memiliki sinyal sintaksis yang kuat. Komponen *Controller* mencapai nilai *recall* tertinggi sebesar 0,90, dengan hasil prediksi sampel yang benar 73 dari 81 total sampel *Controller*. Hasil ini mengonfirmasi bahwa model berhasil memelajari pola anotasi UI dan interaksi luar pada lapisan *Presentation* (Martin, 2019).

Selain itu, tercatat pada komponen *Application Service* ada kesalahan prediksi sebanyak 8 kali sebagai *Controller* dan 6 kali sebagai *External*. Ini mengindikasikan bahwa adanya kemiripan pola kode yang menyebabkan *Application Service* sering kali mengandung referensi yang kuat ke komponen luar, sehingga model kesulitan dalam mengisolasi logika koordinasi *Application Service.*

4.2.2.3 Confusion Matrix EL-CodeBERT

Tabel 4.14 Confusion Matrix EL-CodeBERT

Aktual / Prediksi	AS	Command	Controller	DS	Entity	External	Query	Repo	Storage	VO
AS	28	0	7	0	0	5	0	0	1	0
Command	6	0	0	0	0	0	0	0	0	2
Controller	6	0	71	0	0	3	0	0	0	1
DS	0	0	0	3	0	1	1	2	1	2
Entity	0	0	1	0	2	2	0	0	2	12

External	5	0	5	0	0	51	1	1	2	3
Query	7	0	0	0	0	4	1	0	1	4
Repository	1	0	0	0	0	2	1	6	0	0
Storage	2	0	0	0	0	2	0	1	29	1
VO	0	0	1	0	1	5	2	0	0	15

Berdasarkan analisa yang dilakukan terhadap Tabel 4.14 menunjukkan bahwa EL-CodeBERT mengalami penurunan yang signifikan dari model CodeBERT. Kesalahan paling krusial terjadi pada komponen *Command* dengan nilai *recall* yang ada pada di 0,0 (nol mutlak) karena gagal dalam menebak satu pun dari 8 sampel yang tersedia. Mayoritas sampel *Command* salah diklasifikasikan sebagai *Application Service*.

Dapat dilihat juga pada sampel komponen *Entity* yang salah diklasifikasikan dengan komponen *Value Object* sebanyak 12 kali. Hal ini membuktikan bahwa arsitektur *Aggregated Embedding* yang mengakses lapisan 0-12 justru memperburuk pemahaman model terhadap konsep "identitas unik" pada *Entity* (Liu et al., 2022). Lapisan awal (0-4) yang menangkap fitur sintaksis dasar, seperti deklarasi atribut, memberikan *noise* atau distraksi yang mengaburkan fitur semantik dari lapisan akhir sehingga model cenderung melabeli *Entity* sebagai *Value Object* karena kemiripan struktur data.

4.2.3 Analisis Generalisasi Model pada Sistem Legacy

Pengujian pada sistem *legacy* dilakukan untuk menjawab RQ2 mengenai ketangguhan model saat menghadapi kode yang tidak terstruktur secara standar CA. Eksperimen ini melibatkan 42 sampel Java yang diambil dari dua repositori: BikeCourier-master (30 sampel) dan BookStore-master (12 sampel).

Hasil perbandingan seluruh model pada Tabel 4.11 menunjukkan pola yang konsisten dengan hasil evaluasi data uji, dengan performa *baseline* varian *pre-trained* CodeBERT pada sistem *legacy* hanya mencapai akurasi 16,67% atau 35 kesalahan dari 42 sampel. Rendahnya hasil ini menjadi patokan batas bawah bahwa tanpa pelatihan, model tidak dapat diandalkan untuk membantu proses migrasi arsitektur. Hasil inferensi menunjukkan bahwa *pre-trained* EL memiliki performa terendah, pada 7,14%, dibandingkan seluruh model lainnya. Hal ini membuktikan secara empiris bahwa mengambil informasi dari seluruh lapisan (0-12) tanpa bobot yang terlatih hanya akan mengakumulasi *noise* sintaksis, sehingga model gagal menangkap sinyal arsitektur apa pun pada sistem legacy.

Lonjakan signifikan terlihat pada Model CodeBERT yang mencapai akurasi 64,29%. Kemampuan model untuk tetap mengenali komponen inti seperti *Controller* atau *Entity* pada sistem *legacy* membuktikan bahwa model telah berhasil membangun representasi vektor yang bersifat fungsional, bukan sekadar mengikuti struktur folder (*packaging*). Penurunan dari akurasi data uji (72,20%) ke data *legacy* (64,29%) dianggap wajar mengingat adanya fenomena kode tidak terstruktur pada sistem lama (Martin, 2019).

Di sisi lain, EL-CodeBERT hanya mencapai akurasi 42,86% pada data *legacy*. Hal ini mengonfirmasi bahwa penambahan parameter kompleks pada arsitektur EL-CodeBERT justru membuat model lebih sensitif terhadap *noise* pada kode *legacy*. Pada proyek seperti BikeCourier-master yang mengandung banyak logika sensor mentah, informasi multi-lapis EL cenderung menangkap fitur sintaksis yang menyesatkan, sehingga gagal melakukan generalisasi sebaik model CodeBERT yang hanya fokus pada lapisan semantik terakhir.

