DETEKSI PLAGIARISME KODE PROGRAM MENGGUNAKAN GRAPH NEURAL NETWORK BERBASIS ABSTRACT SYNTAXT TREE

SKRIPSI

DISUSUN OLEH

FITRA AFFANDI HASIBUAN

2209020201

PROGRAM STUDI TEKNOLOGI INFORMASI

FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI

UNIVERSITAS MUHAMMADIYAH SUMATERA UTARA

MEDAN

2026

DETEKSI PLAGIARISME KODE PROGRAM MENGGUNAKAN GRAPH NEURAL NETWORK BERBASIS ABSTRACT SYNTAXT TREE

SKRIPSI

Diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer (S.Kom) dalam program studi Teknologi Informasi. Pada Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Muhammadiyah Sumatera Utara

FITRA AFFANDI HASIBUAN

NPM. 2209020201

PROGRAM STUDI TEKNOLOGI INFORMASI

FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI

UNIVERSITAS MUHAMMADIYAH SUMATERA UTARA

MEDAN

2026

LEMBAR PENGESAHAN

Judul Skripsi	:	Deteksi Plagiarisme Kode Program Menggunakan Graph Neural Network Berbasis Abstract Syntaxt Tree
Nama Mahasiswa	:	Fitra Affandi Hasibuan
NPM	:	2209020201
Program Studi	:	Teknologi Informasi

Menyetujui
Komisi Pembimbing

(Dr. Al-Khowarizmi, S.Kom., M.Kom.)
NIDN. 0127099201

Ketua Program Studi

(Fatma Sari Hutagalung, S.Kom., M.Kom)
NIDN. 0117019301

Dekan

(Dr. Al-Khowarizmi, S.Kom., M.Kom.)
NIDN. 0127099201

PERNYATAAN ORISINALITAS

DETEKSI PLAGIARISME KODE PROGRAM MENGGUNAKAN GRAPH NEURAL NETWORK BERBASIS ABSTRACT SYNTAXT TREE

SKRIPSI

Saya menyatakan bahwa karya tulis ini adalah hasil karya sendiri, kecuali beberapa kutipan dari ringkasan yang masing-masing disebutkan sumbernya.

Medan, Maret 2026

Yang membuat pernyataan

Fitra Alfandi Hasibuan

NPM. 2209020201

PERNYATAAN PERSETUJUAN PUBLIKASI
KARYA ILMIAH UNTUK KEPENTINGAN
AKADEMIS

Saya sivitas akademia Universitas Muhammadiyah Sumatera Utara, saya bertanda tangan dibawah ini:

Nama	:	Fitra Affandi Hasibuan
NPM	:	2209020201
Pogram Studi	:	Teknologi Informasi
Karya Ilmiah	:	Skripsi

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Muhammadiyah Sumatera Utara Hak Bebas Royalty Non-Eksekusif *(Non-Exclusive Royalty free Right)* atas penelitian skripsi saya yang berjudul:

DETEKSI PLAGIARISME KODE PROGRAM MENGGUNAKAN GRAPH NEURAL NETWORK BERBASIS ABSTRACT SYNTAXT TREE

Beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalty Non-Eksekutif ini, Universitas Muhammadiyah Sumatera Utara berhak menyimpan, mengalih media, memformat, mengelolah dalam bentuk database, merawat dan mempublikasikan Skripsi saya ini tanpa meminta izin dari saya selama tetap mencantumkan nama saya sebagai penulis dan sebagai pemegang dan atau sebagai pemilik hak cipta.

Demikian pernyataan ini dibuat dengan sebenarnya.

Medan, Maret 2026

Yang membuat pernyataan

Fitra Affandi Hasibuan

NPM. 2209020201

RIWAYAT HIDUP

DATA PRIBADI

Nama Lengkap	:	Fitra Affandi Hasibuan
Tempat dan Tanggal Lahir	:	Medan, 17 Desember 2002
Alamat Rumah	:	JL. Murni No 14 C, Kel. Tanjung Rejo, Kec. Medan Sunggal
Telepon/Faks/HP	:	082160949056
E-mail	:	fitraaffandihasibuan@gmail.com
Instansi Tempat Kerja	:	-
Alamat Kantor	:	-

DATA PENDIDIKAN

SD	:	SD Negeri 106835		TAMAT	:	2015
SMP	:	SMP Negeri 2 Tanjung Morawa		TAMAT	:	2018
SMA	:	SMA Swasta Dwi Tunggal		TAMAT	:	2021

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Tuhan yang Maha Esa atas segala rahmat, hidayah, dan karunia-Nya, sehingga penulis dapat menyelesaikan skripsi yang berjudul **DETEKSI PLAGIARISME KODE PROGRAM MENGGUNAKAN GRAPH NEURAL NETWORK BERBASIS ABSTRACT SYNTAXT TREE** ini dengan baik. Skripsi ini disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer pada Program Studi Teknologi Informasi, Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Muhammadiyah Sumatera Utara.

Penulis tentunya berterima kasih kepada berbagai pihak dalam dukungan serta doa dalam penyelesaian skripsi. Penulis juga mengucapkan terima kasih kepada:

1. Bapak Prof. Dr. Agussani, M.AP., Rektor Universitas Muhammadiyah Sumatera Utara (UMSU).
2. Bapak Assoc. Prof. Dr. Al-Khowarizmi, M.Kom., Dekan Fakultas Ilmu Komputer dan Teknologi Informasi sekaligus Dosen Pembimbing, atas segala bimbingan dan arahan yang telah diberikan kepada penulis.
3. Ibu Dr. Firahmi Rizky, S.Kom., M.Kom., selaku Wakil Dekan I Fakultas Ilmu Komputer dan Teknologi Informasi (FIKTI) UMSU.
4. Bapak Mhd Basri, S.Si., M.Kom., selaku Wakil Dekan III Fakultas Ilmu Komputer dan Teknologi Informasi (FIKTI) UMSU.
5. Ibu Fatma Sari Hutagalung, M.Kom., Ketua Program Studi Teknologi Informasi.
6. Bapak Okvi Nugroho, S.Kom., M.Kom., Sekretaris Program Studi Teknologi Informasi.
7. Seluruh Bapak/Ibu Dosen dan Staf di Fakultas Ilmu Komputer dan Teknologi Informasi yang telah membantu penulis selama masa perkuliahan dan proses administrasi penyusunan proposal skripsi.

8. Kedua orang tua tercinta, Bapak Ali Sofyan Hasibuan dan Ibu Lily Handayani, atas doa tulus, kasih sayang, dan dukungan luar biasa yang menjadikan kekuatan utama bagi penulis.

Penulis menyadari bahwa proposal skripsi ini masih jauh dari sempurna, baik dari segi penyusunan maupun isi, karena keterbatasan pengetahuan dan pengalaman penulis. Oleh karena itu, penulis sangat mengharapkan kritik dan saran yang membangun demi perbaikan dan penyempurnaan di masa mendatang. Semoga skripsi ini dapat bermanfaat bagi pembaca dan dapat memberikan kontribusi untuk pengembangan ilmu pengetahuan.

ABSTRAK

Plagiarisme kode program merupakan permasalahan yang sering terjadi dalam dunia pendidikan dan pengembangan perangkat lunak, yang sulit dideteksi secara akurat menggunakan pendekatan berbasis teks. Metode konvensional seperti *Term Frequency–Inverse Document Frequency* (TF-IDF) dan *cosine similarity* cenderung hanya memperhatikan kesamaan token, sehingga kurang efektif dalam menangani perubahan struktur kode. Oleh karena itu, penelitian ini bertujuan untuk mengembangkan sistem deteksi plagiarisme kode program berbasis struktur menggunakan *Abstract Syntax Tree* (AST) dan *Graph Neural Network* (GNN). Metode yang digunakan melibatkan proses parsing kode program menjadi AST, kemudian direpresentasikan dalam bentuk *graph* dan diproses menggunakan model GNN dalam skema *pairwise*. Selain itu, dilakukan perbandingan dengan metode *baseline* berbasis TF-IDF dan *cosine similarity* untuk mengevaluasi kinerja model. Dataset yang digunakan terdiri dari data rekayasa dan data nyata yang dibagi menjadi data pelatihan dan pengujian. Hasil penelitian menunjukkan bahwa model GNN memiliki performa yang sangat baik dengan nilai *accuracy* sebesar 0.9946, *precision* sebesar 0.9949, *recall* sebesar 0.9974, dan *F1-score* sebesar 0.9962, sedangkan metode *baseline* hanya mencapai *accuracy* sebesar 0.7392 dan *recall* sebesar 0.6343. Hal ini menunjukkan bahwa model GNN mampu mendeteksi plagiarisme secara lebih efektif, terutama dalam menangani perubahan struktur kode. Dengan demikian, dapat disimpulkan bahwa pendekatan berbasis struktur menggunakan AST dan GNN lebih unggul dibandingkan pendekatan berbasis teks dalam mendeteksi plagiarisme kode program.

Kata Kunci: Plagiarisme Kode, *Graph Neural Network*, *Abstract Syntax Tree*, *Cosine Similarity*, Deteksi Kemiripan.

ABSTRACT

Code plagiarism is a common issue in education and software development, which is difficult to detect accurately using text-based approaches. Conventional methods such as Term Frequency–Inverse Document Frequency (TF-IDF) and cosine similarity tend to focus only on token similarity, making them less effective in handling structural changes in code. Therefore, this study aims to develop a structure-based code plagiarism detection system using Abstract Syntax Tree (AST) and Graph Neural Network (GNN). The proposed method involves parsing source code into AST, representing it as a graph, and processing it using a GNN model in a pairwise scheme. In addition, a comparison is conducted with a baseline method based on TF-IDF and cosine similarity to evaluate model performance. The dataset used consists of both synthetic and real data, which are divided into training and testing sets. The results show that the GNN model achieves excellent performance with an accuracy of 0.9946, precision of 0.9949, recall of 0.9974, and F1-score of 0.9962, while the baseline method only achieves an accuracy of 0.7392 and a recall of 0.6343. These results indicate that the GNN model is more effective in detecting plagiarism, especially in handling structural code modifications. Therefore, it can be concluded that the structure-based approach using AST and GNN outperforms text-based approaches in code plagiarism detection.