4.2.4 Perbandingan Performa Model

Berdasarkan hasil pengujian secara menyeluruh, terdapat perbedaan performa yang signifikan antara ketiga skenario model yang diuji. Tabel 4.15 merangkum perbandingan metrik utama untuk memberikan gambaran komprehensif mengenai efektivitas setiap pendekatan.

Tabel 4.15 Perbandingan Performa Model

Model	Accuracy (W)	Precision (W)	Recall (W)	F1-Score (W)
Pre-trained	13,10%	0,02	0,13	0,03
CodeBERT	72,20%	0,72	0,72	0,72
EL-CodeBERT	65,81%	0,65	0,66	0,63

Hasil evaluasi menunjukkan bahwa Model Standar mengungguli EL-CodeBERT dengan selisih akurasi sebesar 6,39%. Fenomena di mana arsitektur yang lebih kompleks (EL-CodeBERT) menghasilkan performa yang lebih rendah dibandingkan model dasar setelah *fine-tuning* dapat dianalisis melalui tiga sudut pandang ilmiah.

Pertama, faktor Keterbatasan Data (*Data Scarcity*). Arsitektur EL-CodeBERT yang menggunakan teknik *Weighted Aggregated Embedding* memperkenalkan parameter tambahan berupa *layer_weights* yang harus dipelajari dari titik nol (*random initialization*). Dalam teori *machine learning*, model dengan ruang parameter yang lebih luas membutuhkan volume data yang jauh lebih besar untuk mencapai titik konvergensi optimal (Goodfellow, 2016).

Kedua, adanya fenomena Pengenceran Fitur (*Feature Dilution*). Secara teoritis, lapisan awal *transformer* (lapisan 0-4) cenderung menangkap fitur permukaan (*surface features*) dan *noise* sintaksis, sedangkan lapisan akhir (lapisan 9-12) menangkap fitur semantik fungsional (Liu et al., 2022). Pada komponen arsitektur yang sangat bergantung pada identitas unik seperti *Entity*, akses terhadap lapisan awal justru memperkeruh sinyal semantik matang dari lapisan ke-12. Hal ini terbukti dari banyaknya kesalahan klasifikasi *Entity* sebagai *Value Object* pada model EL-CodeBERT dibandingkan model CodeBERT.

Ketiga, berlakunya hukum *Occam's Razor* dalam *deep learning*. Prinsip ini menyatakan bahwa pada kondisi dataset yang terbatas, model yang lebih sederhana cenderung lebih kokoh (*robust*) dan memiliki kemampuan generalisasi yang lebih baik (Sun, 2025). Model CodeBERT, yang sepenuhnya memercayai ekstraksi fitur lapisan terakhir bawaan Microsoft yang telah dilatih pada miliaran baris kode (Feng et al., 2022), terbukti lebih efisien dalam menangkap peran komponen arsitektur CA dibandingkan mencoba mendefinisikan ulang porsi pembobotan antar-lapisan secara mandiri.

BAB 5 KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil penelitian, implementasi, dan pengujian yang telah dilakukan mengenai klasifikasi peran komponen arsitektur menggunakan model CodeBERT, maka dapat ditarik beberapa kesimpulan sebagai berikut:

5.1.1 Pembangunan Model Klasifikasi

Menjawab permasalahan pertama, penelitian ini berhasil menjawab permasalahan pertama dengan membangun model klasifikasi peran komponen arsitektur perangkat lunak berbasis microsoft/codebert-base melalui tahapan *fine-tuning*. Hasil eksperimen membuktikan bahwa proses *fine-tuning* sangat penting. Model yang sudah dilatih (CodeBERT) mampu mencapai akurasi optimal sebesar 72,20% dengan F1-Score 0,72. Angka ini menunjukkan peningkatan yang sangat signifikan dibandingkan model *pre-trained* asli (tanpa *fine-tuning*) yang hanya memiliki akurasi 13,10% akibat fenomena *Output Degeneracy* atau *Collapsed Prediction*.

5.1.2 Akurasi Model Klasifikasi

Menjawab permasalahan kedua, sesuai dengan hasil penelitian yang telah dilakukan, meskipun akurasi inferensi mengalami penurunan, hal ini masih sesuai dengan ekspektasi praktik karena model berhasil membuktikan ia dapat membangun representasi vektor berdasarkan logika kode, dan bukan hanya sekadar menghafal struktur folder atau konvensi penamaan standar.

5.2 Saran

5.2.1 Penyeimbangan Kelas (*Class Balancing*)

Mengingat adanya ketimpangan data yang sangat besar antara komponen *Controller* dengan komponen seperti *Command* atau *Domain Service*, penelitian selanjutnya disarankan menggunakan teknik augmentasi data atau fungsi *loss* yang memberikan bobot lebih pada kelas minoritas untuk meningkatkan nilai *Recall* pada komponen-komponen sulit.

5.2.2 Integrasi Struktur AST

Mengingat keterbatasan model dalam membedakan *Entity* dan *Value Object* karena kemiripan deklarasi atributnya, penggunaan informasi dari *Abstract Syntax Tree* (AST) dapat dipertimbangkan untuk membantu model menangkap struktur relasional antar-objek secara lebih mendalam.

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., dan Polosukhin, I., Des. 2017. "Attention is all you need". *Advances in Neural Information Processing Systems* 30:5998–6008

Devlin, J., Chang, M. W., Lee, K., dan Toutanova, K. 2019. "BERT: Pre-training of deep bidirectional transformers for language understanding". *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Minneapolis, 2-7 Juni. Vol. 1:4171–4186.

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., dan Zhou, M. 2020. "CodeBERT: A pre-trained model for programming and natural languages". *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online, 16-20 November. 1536–1547.

Liu, K., Yang, G., Chen, X., dan Zhou, Y. 2022. "EL-CodeBert: Better Exploiting CodeBert to Support Source Code-Related Classification Tasks". *Proceedings of the 13th Asia-Pacific Symposium on Internetware (Internetware 2022)*. Hohhot, 11-12 Juni. 147–155.

Wang, S., Wang, D., Zhou, M., dan Tang, D. 2021. "Evaluating and enhancing pre-trained programming language representations with semantic-aware probing". *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online dan Punta Cana, 7-11 November. 8692–8704.

Martin, R. C. 2019. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Boston: Pearson Education, Inc.

Artificial Analysis. 2026. *Artificial Analysis Intelligence Index*, URL:https://artificialanalysis.ai/evaluations/artificial-analysis-intelligence-index.

Gezici, G., Giannotti, F., Pedreschi, D., Knott, A., dan Pappalardo, L. 2024. "Learning by Surprise: Surplexity for Mitigating Model Collapse in Generative AI". *arXiv preprint arXiv:2410.12341*, URL:https://doi.org/10.48550/arXiv.2410.12341.

Millett, S. dan Tune, N. 2015. *Patterns, Principles, and Practices of Domain-Driven Design*. Indianapolis: John Wiley & Sons, Inc.

Vernon, V. 2013. *Implementing Domain-Driven Design*. Upper Saddle River: Addison-Wesley.

Goodfellow, I., Bengio, Y., dan Courville, A. 2016. *Deep Learning*. Cambridge: MIT Press

Sun, K., dan Nielsen, F. 2025. "A Geometric Modeling of Occam's Razor in Deep Learning". *Information Geometry*, Special Issue: Half a Century of Information Geometry, Part 2. URL:https://doi.org/10.1007/s41884-025-00167-2.

Alon, U., Zilberstein, M., Levy, O., dan Yahav, E. 2019. "code2vec: Learning Distributed Representations of Code". *Proceedings of the ACM on Programming Languages (POPL 2019)*. Cascais, 13-19 Januari. 1–29.

Zhong, C., Zhang, H., Huang, H., Chen, Z., Li, C., Liu, X., dan Li, S. 2024. “DOMICO: Checking conformance between domain models and implementations”. Software: Practice and Experience 54 (4):595–616.

Allamanis, M., Barr, E. T., Devanbu, P., dan Sutton, C. 2018. “A survey of machine learning for big code and naturalness”. ACM Computing Surveys (CSUR) 51 (4):1–37.

Gopalani, P., dan Hu, W. 2025. “What Happens During the Loss Plateau? Understanding Abrupt Learning in Transformers”. *Proceedings of the Thirteenth International Conference on Learning Representations*. Singapura, 24–28 April.

LAMPIRAN

LAMPIRAN 1 Kode Etik

	DEPARTEMEN TEKNIK INFORMATIKA FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS INSTITUT TEKNOLOGI SEPULUH NOPEMBER	Nama Dokumen	:	FORM PERNYATAAN PENGGUNAAN AI
		No. Dokumen	:	001/ST_AI/DTE/V/2025
		Halaman	:	1 dari 2

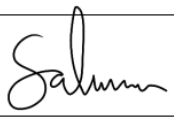
PERNYATAAN KODE ETIK
PENGGUNAAN GENERATIVE AI

Saya yang bertanda tangan di bawah ini:

Nama Lengkap	: Muhammad Budhi Salmanjannah
NRP	: 5025201084
Program Studi	: Teknik Informatika
Judul	: Klasifikasi Peran Komponen Arsitektur Perangkat Lunak Menggunakan Model Codebert
Pembimbing	: Rizky Januar Akbar, S.Kom., M.Eng.

Menyatakan dalam pengerjaan Tugas Akhir dengan judul tersebut di atas:

No.	Pernyataan / *Statement*	Checklist
1	Hanya menggunakan Generative AI sebagai alat bantu untuk memperbaiki tata bahasa pada Tugas Akhir / Thesis / Disertasi saya. Ini tidak digunakan untuk membuat keseluruhan isi pekerjaan saya.	✓
2	Telah memeriksa dan/atau memperbaiki seluruh bagian dari pekerjaan saya yang dibantu oleh Generative AI agar sesuai dengan baku mutu penulisan karya ilmiah.	✓
3	Tidak menggunakan Generative AI untuk pembuatan data primer, grafik dan/atau tabel pada pekerjaan saya.	✓
4	Penulis telah memberikan atribusi/pengakuan terhadap alat AI yang digunakan, secara rinci pada suatu bagian pada lampiran.	✓
5	Buku tugas akhir telah mematuhi Baku Mudu Akademik ITS terkait keaslian karya dan etika penulisan.	✓
6	Penulis memastikan tidak ada plagiarisme, termasuk yang berasal dari penggunaan AI	✓

Tempat	: Surabaya
Tanggal	: 29 Juli 2026
Nama Lengkap	: Muhammad Budhi Salmanjannah
Tanda tangan	: 