Keyword: *Code Plagiarism, Graph Neural Network, Abstract Syntax Tree, Cosine Similarity, Similarity Detection*.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi telah menjadikan pemrograman sebagai kompetensi fundamental dalam pendidikan tinggi dan industri perangkat lunak, di mana tugas pemrograman berperan sebagai instrumen utama untuk menilai pemahaman mahasiswa terhadap algoritma, struktur kontrol, serta kemampuan pemecahan masalah. Namun, dalam praktiknya, plagiarisme kode program masih menjadi tantangan serius dalam pendidikan ilmu komputer, khususnya dengan meningkatnya akses terhadap sumber kode digital dan kemudahan berbagi program secara daring (Maertens et al., 2024). Praktik plagiarisme tidak hanya berupa penyalinan langsung (*copy-paste*), tetapi juga melibatkan modifikasi sederhana seperti penggantian nama variabel, perubahan format, atau refactoring ringan yang bertujuan menghindari deteksi (Zakeri-Nasrabadi et al., 2023). Kondisi ini menyulitkan proses pemeriksaan manual, terutama pada kelas dengan jumlah mahasiswa besar, sehingga diperlukan pendekatan deteksi yang lebih objektif dan efisien berbasis analisis struktur kode (D. Yu et al., 2023)

Berbagai metode deteksi plagiarisme kode telah digunakan, namun banyak pendekatan tradisional masih bertumpu pada kemiripan berbasis teks atau token. Pendekatan berbasis teks cenderung sensitif terhadap perubahan permukaan, sehingga mudah dikelabui oleh plagiarisme yang hanya melakukan modifikasi kosmetik tanpa mengubah logika program. Di sisi lain, kajian mutakhir mengenai pengukuran kemiripan kode dan *clone detection* menegaskan bahwa tantangan utama terletak pada pemilihan representasi yang mampu menangkap struktur dan

semantik program, bukan sekadar persamaan karakter atau token (Zakeri-Nasrabadi et al., 2023). Oleh karena itu, diperlukan pendekatan yang mampu menganalisis program pada level struktur logika.

Penelitian ini mengusulkan penggunaan *Abstract Syntaxt Tree* (AST) sebagai representasi kode untuk memperkuat deteksi plagiarisme. AST merepresentasikan struktur program dalam bentuk pohon yang menggambarkan elemen sintaks secara hierarkis, seperti *assignment*, percabangan, perulangan, ekspresi, dan pemanggilan fungsi. Keunggulan AST terletak pada kemampuannya mempertahankan struktur logika program meskipun terjadi perubahan kosmetik seperti spasi, komentar, atau penggantian nama identifier. Dalam ranah pembelajaran mesin untuk kode, representasi struktural seperti AST/*graph* dipandang lebih informatif untuk memodelkan pola kode dibanding representasi teks murni, karena menyediakan konteks hubungan antar komponen program (Dong et al., 2024). Dengan demikian, AST menjadi dasar yang relevan untuk mendeteksi plagiarisme yang berupaya menghindari deteksi melalui perubahan tampilan.

Meskipun AST kuat sebagai representasi struktural, analisis AST memerlukan algoritma yang mampu mempelajari pola hubungan antar *node*. Karena AST dapat dipandang sebagai bentuk khusus dari graph, penelitian ini memilih *Graph Neural Network* (GNN) sebagai pendekatan utama. GNN dirancang untuk memproses data berbentuk graph melalui mekanisme propagasi informasi antar *node*, sehingga efektif untuk mempelajari pola struktural dan hubungan yang terdapat pada representasi *graph* kode (Dong et al., 2024). Selain itu, pendekatan berbasis graph juga digunakan untuk pembelajaran semantik kode yang mendukung

tugas kemiripan dan *clone detection*, sehingga relevan untuk digunakan dalam deteksi plagiarisme yang memiliki konsep dasar "kemiripan kode" (D. Yu et al., 2023).

Berdasarkan pertimbangan tersebut, penelitian ini berfokus pada pengembangan sistem deteksi plagiarisme kode program secara *pairwise* (membandingkan kode A vs kode B) dengan keluaran skor kemiripan dan label dua kelas (plagiat atau tidak plagiat). Skor kemiripan memberikan ukuran kuantitatif yang dapat digunakan untuk menganalisis tingkat kemiripan dan menentukan ambang batas (threshold) keputusan, sedangkan label dua kelas memudahkan interpretasi hasil bagi pengguna. Secara umum, alur sistem adalah: (1) kode program diparsing menjadi AST; (2) AST dikonversi menjadi *graph*; (3) graph diproses menggunakan GNN untuk menghasilkan representasi (*embedding*); (4) *embedding* dibandingkan untuk memperoleh skor kemiripan; dan (5) skor dipetakan menjadi label berdasarkan *threshold* yang ditetapkan. Desain *pairwise* ini sejalan dengan praktik umum pada studi kemiripan kode karena memudahkan evaluasi kinerja model menggunakan metrik klasifikasi seperti akurasi, precision, recall, dan F1-score (Zakeri-Nasrabadi et al., 2023).

Dengan adanya sistem ini, diharapkan proses deteksi plagiarisme kode dapat dilakukan secara lebih cepat, konsisten, dan sulit dimanipulasi oleh perubahan kosmetik, sehingga dapat membantu meningkatkan integritas akademik pada mata kuliah pemrograman. Selain kontribusi akademik melalui pemanfaatan representasi AST dan pemodelan GNN, penelitian ini juga diarahkan untuk menghasilkan prototipe sistem yang memberikan keluaran skor dan keputusan yang dapat dijadikan dasar pemeriksaan lanjutan oleh dosen.

1.2 Rumusan masalah

Berdasarkan latar belakang yang telah diuraikan, penelitian ini berfokus pada perancangan sistem deteksi plagiarisme kode program berbasis *Abstract Syntax Tree* (AST) dan *Graph Neural Network* (GNN) dalam skema *pairwise*. Untuk memperjelas arah penelitian, maka rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana efektivitas struktur AST dalam merepresentasikan karakteristik plagiarisme kode program *Type-1* (*copy-paste* kosmetik) dan *Type-2* (*rename identifier*)?
2. Bagaimana rancangan representasi *graph* yang optimal dari AST kode Python (fitur node dan edge) agar dapat diproses oleh Graph Neural Network?
3. Bagaimana arsitektur model GNN dengan skema *pairwise* untuk menghasilkan skor kemiripan dan klasifikasi biner plagiarisme?
4. Bagaimana performa model AST+GNN dibandingkan dengan metode baseline berdasarkan metrik akurasi, *precision, recall,* dan *F1-score*?
5. Bagaimana penentuan *threshold* skor kemiripan untuk keputusan plagiarisme yang objektif dan konsisten, serta bagaimana *robustness* model pada Type-3 ringan sebagai uji ketahanan?

1.3 Batasan Masalah

Agar penelitian ini tetap terarah dan tidak menyimpang dari tujuan yang telah ditetapkan, diperlukan pembatasan ruang lingkup kajian. Batasan masalah ini ditetapkan untuk memperjelas fokus penelitian serta menghindari perluasan

pembahasan di luar konteks yang direncanakan. Adapun batasan masalah dalam penelitian ini adalah sebagai berikut:

1. Penelitian ini dibatasi pada deteksi plagiarisme kode program secara *pairwise*, yaitu membandingkan dua kode program dalam satu proses analisis, tanpa membahas pendekatan berbasis *clustering* atau deteksi massal.
2. Objek penelitian terbatas pada kode sumber bahasa pemrograman *Python* yang valid secara sintaks.
3. Fokus penelitian diarahkan pada plagiarisme *Type-1* (*copy-paste* kosmetik) dan *Type-2* (*rename identifier*)
4. Representasi kode dibatasi pada *Abstract Syntax Tree* (AST) yang dikonversi ke bentuk *graph* dengan relasi struktural dasar, tanpa menggunakan representasi lain seperti CFG, DFG, atau PDG.
5. Metode yang digunakan terbatas pada *Graph Neural Network* (GNN) untuk menghasilkan skor kemiripan dan klasifikasi biner (plagiarisme/tidak plagiarisme), tanpa membahas klasifikasi multi-kelas atau sistem rekomendasi lanjutan
6. Evaluasi kinerja dibatasi pada analisis metrik klasifikasi dan skor kemiripan, tanpa membahas optimasi sistem untuk implementasi skala industri.

1.4 Tujuan Penelitian

Tujuan penelitian ini dirumuskan untuk memberikan arah yang jelas terhadap tahapan perancangan, implementasi, dan pengujian sistem yang dikembangkan. Adapun tujuan penelitian ini adalah sebagai berikut:

1. Menganalisis karakteristik plagiarisme kode program pada tingkat perubahan kosmetik, khususnya *Type-1* (*copy-paste* kosmetik) dan *Type-2* (*rename identifier*), berdasarkan struktur *Abstract Syntax Tree* (AST).
2. Membangun representasi graph dari AST pada kode program *Python* dengan menentukan *node*, *edge*, dan fitur yang sesuai sehingga dapat diproses oleh model *Graph Neural Network* (GNN).
3. Merancang dan mengimplementasikan model deteksi plagiarisme berbasis GNN dalam skema *pairwise* (kode A vs kode B) yang menghasilkan skor kemiripan dan label dua kelas (plagiat atau tidak plagiat).
4. Mengevaluasi kinerja model AST+GNN dalam mendeteksi plagiarisme kode program pada kasus *Type-1* dan *Type-2*, menggunakan metrik evaluasi akurasi, *precision, recall,* dan *F1-score*, serta membandingkannya dengan pendekatan pembanding (*baseline*) yang lebih sederhana.
5. Menentukan ambang batas (*threshold*) skor kemiripan yang optimal untuk menghasilkan keputusan plagiarisme yang objektif dan konsisten.
6. Menguji ketahanan (*robustness*) model AST+GNN terhadap *Type-3* ringan sebagai robustness test, guna mengetahui pengaruh refactoring sederhana terhadap skor kemiripan dan keputusan label yang dihasilkan sistem.