LAMPIRAN 2 Laporan Klasifikasi Per-Kelas

Pre-Trained CodeBERT

```
,precision,recall,f1-score,support
AS,0.13099041533546327,1.0,0.23163841807909605,41.0
Command,0.0,0.0,0.0,8.0
```

```
4.  Controller,0.0,0.0,0.0,81.0
5.  DS,0.0,0.0,0.0,10.0
6.  Entity,0.0,0.0,0.0,19.0
7.  External,0.0,0.0,0.0,68.0
8.  Query,0.0,0.0,0.0,17.0
9.  Repository,0.0,0.0,0.0,10.0
10. Storage,0.0,0.0,0.0,35.0
11. VO,0.0,0.0,0.0,24.0
12. accuracy,0.13099041533546327,0.13099041533546327,0.13099041533546327,0.13099041533546327
13. macro avg,0.013099041533546327,0.1,0.023163841807909605,313.0
14. weighted avg,0.01715848890975717,0.13099041533546327,0.030342412591830473,313.0
```

Pre-Trained EL-CodeBERT

```
1.  ,precision,recall,f1-score,support
2.  AS,0.13099041533546327,1.0,0.23163841807909605,41.0
3.  Command,0.0,0.0,0.0,8.0
4.  Controller,0.0,0.0,0.0,81.0
5.  DS,0.0,0.0,0.0,10.0
6.  Entity,0.0,0.0,0.0,19.0
7.  External,0.0,0.0,0.0,68.0
8.  Query,0.0,0.0,0.0,17.0
9.  Repository,0.0,0.0,0.0,10.0
10. Storage,0.0,0.0,0.0,35.0
11. VO,0.0,0.0,0.0,24.0
12. accuracy,0.13099041533546327,0.13099041533546327,0.13099041533546327,0.13099041533546327
13. macro avg,0.013099041533546327,0.1,0.023163841807909605,313.0
14. weighted avg,0.01715848890975717,0.13099041533546327,0.030342412591830473,313.0
```

Finetuned CodeBERT

```
1.  ,precision,recall,f1-score,support
2.  AS,0.6129032258064516,0.4634146341463415,0.5277777777777778,41.0
3.  Command,0.5454545454545454,0.75,0.631578947368421,8.0
4.  Controller,0.8488372093023255,0.9012345679012346,0.874251497005988,81.0
5.  DS,0.5,0.7,0.5833333333333334,10.0
6.  Entity,0.6086956521739131,0.7368421052631579,0.6666666666666666,19.0
```

```
External,0.726027397260274,0.7794117647058824,0.75177304964539,68.0
Query,0.46153846153846156,0.35294117647058826,0.4,17.0
Repository,0.8333333333333334,0.5,0.625,10.0
Storage,0.8285714285714286,0.8285714285714286,0.8285714285714286,35.0
VO,0.6666666666666666,0.5833333333333334,0.6222222222222222,24.0
accuracy,0.7220447284345048,0.7220447284345048,0.7220447284345048,0.7220447284345048
macro avg,0.66320279201074,0.6595749010391966,0.6511174922591227,313.0
weighted avg,0.7200097449191035,0.7220447284345048,0.7160057266279269,313.0
```

Finetuned EL-CodeBERT

```
,precision,recall,f1-score,support
AS,0.509090909090909,0.6829268292682927,0.5833333333333334,41.0
Command,0.0,0.0,0.0,8.0
Controller,0.8352941176470589,0.8765432098765432,0.8554216867469879,81.0
DS,1.0,0.3,0.46153846153846156,10.0
Entity,0.6666666666666666,0.10526315789473684,0.18181818181818182,19.0
External,0.68,0.75,0.7132867132867133,68.0
Query,0.16666666666666666,0.058823529411764705,0.08695652173913043,17.0
Repository,0.6,0.6,0.6,10.0
Storage,0.8055555555555556,0.8285714285714286,0.8169014084507042,35.0
VO,0.375,0.625,0.46875,24.0
accuracy,0.65814696485623,0.65814696485623,0.65814696485623,0.65814696485623
macro avg,0.5638273915626857,0.4827128155022765,0.4768006306913513,313.0
weighted avg,0.6500511030242283,0.65814696485623,0.6297094569710515,313.0
```

LAMPIRAN 2 Analisis Sampel Salah Klasifikasi

project,file_path,file_name,content,label,predicted_label

CodeBERT

```
Movies-Finder-master,Movies-Finder-master\android\src\main\java\com\raulh82vlc\MoviesFinder\di\modules\MovieDetailsModule.java,MovieDetailsModule.java,"public class MovieDetailsModule {

    @Provides
```

```
4.      @ActivityScope
5.      MovieDetailsPresenter provideMovieDetailsPresenter(MovieDetailsPresenterImpl presenter)
    {
6.          return presenter;
7.      }
8.
9.      @Provides
10.     @ActivityScope
11.     GetMovieDetailsInteractor provideGetMovieDetailsInteractor(GetMovieDetailsInteractorImpl
    interactor) {
12.         return interactor;
13.     }
14. }",AS,Controller
15. Music-App-master,Music-App-
    master\app\src\main\java\com\vpaliy\melophile\di\component\PlayerComponent.java,PlayerCompon
    ent.java,"@Component(dependencies = ApplicationComponent.class,
16.         modules = PlaybackModule.class)
17. public interface PlayerComponent {
18.   void inject(MusicPlaybackService service);
19.
20.   void inject(TrackFragment fragment);
21. }",AS,External
```

EL-CodeBERT

```
1.  Kamus-Inggris-Indonesia-master,Kamus-Inggris-Indonesia-
    master\app\src\main\java\com\paperplanes\mykamus\domain\usecases\ClearBookmarksUseCase.java,
    ClearBookmarksUseCase.java,"public class ClearBookmarksUseCase extends
    UseCase<ClearBookmarksUseCase.Params, ClearBookmarksUseCase.Result> {
2.      private DictionaryDataSource mDataSource;
3.      @Inject
4.      public ClearBookmarksUseCase(DictionaryDataSource dataSource) {
5.          mDataSource = dataSource;
6.      }
7.      @Override
8.      protected void execute(Params params) {
9.          mDataSource.clearBookmark(params.mode);
10.         getCallback().onSuccess(new Result());
11.     }
12.     public static class Params implements UseCase.Params {
```

```
13.         TranslationMode mode;
14.
15.         public Params(TranslationMode mode) {
16.             this.mode = mode;
17.         }
18.     }
19.     public static class Result implements UseCase.Result {}
20. }",Command,AS
21. ShoppingList-
    master,app\src\main\java\com\jshvarts\shoppinglist\common\domain\model\ShoppingList.java,Sho
    ppingList.java,"public class ShoppingList {
22.     private String id;
23.     // Only List is supported by Firebase. Set would be more is more applicable though
24.     private List<ShoppingListItem> items = new ArrayList<>();
25.     private Long timestamp;
26.     public ShoppingList() {
27.         // required by Firebase
28.     }
29.     public ShoppingList(List<ShoppingListItem> items) {
30.         this.items = items;
31.     }
32.     @Exclude
33.     public String getId() {
34.         return id;
35.     }
36.     public void setId(String id) {
37.         this.id = id;
38.     }
39.     public List<ShoppingListItem> getItems() {
40.         return items;
41.     }
42.     public void setItems(List<ShoppingListItem> items) {
43.         this.items = items;
44.     }
45.     public Map<String, String> getTimestamp() {
46.         return ServerValue.TIMESTAMP;
47.     }
48.     @Exclude
```

```
49.     public Long getTimestampLong() {
50.         return timestamp;
51.     }
52. }",Entity,VO
```

LAMPIRAN 3 Sampel Dataset

Entity

```
1.  QuizAlarmClock-
    master,app\src\main\java\linc\com\alarmclockforprogrammers\domain\models\Alarm.java,Alarm.ja
    va,"public class Alarm {
2.
3.      private int id;
4.      private int hour;
5.      private int minute;
6.      private int difficult;
7.      private boolean containsTask;
8.      private boolean enable;
9.      private String language;
10.     private String label;
11.     private String songPath;
12.     private boolean[] selectedDays;
13.
14.     public int getId() {
15.         return id;
16.     }
17.     public void setId(int id) {
18.         this.id = id;
19.     }
20.     public int getHour() {
21.         return hour;
22.     }
23.     public void setHour(int hour) {
24.         this.hour = hour;
25.     }
26.     public int getMinute() {
27.         return minute;
```

```
28.     }
29.     public void setMinute(int minute) {
30.         this.minute = minute;
31.     }
32.     public String getLanguage() {
33.         return language;
34.     }
35.     public void setLanguage(String language) {
36.         this.language = language;
37.     }
38.     public int getDifficult() {
39.         return difficult;
40.     }
41.     public void setDifficult(int difficult) {
42.         this.difficult = difficult;
43.     }
44.     public boolean isContainsTask() {
45.         return containsTask;
46.     }
47.     public void setContainsTask(boolean containsTask) {
48.         this.containsTask = containsTask;
49.     }
50.     public boolean isEnable() {
51.         return enable;
52.     }
53.     public void setEnable(boolean enable) {
54.         this.enable = enable;
55.     }
56.     public String getLabel() {
57.         return label;
58.     }
59.     public void setLabel(String label) {
60.         this.label = label;
61.     }
62.     public String getSongPath() {
63.         return songPath;
64.     }
```

```
65.     public void setSongPath(String songPath) {
66.         this.songPath = songPath;
67.     }
68.     public boolean[] getSelectedDays() {
69.         return selectedDays;
70.     }
71.     public void setSelectedDays(boolean[] selectedDays) {
72.         this.selectedDays = selectedDays;
73.     }
74. }",Entity
```

Value Object

```
1.  pet-visits-service-
    main,src\main\java\visits\domain\VisitStatus.java,VisitStatus.java,"public enum VisitStatus
    {
2.      PENDING,
3.      APPROVED,
4.      REJECTED
5.  }",VO
```

Repository

```
1.  Neuronizer-
    master,app\src\main\java\de\djuelg\neuronizer\domain\repository\NoteRepository.java,NoteRepo
    sitory.java,"public interface NoteRepository {
2.      List<Note> getAll();
3.      Optional<Note> get(String uuid);
4.      boolean insert(Note note);
5.      void update(Note updatedNote);
6.      void delete(Note deletedNote);
7.  }",Repository
```