1.5 Manfaat Penelitian

Penelitian ini diharapkan memberikan kontribusi secara akademik dan praktis dalam pengembangan sistem deteksi plagiarisme kode program berbasis AST dan GNN. Adapun manfaat penelitian ini adalah sebagai berikut:

1. Memberikan kontribusi dalam pengembangan kajian deteksi plagiarisme kode program dengan memanfaatkan representasi *Abstract Syntax Tree*

(AST) dan pendekatan *Graph Neural Network* (GNN), khususnya pada skema deteksi *pairwise*.

2. Memberikan gambaran empiris mengenai efektivitas pendekatan AST+GNN dalam menghadapi plagiarisme berbasis perubahan kosmetik, seperti *copy-paste* dan *rename identifier*, serta ketahanannya terhadap refactoring ringan.
3. Menyediakan prototipe sistem deteksi plagiarisme yang mampu menghasilkan skor kemiripan dan keputusan plagiat/tidak plagiat, sehingga dapat dijadikan alat bantu dalam proses evaluasi tugas pemrograman.
4. Menjadi dasar pengembangan sistem lanjutan yang dapat diintegrasikan ke dalam platform pembelajaran atau sistem akademik, dengan penyesuaian dan pengembangan lebih lanjut.
5. Memberikan wawasan bagi mahasiswa mengenai pentingnya integritas akademik dan orisinalitas dalam penulisan kode program, serta mendorong praktik pembelajaran pemrograman yang lebih jujur dan bertanggung jawab.

BAB II

LANDASAN TEORI

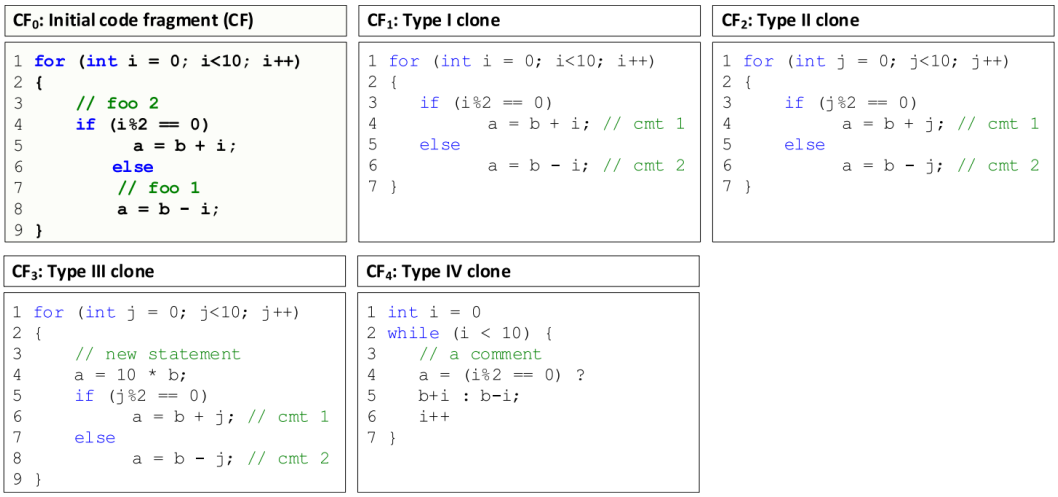
2.1 Plagiarisme Kode Program

Plagiarisme adalah tindakan mengambil atau menggunakan karya orang lain tanpa memberikan pengakuan yang semestinya, yang dalam konteks akademik dipandang sebagai pelanggaran etika dan integritas ilmiah (Maertens et al., 2024). Seiring kemudahan akses ke sumber daya digital, plagiarisme tidak hanya terjadi pada karya tulis, tetapi juga meluas ke kode program, khususnya pada tugas pemrograman di lingkungan pendidikan.

plagiarisme kode program didefinisikan sebagai tindakan penyalinan atau pemodifikasian kode sumber milik pihak lain tanpa atribusi yang sesuai untuk kemudian diakui sebagai karya sendiri. Berbeda dengan plagiarisme teks, plagiarisme kode sering kali lebih sulit dideteksi karena pelaku dapat melakukan berbagai teknik penyamaran sederhana, seperti mengganti nama variabel, atau fungsi (*rename identifier*), mengubah format penulisan, menambahkan komentar, atau melakukan *refactoring* ringan, tanpa mengubah logika maupun keluaran (*output*) dari program tersebut (Maertens et al., 2024). Oleh karena itu, metode deteksi yang hanya mengandalkan perbandingan teks atau *token* menjadi sangat rentan karena tidak mampu menangkap esensi struktur program yang disembunyikan di balik teknik-teknik penyamaran tersebut (Zakeri-Nasrabadi et al., 2023).

Dalam literatur rekayasa perangkat lunak, studi mengenai *code clone detection* memberikan kerangka konseptual yang relevan bagi deteksi plagiarisme kode. Kajian terkini menyajikan klasifikasi kemiripan kode atau *clone* yang umum

digunakan, yaitu *Type-1* hingga *Type-4*, yang membedakan tingkat perubahan terhadap kode sumber mulai dari perubahan kosmetik hingga perbedaan semantik yang kompleks (Zakeri-Nasrabadi et al., 2023). Klasifikasi tersebut sangat berguna untuk merancang strategi deteksi yang sangat tepat pada setiap tingkat atau kategori plagiarisme.



Gambar 2. 1 Ilustrasi Klasifikasi Plagiarisme kode (Type-1 sampai Type-4). Sumber: Zakeri-Nasrabadi et al., (2023).

2.1.1 Type-1 (Copy-Paste Kosmetik)

Pada tingkat ini, kode sumber disalin secara hampir identik dengan perubahan yang terbatas pada aspek kosmetik, seperti spasi, indentasi, atau komentar. Karena struktur dan logika program tetap sama, metode deteksi berbasis struktur atau *Abstract Syntax Tree* (AST) dapat mengidentifikasi kasus ini dengan tingkat akurasi yang tinggi (Zakeri-Nasrabadi et al., 2023).

2.1.2 Type-2 (Rename Identifier)

Pada tingkat ini, pelaku mengganti nama variabel, fungsi, atau parameter tanpa mengubah alur logika program. Meskipun secara permukaan kode tampak berbeda, struktur sintaksisnya tetap setara. Oleh karena itu, pendekatan yang mampu menangkap struktur, seperti AST atau *graph*, memiliki keunggulan dalam mendeteksi *Type-2* dibandingkan dengan metode berbasis *token* saja (D. Yu et al., 2023).

2.1.3 Type-3 (Refactoring Ringan)

Tingkat ini meliputi modifikasi struktur lokal atau refactoring sederhana, seperti mengganti bentuk penugasan ($\alpha = \alpha + 1$ menjadi $\alpha += 1$) atau mengonversi perulangan *for* menjadi *while* dengan alur logika yang ekuivalen. Perubahan tersebut sebagian akan mengubah struktur AST, namun tidak mengubah tujuan maupun keluaran dari program. Dalam penelitian ini, *Type-3* hanya digunakan sebagai uji ketahanan (*robustness test*) untuk mengevaluasi kemampuan model dalam menghadapi perubahan struktural ringan (Büyük & Nizam, 2023).

2.1.4 Type-4 (Semantic Clone)

Pada tingkat ini, dua fungsi atau program menghasilkan keluaran yang sama, namun menggunakan algoritma atau struktur yang berbeda secara signifikan. Deteksi pada tingkat ini memerlukan pemodelan semantik yang sangat mendalam dan umumnya berada di luar ruang lingkup penelitian ini karena kompleksitas perubahannya (Zakeri-Nasrabadi et al., 2023). Fokus penelitian ini dibatasi hingga modifikasi struktur ringan guna menjaga objektivitas evaluasi terhadap model GNN yang dikembangkan.

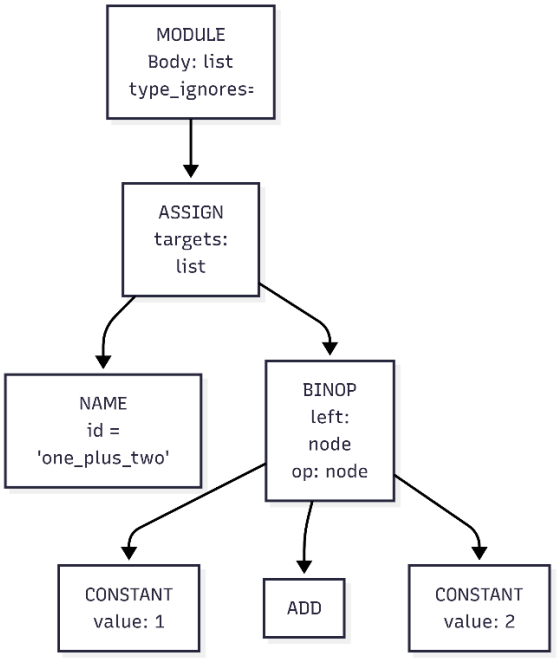
Kajian mutakhir menunjukkan bahwa alat deteksi plagiarisme kode di lingkungan pendidikan, seperti *Dolos*, terus berkembang guna meningkatkan pengalaman pengguna (Maertens et al., 2024). Secara teknis, integrasi representasi struktur (AST atau *graph*) dengan *Graph Neural Network* (GNN) terbukti potensial dalam meningkatkan ketahanan deteksi terhadap modifikasi kosmetik dan *rename identifier* (Guo et al., 2023). Oleh sebab itu, penelitian ini memfokuskan deteksi pada *Type-1* dan *Type-2*, sementara *Type-3* tingkat ringan digunakan sebagai uji ketahanan (*robustness test*) model.

2.2 Abstract Syntax Tree (AST)

Abstract Syntax Tree (AST) merupakan representasi struktur sintaksis dari suatu program dalam bentuk pohon yang menggambarkan konstruksi bahasa pemrograman secara hierarkis. AST menyajikan elemen-elemen program dalam bentuk node abstrak yang merepresentasikan pernyataan (*statement*), ekspresi (*expression*), dan struktur kontrol, tanpa menampilkan detail sintaks permukaan seperti spasi, komentar, atau tanda baca yang tidak memengaruhi logika program (Dong et al., 2024).