Domain Service

```
1.  LittleLight-master,LittleLight-
    master\littlelight\src\main\java\com\jam01\littlelight\domain\inventory\DestinyInventoryServ
    ice.java,DestinyInventoryService.java,"public interface DestinyInventoryService {
2.      void synchronizeInventoryFor(Account anAccount, InventoryRepository repository);
3.
4.      void transferItem(String anItemId, String toBagId, Inventory inventory, Account
    anAccount);
5.
6.      boolean equip(String anItemId, Character onCharacter, Account anAccount);
7.
8.      boolean unequip(String anItemId, Character onCharacter, Account anAccount);
9.
10.     Collection<ItemType> getExoticTypes();
11. }",DS
```

Application Service

```
1.  LittleLight-master,LittleLight-
    master\littlelight\src\main\java\com\jam01\littlelight\application\ActivityService.java,Acti
    vityService.java,"public class ActivityService {
2.      private DestinyActivityService destinyService;
3.      private User user;
4.
5.      public ActivityService(DestinyActivityService destinyService, User user) {
6.          this.destinyService = destinyService;
7.          this.user = user;
8.      }
9.
```

```
10.     public Account ofAccount(AccountId anAccountId) {
11.         return destinyService.getFor(user.ofId(anAccountId));
12.     }
13. }",AS
```

Command

```
1.  Kamus-Inggris-Indonesia-master,Kamus-Inggris-Indonesia-
    master\app\src\main\java\com\paperplanes\mykamus\domain\usecases\ClearBookmarksUseCase.java,
    ClearBookmarksUseCase.java,"public class ClearBookmarksUseCase extends
    UseCase<ClearBookmarksUseCase.Params, ClearBookmarksUseCase.Result> {
2.      private DictionaryDataSource mDataSource;
3.      @Inject
4.      public ClearBookmarksUseCase(DictionaryDataSource dataSource) {
5.          mDataSource = dataSource;
6.      }
7.      @Override
8.      protected void execute(Params params) {
9.          mDataSource.clearBookmark(params.mode);
10.         getCallback().onSuccess(new Result());
11.     }
12.     public static class Params implements UseCase.Params {
13.         TranslationMode mode;
14.
15.         public Params(TranslationMode mode) {
16.             this.mode = mode;
17.         }
18.     }
19.     public static class Result implements UseCase.Result {}
20. }",Command
```

Query

```
1.  LittleLight-master,LittleLight-
    master\littlelight\src\main\java\com\bungie\netplatform\destiny\representation\DataResponse.
    java,DataResponse.java,"public class DataResponse<T> {
2.
3.      @Expose
4.      private T data;
```

```
5.
6.      public T getData() {
7.          return data;
8.      }
9.      public void setData(T data) {
10.         this.data = data;
11.     }
12. }",Query
```

Controller

```
1.  LittleLight-master,LittleLight-
    master\androidadapter\app\src\main\java\com\jam01\littlelight\adapter\android\utils\NonScrol
    lableListView.java,NonScrollableListView.java,"public class NonScrollableListView extends
    ListView {
2.      public NonScrollableListView(Context context) {
3.          super(context);
4.      }
5.      public NonScrollableListView(Context context, AttributeSet attrs) {
6.          super(context, attrs);
7.      }
8.      public NonScrollableListView(Context context, AttributeSet attrs, int defStyle) {
9.          super(context, attrs, defStyle);
10.     }
11.     @Override
12.     public void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
13.         int heightMeasureSpec_custom = View.MeasureSpec.makeMeasureSpec(
14.                 Integer.MAX_VALUE >> 2, View.MeasureSpec.AT_MOST);
15.         super.onMeasure(widthMeasureSpec, heightMeasureSpec_custom);
16.         ViewGroup.LayoutParams params = getLayoutParams();
17.         params.height = getMeasuredHeight();
18.     }
19. }",Controller
```

Storage

```
1.  Mediateka-master,Mediateka-
    master\app\src\main\java\com\ru\devit\mediateka\MediatekaApp.java,MediatekaApp.java,"public
    class MediatekaApp extends Application {
```

```
2.      private static ComponentsManager componentsManager;
3.      private static MediatekaDatabase database;
4.      private static final String DATABASE_NAME = ""mediateka_db"";
5.
6.      @Override
7.      public void onCreate() {
8.          super.onCreate();
9.          initComponentsManager();
10.         initAppComponent();
11.         RxJavaPlugins.setErrorHandler(Functions.emptyConsumer());
12.     }
13.     public static ComponentsManager getComponentsManager(){
14.         return componentsManager;
15.     }
16.     public void setConnectionListener(SyncConnectionListener connectionListener){
17.         ConnectionReceiver.connectionListener = connectionListener;
18.     }
19.     public MediatekaDatabase getDatabaseInstance(){
20.         if (database == null){
21.             database = Room.databaseBuilder(this , MediatekaDatabase.class ,
    DATABASE_NAME).build();
22.         }
23.         return database;
24.     }
25.     private void initComponentsManager(){
26.         componentsManager = new ComponentsManager(this);
27.     }
28.     private void initAppComponent(){
29.         componentsManager.getAppComponent();
30.     }
31. }",Storage
```