AST dibentuk melalui proses *parsing* kode sumber berdasarkan aturan tata bahasa (*grammar*) bahasa pemrograman yang digunakan. Setiap node pada AST merepresentasikan satu konstruksi sintaks, sedangkan hubungan *parent–child* antar node menunjukkan keterkaitan hierarkis antar elemen program. Dengan karakteristik tersebut, AST mampu mempertahankan struktur logika program meskipun terjadi perubahan kosmetik pada kode sumber, seperti perbedaan format penulisan atau penggantian nama variabel.

Pada bahasa pemrograman Python, AST merepresentasikan program dalam bentuk node-node seperti *Module*, *FunctionDef*, *Assign*, *For*, *While*, *BinOp*, dan *Return*. Struktur ini secara eksplisit mencerminkan organisasi dan struktur logika program. Dukungan bawaan Python terhadap pembentukan AST menjadikan bahasa ini banyak digunakan dalam penelitian analisis kode berbasis struktur, khususnya dalam tugas-tugas yang berkaitan dengan kemiripan kode dan *code analysis* (Büyük & Nizam, 2023).



Gambar 2. 2 Contoh representasi Abstract Syntax Tree (AST) dari potongan kode Python.

(Sumber: pybites, 2021, diolah Kembali).

Gambar 2.2 menunjukkan representasi Abstract Syntax Tree (AST) dari sebuah potongan kode Python. Setiap node pada AST merepresentasikan konstruksi sintaks tertentu, seperti *Assign*, *BinOp*, dan *Constant*, sedangkan hubungan *parent–child* menggambarkan struktur hierarkis program. Representasi ini

mengabstraksikan detail permukaan kode, seperti format penulisan dan penamaan variabel, sehingga menekankan struktur logika program secara keseluruhan.

Keunggulan utama AST dalam analisis kode program terletak pada kemampuannya menangkap struktur dan konteks sintaksis program. Pendekatan berbasis AST terbukti lebih tahan terhadap teknik plagiarisme sederhana, seperti *copy-paste* kosmetik (*Type-1*) dan *rename identifier* (*Type-2*), dibandingkan metode berbasis teks atau token semata (Zakeri-Nasrabadi et al., 2023). Oleh karena itu, AST menjadi representasi yang relevan untuk mendeteksi kemiripan kode yang berupaya menyamarkan kesamaan melalui perubahan permukaan.

Selain itu, AST dapat dipandang sebagai bentuk khusus dari *graph* terstruktur, di mana node merepresentasikan elemen sintaks dan *edge* merepresentasikan hubungan hierarkis antar *node*. Pandangan ini membuka peluang untuk memanfaatkan model pembelajaran berbasis *graph* dalam mempelajari pola struktur kode secara lebih mendalam. Sejumlah penelitian terkini menunjukkan bahwa kombinasi representasi AST dengan model pembelajaran mesin berbasis *graph* mampu meningkatkan kinerja berbagai tugas analisis kode, termasuk klasifikasi kode dan pengukuran kemiripan kode (Guo et al., 2023).

Berdasarkan karakteristik tersebut, AST dipilih sebagai representasi utama kode program dalam penelitian ini. Dengan menggunakan AST, sistem yang dikembangkan diharapkan mampu menangkap kemiripan struktur logika antar kode program meskipun terjadi perubahan kosmetik. Representasi AST selanjutnya akan dikonversi ke dalam bentuk graph untuk diproses menggunakan *Graph Neural Network* (GNN).

2.3 Representasi Graph pada Kode Program

Representasi *graph* merupakan pendekatan yang digunakan untuk memodelkan data terstruktur yang memiliki hubungan antar elemen. Secara konseptual, sebuah *graph* didefinisikan sebagai pasangan $G = (V , E)$, di mana V merepresentasikan himpunan *node* dan E merepresentasikan himpunan *edge* yang menggambarkan hubungan antar *node*. Representasi ini memungkinkan pemodelan struktur kompleks secara eksplisit, sehingga banyak digunakan dalam berbagai domain yang melibatkan data relasional, termasuk analisis kode program (Dong et al., 2024).

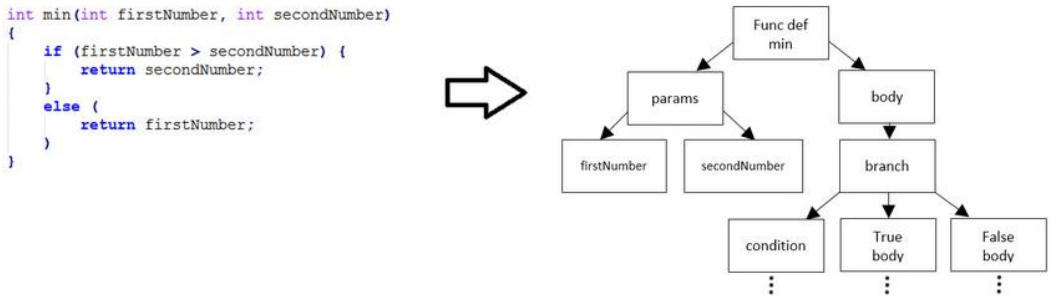
Dalam konteks analisis kode sumber, *Abstract Syntax Tree* (AST) dapat dipandang sebagai bentuk khusus dari *graph* terarah dan hierarkis. Setiap *node* pada AST merepresentasikan konstruksi sintaks tertentu dalam bahasa pemrograman, sedangkan *edge* merepresentasikan hubungan struktural antar elemen sintaks tersebut. Dengan demikian, AST tidak hanya merepresentasikan urutan token kode, tetapi juga menangkap struktur logika program secara eksplisit melalui relasi antar *node* (Zakeri-Nasrabadi et al., 2023).

Representasi kode program dalam bentuk *graph* dilakukan dengan memetakan struktur AST ke dalam komponen *graph*. Secara konseptual, pemetaan ini mencakup tiga elemen utama, yaitu:

1. *Node*, yang merepresentasikan elemen sintaks program seperti deklarasi fungsi, pernyataan penugasan, struktur perulangan, dan ekspresi operasi;
2. *Edge,* yang merepsentasikan hubungan struktural antar *node*, umumnya berupa hubungan *parent-child* sesuai hierarki AST;

3. *Fitur node*, yang menyimpan informasi tambahan mengenai karakteristik sintaks, seperti jenis *node* atau operator yang digunakan.

Melalui pemetaan ini, kode program dapat direpresentasikan sebagai *graph* yang mempertahankan struktur logika program secara menyeluruh.



Gambar 2. 3 Ilustrasi representasi graph dari Abstract Syntax Tree.

Sumber: Ďuračík et al., (2017)

Gambar 2.3 menunjukkan representasi *Abstract Syntax Tree* (AST) dalam bentuk *graph*, di mana *node* sebagai elemen sintaks dan *edge* sebagai hubungan strukturalnya. Melalui model ini, dua kode dengan logika serupa akan menghasilkan pola *graph* yang hampir identik meskipun terdapat perbedaan pada penamaan variabel atau format penulisan. Hal ini memungkinkan analisis kode yang lebih fokus pada struktur logika daripada tampilan tekstualnya.

Keunggulan utama penggunaan representasi *graph* dalam analisis kode program terletak pada kemampuannya menangkap pola struktural dan konteks hubungan antar elemen sintaks. Hal ini sangat relevan dalam deteksi plagiarisme kode, khususnya pada kasus *Type-1* (*copy-paste* kosmetik) dan *Type-2* (*rename identifier*), di mana perubahan hanya terjadi pada tingkat permukaan kode. Representasi *graph* memungkinkan sistem deteksi untuk lebih berfokus pada kesamaan struktur logika program dibandingkan kesamaan teks semata, yang sering

kali gagal mendeteksi bentuk plagiarisme terselubung (Zakeri-Nasrabadi et al., 2023).

Selain itu, representasi graph juga memungkinkan pemanfaatan model pembelajaran mesin yang dirancang khusus untuk data terstruktur, seperti *Graph Neural Network* (GNN). Model GNN memanfaatkan informasi *node* dan *edge* untuk mempelajari representasi vektor yang mencerminkan struktur keseluruhan *graph*. Berbagai penelitian terkini menunjukkan bahwa kombinasi representasi graph berbasis AST dan GNN mampu meningkatkan performa pada tugas analisis kode, termasuk pengukuran kemiripan dan *code clone detection* (G. Yang et al., 2023).

Dalam konteks penelitian di Indonesia, sebagian besar studi deteksi plagiarisme kode program masih didominasi oleh pendekatan berbasis teks atau token, seperti TF-IDF, *cosine similarity*, *Rabin-Karp*, dan *Winnowing*. Meskipun metode tersebut relatif sederhana, pendekatan ini cenderung sensitif terhadap perubahan kosmetik dan penggantian *identifier* (Eka Putra & Supriana, 2022). Oleh karena itu, penggunaan representasi *graph* berbasis AST pada penelitian ini diharapkan dapat mengatasi keterbatasan pendekatan berbasis teks yang umum digunakan dalam penelitian terdahulu di dalam negeri.

Berdasarkan uraian tersebut, representasi graph berbasis AST diposisikan sebagai tahapan konseptual penting yang menjembatani struktur kode program dan pemodelan menggunakan *Graph Neural Network* (GNN). Representasi ini menjadi dasar bagi proses pembelajaran berbasis graph yang akan dibahas secara mendalam pada subbab berikutnya.

2.4 Graph Berarah (Directed Graph)

Sebagai pengembangan dari definisi umum graph yang telah dipaparkan, representasi kode dalam penelitian ini diwujudkan dalam bentuk Graph Berarah (*Directed Graph*).