External

```
1.  Mediateka-master,Mediateka-
    master\app\src\main\java\com\ru\devit\mediateka\data\ConnectionReceiver.java,ConnectionRecei
    ver.java,"public class ConnectionReceiver extends BroadcastReceiver {
2.      public static SyncConnectionListener connectionListener;
```

```
3.
4.      @Override
5.      public void onReceive(Context context, Intent arg) {
6.          boolean isConnected = checkConnection(context);
7.
8.          if (connectionListener != null) {
9.              connectionListener.onNetworkConnectionChanged(isConnected);
10.         }
11.     }
12.     public boolean isInternetConnected(Context context){
13.         return checkConnection(context);
14.     }
15.     private boolean checkConnection(Context context) {
16.         ConnectivityManager cm = (ConnectivityManager)
    context.getSystemService(Context.CONNECTIVITY_SERVICE);
17.         NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
18.         return activeNetwork != null && activeNetwork.isConnectedOrConnecting();
19.     }
20. }",External
```

LAMPIRAN 4 Bukti Eksperimen CUDA

CodeBERT

```
1.  {'loss': '2.136', 'grad_norm': '12.14', 'learning_rate': '4.9e-05', 'epoch': '0.3185'}
2.  {'loss': '1.953', 'grad_norm': '7.609', 'learning_rate': '4.667e-05', 'epoch': '0.6369'}
3.  {'loss': '1.394', 'grad_norm': '11.26', 'learning_rate': '4.327e-05', 'epoch': '0.9554'}
4.    _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
5.  {'eval_loss': '1.193', 'eval_accuracy': '0.6134', 'eval_f1': '0.5719', 'eval_precision':
    '0.5578', 'eval_recall': '0.6134', 'eval_runtime': '12.59', 'eval_samples_per_second':
    '24.87', 'eval_steps_per_second': '3.178', 'epoch': '1'}
6.  {'loss': '1.121', 'grad_norm': '7.836', 'learning_rate': '3.986e-05', 'epoch': '1.274'}
7.  {'loss': '0.9902', 'grad_norm': '16.56', 'learning_rate': '3.646e-05', 'epoch': '1.592'}
8.  {'loss': '0.9217', 'grad_norm': '7.514', 'learning_rate': '3.306e-05', 'epoch': '1.911'}
9.    _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
10. {'eval_loss': '0.9035', 'eval_accuracy': '0.6997', 'eval_f1': '0.6709', 'eval_precision':
    '0.6951', 'eval_recall': '0.6997', 'eval_runtime': '12.83', 'eval_samples_per_second':
    '24.4', 'eval_steps_per_second': '3.118', 'epoch': '2'}
11. {'loss': '0.7228', 'grad_norm': '13.12', 'learning_rate': '2.966e-05', 'epoch': '2.229'}
```

```
12. {'loss': '0.6052', 'grad_norm': '7.244', 'learning_rate': '2.626e-05', 'epoch': '2.548'}

13. {'loss': '0.6985', 'grad_norm': '12.14', 'learning_rate': '2.286e-05', 'epoch': '2.866'}

14. {'eval_loss': '0.9594', 'eval_accuracy': '0.6901', 'eval_f1': '0.6705', 'eval_precision':
    '0.6698', 'eval_recall': '0.6901', 'eval_runtime': '12.85', 'eval_samples_per_second':
    '24.36', 'eval_steps_per_second': '3.113', 'epoch': '3'}

15. {'loss': '0.5096', 'grad_norm': '5.991', 'learning_rate': '1.946e-05', 'epoch': '3.185'}

16. {'loss': '0.5096', 'grad_norm': '5.991', 'learning_rate': '1.946e-05', 'epoch': '3.185'}

17. {'loss': '0.4431', 'grad_norm': '15.87', 'learning_rate': '1.605e-05', 'epoch': '3.503'}

18. {'loss': '0.3877', 'grad_norm': '15.29', 'learning_rate': '1.265e-05', 'epoch': '3.822'}

19. {'eval_loss': '0.9447', 'eval_accuracy': '0.722', 'eval_f1': '0.7151', 'eval_precision':
    '0.7227', 'eval_recall': '0.722', 'eval_runtime': '12.83', 'eval_samples_per_second':
    '24.4', 'eval_steps_per_second': '3.118', 'epoch': '4'}

20. {'loss': '0.3675', 'grad_norm': '9.577', 'learning_rate': '9.252e-06', 'epoch': '4.14'}

21. {'loss': '0.1989', 'grad_norm': '10.06', 'learning_rate': '5.85e-06', 'epoch': '4.459'}

22. {'loss': '0.2491', 'grad_norm': '4.29', 'learning_rate': '2.449e-06', 'epoch': '4.777'}

23. {'eval_loss': '0.9868', 'eval_accuracy': '0.722', 'eval_f1': '0.716', 'eval_precision':
    '0.72', 'eval_recall': '0.722', 'eval_runtime': '12.88', 'eval_samples_per_second': '24.3',
    'eval_steps_per_second': '3.106', 'epoch': '5'}

24. {'train_runtime': '850.2', 'train_samples_per_second': '7.345', 'train_steps_per_second':
    '0.923', 'train_loss': '0.8183', 'epoch': '5'}

25. 100%|785/785 [14:11<00:00,  1.08s/it]
```