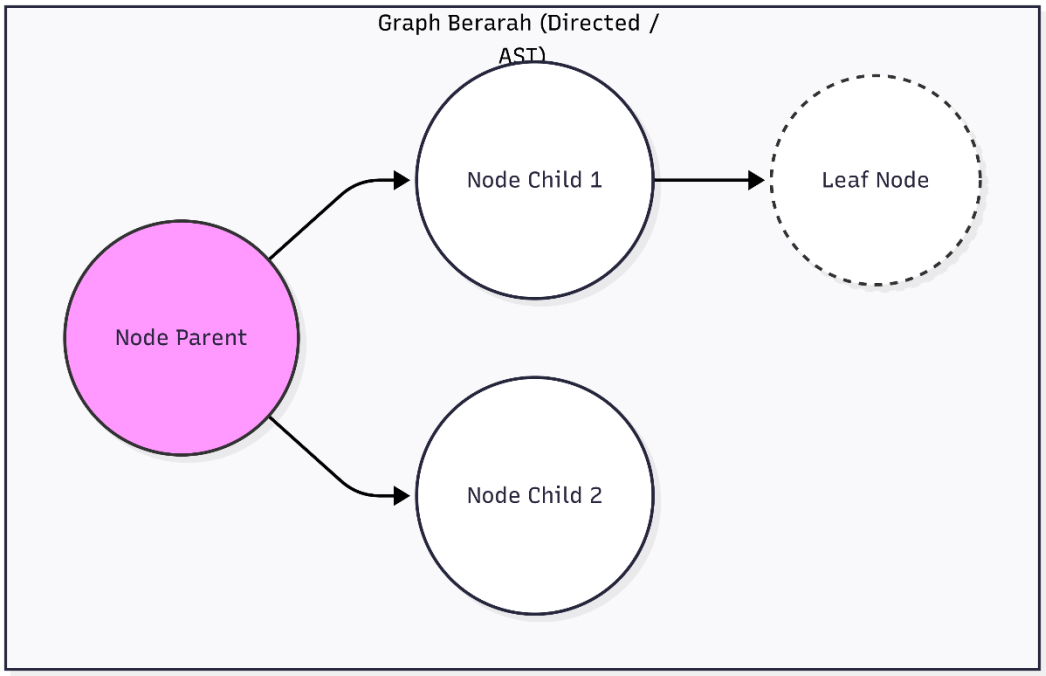
Graph berarah (*directed graph*) merupakan graph yang setiap sisinya memiliki orientasi tertentu. Secara formal, graph didefinisikan sebagai $G = (V, E)$, dengan $E \subseteq V \times V$, sehingga setiap edge dinyatakan sebagai pasangan terurut (u, v). Pasangan ini menunjukkan adanya relasi dari node u menuju node v, dan tidak secara otomatis berlaku sebaliknya (Zhou et al., 2020).

Karakteristik utama graph berarah terletak pada orientasi relasinya. Setiap node memiliki in-degree (jumlah edge masuk) dan out-degree (jumlah edge keluar), yang merepresentasikan arah dependensi struktural. Sifat ini membedakan graph berarah dari graph tak berarah yang relasinya bersifat simetris. Dalam sistem yang memiliki struktur hierarkis atau ketergantungan eksplisit, penggunaan graph berarah menjadi lebih representatif (Wu et al., 2021).

Dalam penelitian ini, struktur *Abstract Syntax Tree* (AST) termasuk dalam kategori *Directed Acyclic Graph* (DAG), yaitu graph berarah yang tidak mengandung siklus. Sifat *acyclic* muncul karena struktur AST mengikuti aturan tata bahasa-bahasa pemrograman, sehingga tidak memungkinkan terbentuknya relasi melingkar (Dong et al., 2024). Dengan demikian, arah edge pada AST merepresentasikan hubungan induk–turunan (*parent–child*) secara eksplisit dan deterministik.

Ilustrasi konseptual graph berarah dalam konteks AST dapat dilihat pada Gambar 2.4. Pada gambar tersebut, node Parent memiliki relasi terarah menuju

Node Child 1 dan *Node Child* 2, yang menunjukkan hubungan hierarkis. Selanjutnya, Node Child 1 memiliki relasi menuju *Leaf Node*, yang merepresentasikan node terminal dalam struktur pohon.



Gambar 2. 4 Ilustrasi struktur graph berarah pada representasi Abstract Syntax Tree (AST).

Sumber: Dokumen Pribadi.

Pada Gambar 2.4 terlihat bahwa arah edge menunjukkan alur dependensi dari node induk menuju node turunan. Orientasi ini menentukan struktur hierarkis dan konteks sintaks antar elemen. Jika arah relasi dihilangkan, maka hubungan induk–anak tidak dapat dibedakan secara eksplisit, sehingga struktur logika menjadi ambigu.

Sifat graph berarah ini memiliki implikasi langsung terhadap mekanisme *message passing* dalam Graph Neural Network (GNN). Pada proses tersebut, propagasi informasi antar node mengikuti struktur edge yang ada pada graph.

Orientasi edge memengaruhi bagaimana informasi dikumpulkan dan diperbarui pada setiap node, sehingga memungkinkan model menangkap pola struktur secara kontekstual (Wu et al., 2021). Oleh karena itu, pemahaman mengenai graph berarah menjadi landasan teoritis penting dalam integrasi AST dan GNN pada sistem deteksi plagiarisme kode program yang dikembangkan dalam penelitian ini.

2.5 Graph Neural Network (GNN)

Graph Neural Network (GNN) merupakan kelas model pembelajaran mesin yang dirancang khusus untuk memproses data berbentuk *graph*. Berbeda dengan model pembelajaran konvensional yang bekerja pada data berstruktur grid seperti vektor atau matriks, GNN mampu mempelajari representasi data yang memiliki hubungan kompleks antar entitas melalui struktur *node* dan *edge*. Kemampuan ini menjadikan GNN sangat sesuai untuk berbagai permasalahan yang melibatkan data relasional, termasuk analisis kode program berbasis *graph* (Wu et al., 2021).

Secara umum, tujuan utama GNN adalah mempelajari representasi vektor (*embedding*) untuk *node*, *edge*, atau keseluruhan graph dengan mempertimbangkan struktur dan hubungan antar elemen. Proses pembelajaran pada GNN dilakukan melalui mekanisme yang dikenal sebagai message passing, di mana setiap node secara iteratif menerima, mengagregasi, dan memperbarui informasi dari node tetangganya. Melalui proses ini, informasi lokal dan global pada graph dapat dipelajari secara bertahap (Zhou et al., 2020).

2.5.1 Mekanisme Message Passing dan Node Embedding

Pada setiap lapisan GNN, representasi *node* diperbarui berdasarkan informasi dari tetangganya. Secara konseptual, mekanisme ini terdiri dari tiga tahapan utama, yaitu:

1. *Message construction*, yaitu pembentukan pesan dari node tetangga;
2. *Aggregation*, yaitu penggabungan pesan-pesan dari seluruh tetangga node;
3. *Update*, yaitu pembaruan representasi node berdasarkan hasil agregasi.

Melalui tahapan ini, GNN mampu menangkap pola struktural dan hubungan kontekstual antar *node* dalam *graph*. Setelah beberapa lapisan, setiap *node* memiliki *embedding* yang mengandung informasi tidak hanya dari dirinya sendiri, tetapi juga dari lingkungan sekitarnya. Pendekatan ini memungkinkan GNN mempelajari representasi graph yang kaya dan informatif (Wu et al., 2021).

2.5.2 Arsitektur GNN yang Umum Digunakan

Berbagai arsitektur GNN telah dikembangkan, di antaranya *Graph Convolutional Network* (GCN), *Graph Attention Network* (GAT), dan *GraphSAGE*. Meskipun memiliki perbedaan pada cara agregasi dan pembobotan informasi, ketiga arsitektur tersebut memiliki prinsip dasar yang sama, yaitu memanfaatkan struktur *graph* untuk memperbarui representasi node secara interatif (Zhou et al., 2020).

Dalam konteks analisis kode program, arsitektur GNN digunakan untuk mempelajari pola struktur *graph* yang berasal dari representasi AST. Setiap *node* pada *graph* AST dipetakan ke *embedding* vektor, sedangkan struktur graph memungkinkan GNN untuk memahami hubungan sintaks antar elemen kode. Pendekatan ini terbukti efektif dalam berbagai tugas *machine learning for source code*, termasuk klasifikasi kode dan pengukuran kemiripan kode (Dong et al., 2024).

2.5.3 GNN untuk Analisis dan Kemiripan Kode Program

Penggunaan GNN dalam analisis kode program didorong oleh kemampuannya dalam memodelkan struktur kode secara eksplisit. Berbeda dengan pendekatan berbasis teks atau token, GNN tidak hanya memperhatikan urutan elemen kode, tetapi juga hubungan struktural antar elemen tersebut. Hal ini menjadikan GNN lebih tahan terhadap perubahan kosmetik dan penggantian identifier pada kode program (Zakeri-Nasrabadi et al., 2023).

Dalam penelitian deteksi kemiripan kode, graph AST digunakan sebagai input GNN untuk menghasilkan *embedding graph* yang merepresentasikan struktur keseluruhan kode program. *Embedding* tersebut kemudian dapat digunakan untuk mengukur tingkat kemiripan antara dua kode program. Pendekatan ini relevan untuk skema *pairwise detection* (kode A vs kode B), di mana dua *graph* dibandingkan untuk menentukan tingkat kemiripan dan label plagiarisme.

2.5.4 Relevansi GNN dalam Penelitian Ini

Berdasarkan karakteristik tersebut, *Graph Neural Network* dipilih sebagai model pembelajaran dalam penelitian ini karena kemampuannya dalam memanfaatkan struktur *graph* berbasis AST. GNN memungkinkan sistem untuk mempelajari representasi kode program yang menekankan kesamaan struktur logika, bukan sekadar kesamaan teks. Dengan demikian, pendekatan ini diharapkan mampu mengatasi keterbatasan metode konvensional berbasis teks yang masih umum digunakan dalam praktik deteksi plagiarisme kode.

Penggunaan GNN juga sejalan dengan tujuan penelitian ini, yaitu menghasilkan sistem deteksi plagiarisme kode program yang mampu memberikan skor kemiripan dan label keputusan secara objektif berdasarkan struktur kode. Oleh

karena itu, GNN diposisikan sebagai komponen inti dalam pemodelan sistem deteksi plagiarisme kode program berbasis AST.