EL-CodeBERT

```
1.  {'loss': '2.143', 'grad_norm': '7.128', 'learning_rate': '4.9e-05', 'epoch': '0.3185'}

2.  {'loss': '1.946', 'grad_norm': '4.256', 'learning_rate': '4.667e-05', 'epoch': '0.6369'}

3.  {'loss': '1.646', 'grad_norm': '4.336', 'learning_rate': '4.327e-05', 'epoch': '0.9554'}

4.   _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])

5.  {'eval_loss': '1.568', 'eval_accuracy': '0.5048', 'eval_f1': '0.4524', 'eval_precision':
    '0.4197', 'eval_recall': '0.5048', 'eval_runtime': '56.06', 'eval_samples_per_second':
    '5.584', 'eval_steps_per_second': '0.714', 'epoch': '1'}

6.  {'loss': '1.544', 'grad_norm': '4.642', 'learning_rate': '3.986e-05', 'epoch': '1.274'}

7.  {'loss': '1.41', 'grad_norm': '10.84', 'learning_rate': '3.646e-05', 'epoch': '1.592'}

8.  {'loss': '1.208', 'grad_norm': '8.51', 'learning_rate': '3.306e-05', 'epoch': '1.911'}

9.    _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])

10. {'eval_loss': '1.172', 'eval_accuracy': '0.6294', 'eval_f1': '0.5728', 'eval_precision':
    '0.5542', 'eval_recall': '0.6294', 'eval_runtime': '46.69', 'eval_samples_per_second':
    '6.704', 'eval_steps_per_second': '0.857', 'epoch': '2'}

11. {'loss': '1.04', 'grad_norm': '8.799', 'learning_rate': '2.966e-05', 'epoch': '2.229'}

12. {'loss': '0.9732', 'grad_norm': '5.163', 'learning_rate': '2.626e-05', 'epoch': '2.548'}

13. {'loss': '0.974', 'grad_norm': '10.16', 'learning_rate': '2.286e-05', 'epoch': '2.866'}

14. {'eval_loss': '1.12', 'eval_accuracy': '0.6422', 'eval_f1': '0.589', 'eval_precision':
```

```
    '0.6046', 'eval_recall': '0.6422', 'eval_runtime': '44.92', 'eval_samples_per_second':
    '6.968', 'eval_steps_per_second': '0.89', 'epoch': '3'}
15. {'loss': '0.852', 'grad_norm': '8.604', 'learning_rate': '1.946e-05', 'epoch': '3.185'}
16. {'loss': '0.7558', 'grad_norm': '18.43', 'learning_rate': '1.605e-05', 'epoch': '3.503'}
17. {'loss': '0.7293', 'grad_norm': '10.57', 'learning_rate': '1.265e-05', 'epoch': '3.822'}
18.   _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
19. {'eval_loss': '1.041', 'eval_accuracy': '0.6645', 'eval_f1': '0.6246', 'eval_precision':
    '0.6461', 'eval_recall': '0.6645', 'eval_runtime': '44.63', 'eval_samples_per_second':
    '7.013', 'eval_steps_per_second': '0.896', 'epoch': '4'}
20. {'loss': '0.7275', 'grad_norm': '17.78', 'learning_rate': '9.252e-06', 'epoch': '4.14'}
21. {'loss': '0.5521', 'grad_norm': '5.397', 'learning_rate': '5.85e-06', 'epoch': '4.459'}
22. {'loss': '0.6199', 'grad_norm': '7.777', 'learning_rate': '2.449e-06', 'epoch': '4.777'}
23.   _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
24. {'eval_loss': '1.036', 'eval_accuracy': '0.6581', 'eval_f1': '0.6297', 'eval_precision':
    '0.6501', 'eval_recall': '0.6581', 'eval_runtime': '44.12', 'eval_samples_per_second':
    '7.094', 'eval_steps_per_second': '0.907', 'epoch': '5'}
25. {'train_runtime': '1021', 'train_samples_per_second': '6.117', 'train_steps_per_second':
    '0.769', 'train_loss': '1.12', 'epoch': '5'}
26. 100%|785/785 [17:01<00:00,  1.30s/it]
```

LAMPIRAN 5 Daftar Repositori GitHub

REPOSITORI PENELITIAN

https://github.com/salmanhermana/TA-CodeBERT

REPOSITORI DATASET

https://github.com/amezcua/Beak

https://github.com/mohadian/CleanArchitecture

https://github.com/Gaket/Earthquakes

https://github.com/luissouza/flight-booking-clean-architecture

https://github.com/abdularis/Kamus-Inggris-Indonesia

https://github.com/jam01/LittleLight

https://github.com/PopPsyA/Mediateka

https://github.com/vpaliy/Music-App

https://github.com/lincollincol/QuizAlarmClock

https://github.com/drulabs/RecipePuppy

https://github.com/esoxjem/movieguide

https://github.com/raulh82vlc/Movies-Finder

https://github.com/alexmarqs/musicweather-api-clean-architecture

https://github.com/djuelg/Neuronizer

https://github.com/fortiss-cce/pet-visits-service

https://github.com/flipper83/protohipster

https://github.com/fiveagency/Reedly

https://github.com/yemyatthu1990/RestaurantBooker

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Penulis dilahirkan di Surabaya, 14 Januari 2002, merupakan anak terakhir dari 4 bersaudara. Penulis telah menempuh pendidikan formal yaitu di TK Ya Bunaya Surabaya, SD Luqman Al Hakim, SMP Al Hikmah Surabaya dan SMA 2 Muhammadiyah Surabaya. Setelah lulus dari SMA tahun 2020, Penulis diterima di Departemen Teknik Informatika FTEIC - ITS pada tahun 2020 dan terdaftar dengan NRP 5025201084.