2.6 Karakteristik Pembobotan dalam Graph dan Graph Neural Network

Dalam teori graph, graf berbobot (*weighted graph*) adalah graf yang setiap sisinya memiliki nilai numerik tertentu, sehingga secara formal dinyatakan sebagai $G = (V, E, w)$, dengan fungsi bobot w yang memetakan setiap edge ke suatu nilai. Sebaliknya, graf tak berbobot (*unweighted graph*) hanya merepresentasikan keberadaan relasi antar node tanpa nilai numerik tambahan, sehingga dinyatakan sebagai $G = (V, E)$ (Diestel, 2017).

Dalam penelitian ini, Abstract Syntax Tree (AST) direpresentasikan sebagai *directed unweighted graph*, karena edge hanya menyatakan hubungan struktural *parent–child* tanpa bobot numerik. Relasi antar *node* diperlakukan setara secara struktural dan tidak merepresentasikan tingkat kekuatan atau intensitas tertentu.

Meskipun demikian, proses pembelajaran menggunakan *Graph Neural Network* (GNN) melibatkan parameter bobot (*learnable weights*) pada tingkat model. Dalam mekanisme *message passing*, pembaruan representasi node dilakukan melalui transformasi berbobot, misalnya:

Rumus 2.1 Persamaan Pembaruan Representasi Node pada GNN.

$$h_v^{(k+1)} = \sigma\left(W^{(k)} \cdot AGGREGATE\left(\left\{h_u^{(k)}\right\}\right)\right)$$

Dimana $W^{(k)}$ merupakan matriks bobot yang dipelajari selama proses pelatihan, dan σadalah fungsi aktivasi nonlinier (Wu et al., 2021). Matriks bobot tersebut memungkinkan model menyesuaikan representasi embedding berdasarkan pola struktur graph yang relevan terhadap tugas deteksi plagiarisme.

Dengan demikian, meskipun graph yang digunakan bersifat tak berbobot pada tingkat struktur, sistem deteksi plagiarisme yang dikembangkan merupakan model parametrik berbobot pada tingkat pembelajaran. Pemisahan konseptual ini penting untuk membedakan antara bobot pada struktur graph dan bobot pada parameter GNN dalam proses klasifikasi.

2.7 Deteksi Kemiripan Kode Program (Code Similarity & Clone Detection)

Deteksi kemiripan kode program (*code similarity detection*) merupakan proses untuk mengukur tingkat kesamaan antara dua atau lebih potongan kode sumber berdasarkan karakteristik tertentu. Dalam literatur rekayasa perangkat lunak, topik ini sering dibahas beririsan dengan *code clone detection* dan plagiarism detection, meskipun ketiganya memiliki fokus yang berbeda. *Code similarity* menekankan pada pengukuran kesamaan, *code clone detection* berfokus pada identifikasi *fragmen* kode yang serupa, sedangkan *plagiarism detection* menempatkan kesamaan tersebut dalam konteks pelanggaran etika dan integritas akademik (Roy et al., 2009).

Secara umum, kemiripan kode dapat dianalisis pada beberapa tingkat, yaitu teks, token, struktur, dan semantik. Analisis pada tingkat teks dan token mengukur kesamaan berdasarkan karakter atau urutan token, sedangkan analisis struktural memanfaatkan representasi sintaks seperti *Abstract Syntax Tree* (AST). Analisis semantik berupaya menilai kesamaan berdasarkan perilaku atau makna program. Perbedaan tingkat analisis ini memengaruhi ketahanan metode terhadap teknik penyamaran plagiarisme (Zakeri-Nasrabadi et al., 2023).

2.7.1 Code Clone Detection dan Klasifikasinya

Dalam kajian code clone detection, kemiripan kode diklasifikasikan berdasarkan tingkat perubahan yang dilakukan terhadap kode sumber. Klasifikasi yang paling umum membagi code clone ke dalam Type-1, Type-2, dan Type-3 (Koschke, 2007).

1. *Type-1* (*Copy-Paste* Kosmetik): penyalinan kode dengan perubahan terbatas pada spasi, indentasi, atau komentar, tanpa mengubah struktur logika program.
2. *Type-2* (*Rename Identifier*): penyalinan kode dengan perubahan pada nama variabel, fungsi, atau parameter, sementara struktur dan alur logika tetap sama.
3. *Type-3* (*Refactoring* Ringan): penyalinan kode dengan perubahan struktur lokal atau *refactoring* sederhana yang masih mempertahankan fungsi program.

Klasifikasi ini penting karena tingkat kesulitan deteksi meningkat seiring naiknya tipe *clone*. Metode berbasis teks umumnya efektif untuk *Type-1*, namun kinerjanya menurun pada *Type-2* dan *Type-3*, sehingga diperlukan pendekatan yang mempertimbangkan struktur kode (Roy et al., 2009).

2.7.2 Pendekatan Deteksi Kemiripan Kode Program

Berbagai pendekatan telah dikembangkan untuk mendeteksi kemiripan kode. Secara konseptual, pendekatan tersebut dapat dikelompokkan ke dalam pendekatan berbasis teks/token dan pendekatan berbasis struktur (Zakeri-Nasrabadi et al., 2023).

1. Pendekatan Berbasis Teks atau Token

Pendekatan berbasis teks atau token menganalisis kode sebagai urutan karakter atau token tanpa mempertimbangkan struktur sintaks. Metode ini relatif sederhana dan efisien secara komputasi, sehingga banyak digunakan dalam praktik awal deteksi plagiarisme. Namun, penelitian menunjukkan bahwa pendekatan ini sangat sensitif terhadap perubahan kosmetik, seperti penggantian nama variabel atau perubahan format, sehingga kurang efektif untuk mendeteksi plagiarisme terselubung (Zakeri-Nasrabadi et al., 2023).

2. Pendekatan Berbasis Struktur

Pendekatan berbasis struktur menganalisis kemiripan kode berdasarkan organisasi dan hubungan antar elemen sintaks program. Representasi seperti AST memungkinkan kode dipandang sebagai struktur hierarkis, sehingga perubahan pada tingkat permukaan tidak secara signifikan memengaruhi representasi. Pendekatan ini terbukti lebih stabil untuk mendeteksi rename identifier dan perubahan format penulisan (Koschke, 2007).

Pendekatan struktural dapat diperluas dengan merepresentasikan AST dalam bentuk *graph*, sehingga hubungan antar elemen sintaks dapat dianalisis secara lebih fleksibel. Representasi *graph* ini membuka peluang penggunaan model pembelajaran berbasis graph untuk mempelajari pola kemiripan struktur kode secara lebih mendalam (Dong et al., 2024).

2.7.3 Skema Deteksi Kemiripan Kode

Deteksi kemiripan kode program dapat dilakukan melalui beberapa skema, salah satunya adalah *pairwise detection*, yaitu membandingkan dua potongan kode secara langsung (kode A dan kode B). Skema ini menghasilkan skor kemiripan yang

merepresentasikan tingkat kesamaan antara kedua kode, yang selanjutnya dapat digunakan untuk pengambilan keputusan. Skema *pairwise* banyak digunakan dalam konteks deteksi plagiarisme akademik karena sesuai dengan kebutuhan pemeriksaan kemiripan antar dua karya secara langsung dan mudah diinterpretasikan (Roy et al., 2009).

2.7.4 Posisi Pendekatan AST dan GNN dalam Deteksi Kemiripan Kode

Berdasarkan kajian literatur, pendekatan berbasis AST dan *graph* dipandang lebih sesuai untuk mendeteksi kemiripan kode yang telah mengalami penyamaran struktur. Dengan memanfaatkan representasi *graph* dan *Graph Neural Network* (GNN), sistem deteksi dapat mempelajari pola kemiripan struktural yang tidak dapat ditangkap oleh pendekatan berbasis teks semata. Sejumlah penelitian internasional menunjukkan bahwa pembelajaran berbasis *graph* efektif untuk berbagai tugas analisis kode, termasuk pengukuran kemiripan dan code clone detection (Wu et al., 2021). Oleh karena itu, pendekatan AST-GNN menjadi landasan konseptual yang kuat bagi sistem deteksi kemiripan kode program dalam penelitian ini.

2.8 Keterkaitan Abstract Syntax Tree (AST) dan Graph Neural Network (GNN) dalam Deteksi Plagiarisme Kode Program

Deteksi plagiarisme kode program yang efektif memerlukan pendekatan yang mampu menangkap kesamaan struktur dan logika program, bukan hanya kesamaan teks. Pada subbab sebelumnya telah dijelaskan bahwa *Abstract Syntax Tree* (AST) merupakan representasi struktural kode program yang merefleksikan konstruksi sintaks serta hubungan hierarkis antar elemen kode, sedangkan *Graph Neural Network* (GNN) merupakan model pembelajaran mesin yang dirancang

untuk memproses data berbentuk graph. Keterkaitan antara AST dan GNN terletak pada kesesuaian bentuk representasi data dan kemampuan model dalam mempelajari pola struktural secara relasional (Wu et al., 2021).

AST menyediakan representasi kode program dalam bentuk struktur pohon yang mengabstraksikan detail permukaan kode, seperti format penulisan dan penamaan variabel. Dengan karakteristik ini, AST mampu mempertahankan struktur logika program meskipun terjadi perubahan kosmetik atau *rename identifie*r. Representasi ini banyak digunakan dalam analisis kode karena relatif stabil terhadap variasi sintaks yang tidak memengaruhi semantik program (Koschke, 2007). Namun demikian, AST sebagai struktur statis belum secara langsung menyediakan mekanisme pembelajaran untuk mengukur kemiripan antar dua struktur kode secara otomatis.

Untuk memanfaatkan struktur AST dalam pembelajaran mesin, AST umumnya direpresentasikan kembali dalam bentuk *graph*, di mana setiap *node* AST dipetakan sebagai *node graph* dan hubungan sintaks dipetakan sebagai *edge*. Representasi graph ini memungkinkan pemodelan hubungan antar elemen sintaks secara lebih fleksibel dan kaya konteks dibandingkan representasi linier atau berbasis token (Zakeri-Nasrabadi et al., 2023). Pendekatan berbasis *graph* juga membuka peluang untuk menggunakan model pembelajaran yang secara eksplisit memanfaatkan struktur relasional data.

Graph Neural Network berperan sebagai model yang mempelajari representasi (*embedding*) dari graph AST tersebut. Melalui mekanisme *message passing*, setiap *node* pada *graph* AST dapat menggabungkan informasi dari *node* tetangganya, sehingga representasi *node* dan *graph* secara keseluruhan

mencerminkan konteks struktur kode program. Proses ini memungkinkan GNN untuk menangkap pola kesamaan struktural antar graph AST yang berasal dari kode program yang berbeda (Wu et al., 2021).

Dalam konteks deteksi plagiarisme kode program, keterkaitan AST dan GNN memungkinkan sistem untuk berfokus pada kesamaan struktur logika, bukan sekadar kesamaan teks. Dua kode program yang memiliki struktur logika serupa, meskipun berbeda secara tekstual, akan menghasilkan *graph* AST dengan pola relasional yang mirip. GNN kemudian mempelajari pola-pola tersebut dan menghasilkan representasi *graph* yang dapat dibandingkan untuk mengukur tingkat kemiripan kode (Dong et al., 2024).

Pendekatan integratif antara AST dan GNN ini sangat relevan untuk mendeteksi plagiarisme kode *Type-1* (*copy-paste* kosmetik) dan *Type-2* (*rename identifier*), yang umumnya sulit dideteksi oleh metode berbasis teks atau token. Selain itu, penelitian terkini menunjukkan bahwa pendekatan berbasis *graph* juga memiliki potensi untuk menangani perubahan struktural ringan pada kode, sehingga dapat digunakan sebagai uji ketahanan (*robustness test*) terhadap plagiarisme *Type-3* ringan (Zakeri-Nasrabadi et al., 2023).

Dengan demikian, keterkaitan antara AST dan GNN membentuk dasar konseptual sistem deteksi plagiarisme kode program dalam penelitian ini. AST berperan sebagai representasi struktural kode program, sedangkan GNN berperan sebagai model pembelajaran yang mengekstraksi dan membandingkan pola kemiripan struktur tersebut. Integrasi kedua komponen ini menghasilkan pendekatan deteksi plagiarisme kode yang lebih robust dan adaptif dibandingkan

metode konvensional berbasis teks, serta menjadi landasan teoritis bagi perancangan sistem pada bab metodologi penelitian.

2.9 Penelitian Terkait

Berbagai studi telah membahas deteksi plagiarisme kode program dan kemiripan kode (*code similarity* dan *code clone detection*) dengan pendekatan yang beragam, mulai dari metode berbasis teks dan token, analisis struktur kode menggunakan *Abstract Syntax Tree* (AST), hingga pendekatan berbasis graph dan pembelajaran mesin modern seperti *Graph Neural Network* (GNN) serta model *embedding* kode. Ringkasan penelitian terkait yang relevan dengan penelitian ini disajikan dalam bentuk tabel untuk memberikan gambaran mengenai fokus pembahasan, metode yang digunakan, serta kelebihan dan kekurangan dari masing-masing penelitian.

Tabel 2. 1 Ringkasan Penelitian Terdahulu.

No	Judul dan Penelitian	Pembahasan	Metode	Kelebihan dan Kekurangan
1	Cross-language Source Code Clone Detection Based on Graph Neural Network (Y. Zhang et al., 2024)	Penelitian ini membahas deteksi *code clone* lintas bahasa pemrograman dengan memanfaatkan representasi struktur kode dalam bentuk graph untuk mengatasi keterbatasan metode berbasis teks.	AST, Graph Neural Network	Kelebihan: Mampu mendeteksi clone lintas bahasa Kekurangan: Tidak spesifik pada plagiarisme akademik
2	A Novel Source Code Clone Detection Method Based on Dual-GCN and IVHFS (H. Yang et al., 2023)	Penelitian ini mengusulkan metode deteksi code clone menggunakan dua Graph Convolutional Network untuk menangkap hubungan struktural antar elemen kode.	Graph, Dual-GCN	Kelebihan: Representasi struktur kaya Kekurangan: Arsitektur kompleks

3	Code Similarity Prediction Model for Industrial Management Features Based on Graph Neural Networks (Z. Li et al., 2024)	Penelitian ini mengembangkan model prediksi kemiripan kode berbasis GNN untuk mengukur kesamaan logika program.	Graph, GNN	Kelebihan: Akurat untuk similarity struktural Kekurangan: Studi kasus industri
4	Exploring the Boundaries Between LLM Code Clone Detection and Code Similarity Assessment on Human and AI-Generated Code (Z. Zhang & Saber, 2025)	Penelitian ini mengevaluasi kemampuan LLM dalam mendeteksi code clone dan menilai kemiripan kode berbasis embedding semantik.	LLM, Code Embedding	Kelebihan: Pendekatan mutakhir Kekurangan: Tidak menggunakan AST/graph
5	Real-Time Code Plagiarism Detection Using NLP and Machine Learning for Academic and Industry Applications (Siddiqui, 2025)	Penelitian ini membahas sistem deteksi plagiarisme kode secara real-time dengan mengombinasikan AST, NLP, dan GNN.	AST, NLP, GNN	Kelebihan: Fokus langsung pada plagiarisme Kekurangan: Evaluasi terbatas
6	Source Code Plagiarism Detection with Pre-Trained Model Embeddings and Automated Machine Learning (Ebrahim & Joy, 2023)	Penelitian ini mengkaji penggunaan embedding model pra-latih dan AutoML untuk klasifikasi plagiarisme kode.	Code Embedding, AutoML	Kelebihan: Tidak bergantung struktur eksplisit Kekurangan: Kurang robust struktural
7	Code Classification with Graph Neural	Penelitian ini mengevaluasi penggunaan GNN dalam analisis kode	Graph, GNN	Kelebihan: Evaluasi metodologis kuat

	Networks: Have You ever struggled to make it work? (Q. Yu et al., 2023)	dan menyoroti pentingnya baseline evaluasi.		Kekurangan: Bukan fokus similarity
8	Graph-Based Code Semantics Learning for Efficient Semantic Code Clone Detection (D. Yu et al., 2023)	Penelitian ini mengusulkan pembelajaran semantik kode berbasis graph untuk mendeteksi semantic code clone.	AST/PDG, Graph Learning	Kelebihan: Mendeteksi clone semantic Kekurangan: Kompleksitas tinggi
9	Semantic Similarity Search for Source Code Plagiarism Detection: An Exploratory Study (Ebrahim & Joy, 2024)	Penelitian ini membahas pencarian kemiripan semantik berbasis embedding untuk mendeteksi plagiarisme kode.	Code Embedding, Similarity Search	Kelebihan: Efektif untuk dataset besar Kekurangan: Tidak memanfaatkan AST
10	CODE CLONE DETECTION WITH SELF-SUPERVISION ON DUAL GRAPHS (C. Li et al., 2024)	Penelitian ini mengusulkan kerangka self-supervised berbasis dual-graph untuk deteksi code clone.	Dual-Graph Learning	Kelebihan: Minim kebutuhan label Kekurangan: Belum diuji khusus plagiarisme akademik

2.10 Analisis Gap

Berdasarkan ringkasan penelitian terkait pada Tabel 2.1, dapat diketahui bahwa deteksi plagiarisme dan kemiripan kode program telah banyak diteliti dengan pendekatan yang beragam, mulai dari metode berbasis teks dan embedding semantik hingga pendekatan berbasis struktur menggunakan *Abstract Syntax Tree* (AST) dan representasi *graph*. Meskipun demikian, hasil telaah terhadap penelitian-

penelitian tersebut menunjukkan masih adanya beberapa celah penelitian yang relevan untuk dikaji lebih lanjut.

Pertama, sebagian penelitian masih mengandalkan representasi kode berbasis *embedding* atau teks, tanpa memanfaatkan struktur sintaks kode secara eksplisit. Pendekatan ini relatif efektif untuk mendeteksi kemiripan permukaan, namun kurang robust terhadap plagiarisme dengan penyamaran struktur, seperti *rename identifier* dan perubahan format penulisan kode (Ebrahim & Joy, 2023).

Kedua, penelitian yang memanfaatkan *Abstract Syntax Tree* (AST) umumnya mampu merepresentasikan struktur kode dengan baik, namun AST sering digunakan sebagai fitur statis atau metrik kesamaan, dan belum banyak penelitian yang mengintegrasikannya secara mendalam dengan pembelajaran berbasis graph untuk mempelajari hubungan relasional antar elemen kode (Chen et al., 2024).

Ketiga, pendekatan berbasis *graph* dan *Graph Neural Network* (GNN) telah menunjukkan potensi yang menjanjikan dalam analisis kode dan *code clone detection*. Namun, sebagian besar penelitian tersebut berfokus pada tugas klasifikasi atau deteksi *clone* secara umum, bukan secara spesifik pada deteksi plagiarisme kode program dalam konteks akademik. Selain itu, skema deteksi yang digunakan sering kali berupa klasifikasi global atau clustering, sehingga kurang sesuai dengan kebutuhan pemeriksaan kemiripan antar dua karya secara langsung (Q. Yu et al., 2023).

Keempat, penggunaan skema deteksi pairwise (kode A vs kode B) dengan keluaran berupa skor kemiripan yang terukur dan label keputusan masih relatif terbatas, khususnya pada penelitian yang mengombinasikan AST sebagai representasi struktural dan GNN sebagai model pembelajaran utama. Padahal,

skema pairwise lebih selaras dengan kebutuhan praktis pemeriksaan plagiarisme pada lingkungan pendidikan.

Berdasarkan celah-celah tersebut, penelitian ini diarahkan untuk mengintegrasikan *Abstract Syntax Tree* (AST) dan *Graph Neural Network* (GNN) dalam skema deteksi kemiripan kode program secara *pairwise*, dengan fokus utama pada plagiarisme *Type-1* (*copy-paste* kosmetik) dan *Type-2* (*rename identifier*), serta *Type-3* ringan sebagai uji ketahanan (*robustness test*). Penelitian ini juga menghasilkan skor kemiripan dan label keputusan, sehingga diharapkan mampu memberikan pendekatan yang lebih terstruktur, *robust*, dan relevan untuk deteksi plagiarisme kode program di lingkungan akademik.

BAB III

METODOLOGI PENELITIAN

3.1 Jenis dan Pendekatan Penelitian

Penelitian ini adalah mengembangkan dan menguji sebuah sistem yang dapat digunakan untuk mendeteksi plagiarisme kode program secara praktis, khususnya dalam konteks akademik. Sistem yang dikembangkan tidak hanya bersifat konseptual, tetapi dirancang untuk dapat diimplementasikan dan dievaluasi kinerjanya berdasarkan data nyata.

Selain itu, penelitian ini juga bersifat eksperimental, karena melibatkan proses perancangan, pelatihan, dan pengujian model untuk mengevaluasi efektivitas pendekatan yang diusulkan. Melalui eksperimen ini, performa sistem deteksi plagiarisme kode program dianalisis menggunakan metrik evaluasi tertentu untuk mengetahui sejauh mana model mampu mendeteksi kemiripan kode secara akurat.

Dari sisi pendekatan, penelitian ini menggunakan pendekatan *supervised learning*. Pendekatan ini dipilih karena data yang digunakan dalam penelitian telah dilengkapi dengan label kelas, yaitu plagiarisme dan tidak plagiarisme, sehingga model dapat dilatih untuk mempelajari pola kemiripan kode berdasarkan contoh pasangan kode yang telah diberi label. Pendekatan *supervised learning* memungkinkan model untuk secara eksplisit mempelajari hubungan antara representasi struktur kode dan keputusan klasifikasi yang dihasilkan.

Penelitian ini juga menggunakan pendekatan kuantitatif, karena evaluasi kinerja sistem dilakukan berdasarkan pengukuran numerik. Kinerja model dinilai menggunakan metrik evaluasi seperti akurasi, presisi, *recall*, dan *F1-score*, serta

skor kemiripan yang dihasilkan oleh sistem. Pendekatan kuantitatif memungkinkan hasil penelitian dianalisis secara objektif dan terukur.

Pemilihan jenis dan pendekatan penelitian tersebut selaras dengan tujuan penelitian, yaitu mengembangkan sistem deteksi plagiarisme kode program berbasis *Abstract Syntax Tree* (AST) dan *Graph Neural Network* (GNN) serta mengevaluasi kinerjanya secara empiris. Dengan mengombinasikan penelitian terapan, eksperimen, supervised learning, dan pendekatan kuantitatif, penelitian ini diharapkan mampu menghasilkan solusi yang tidak hanya valid secara teoretis, tetapi juga efektif dan relevan untuk diterapkan dalam lingkungan akademik.

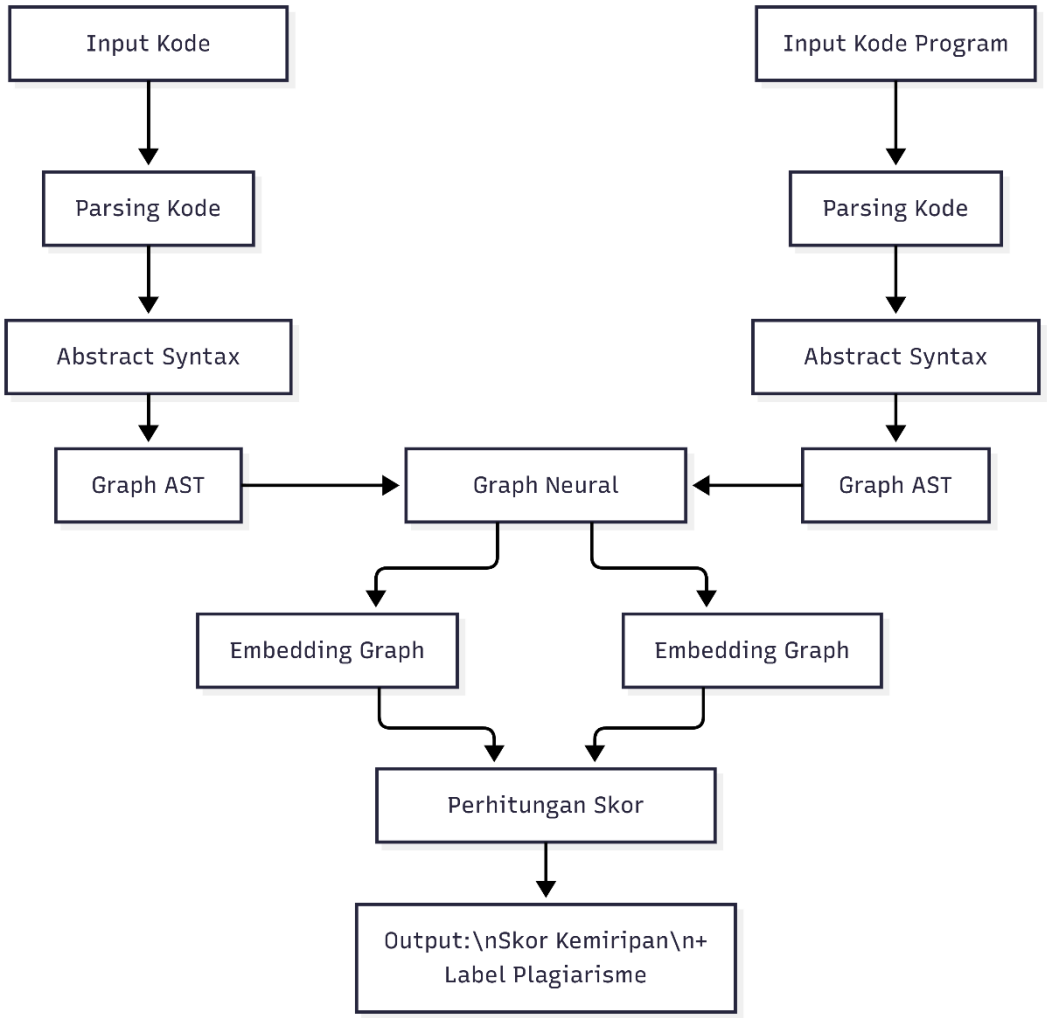
3.2 Gambaran Umum Sistem

Sistem yang dikembangkan dalam penelitian ini merupakan sistem deteksi plagiarisme kode program yang bekerja dengan membandingkan dua potongan kode sumber secara langsung (*pairwise*) untuk menentukan tingkat kemiripan dan status plagiarisme. Sistem dirancang untuk mendeteksi kemiripan kode berdasarkan struktur sintaks program, bukan hanya kesamaan teks, sehingga lebih robust terhadap teknik plagiarisme seperti perubahan format penulisan dan penggantian nama variabel.

Secara umum, sistem menerima dua kode program sebagai masukan, yaitu kode A dan kode B. Kedua kode tersebut diproses melalui tahapan yang sama untuk menghasilkan representasi struktural yang dapat dibandingkan. Proses deteksi dilakukan dengan memanfaatkan *Abstract Syntax Tree* (AST) sebagai representasi struktur kode dan *Graph Neural Network* (GNN) sebagai model pembelajaran untuk mempelajari dan membandingkan pola struktur tersebut.

3.2.1 Alur Kerja Sistem

Alur kerja sistem merupakan rangkaian tahapan sistematis yang menggambarkan proses transformasi kode program menjadi hasil deteksi plagiarisme. Alur ini berfungsi sebagai kerangka kerja untuk menjamin objektivitas proses, mulai dari ekstraksi struktur Abstract Syntax Tree (AST) hingga analisis menggunakan Graph Neural Network (GNN).



Gambar 3. 1 Diagram arsitektur sistem deteksi plagiarisme kode program berbasis Abstract Syntax Tree (AST) dan Graph Neural Network (GNN).

Alur kerja sistem deteksi plagiarisme kode program dalam penelitian ini dapat dilihat pada Gambar 3.1 dengan penjelasan sebagai berikut.

1. Input Kode Program (A vs B)

Sistem menerima dua buah kode program, yaitu kode A dan kode B, yang akan diperiksa tingkat kemiripannya. Kedua kode ini dapat berupa kode asli dan kode yang diduga hasil plagiarisme.

2. Parsing Kode Program ke Abstract Syntax Tree (AST)

Setiap kode program diparsing menggunakan parser bahasa pemrograman yang sesuai (*Python*) untuk menghasilkan *Abstract Syntax Tree* (AST). AST merepresentasikan struktur sintaks kode program dalam bentuk hierarki *node*, sehingga detail permukaan seperti komentar dan format penulisan tidak memengaruhi representasi.

3. Transformasi AST menjadi Graph

AST yang dihasilkan kemudian direpresentasikan dalam bentuk *graph*, di mana setiap node pada AST menjadi node graph dan hubungan sintaks antar elemen kode direpresentasikan sebagai edge. Representasi graph ini memungkinkan pemodelan hubungan struktural antar komponen kode secara lebih fleksibel.

4. Pemrosesan Graph Menggunakan Graph Neural Network (GNN)

Graph hasil transformasi AST selanjutnya diproses menggunakan *Graph Neural Network* (GNN) untuk menghasilkan *embedding graph*, yaitu representasi vektor yang mencerminkan struktur dan karakteristik kode program. Proses ini dilakukan untuk kode A dan kode B secara terpisah.

5. Perhitungan Skor Kemiripan dan Penentuan Label

Embedding dari kedua *graph* dibandingkan untuk menghasilkan skor kemiripan yang merepresentasikan tingkat kesamaan antara kode A dan kode B.

Berdasarkan skor tersebut, sistem memberikan label keputusan, yaitu plagiarisme atau tidak plagiarisme, sesuai dengan ambang batas (*threshold*) yang ditentukan.

3.2.2 Peran Abstract Syntax Tree dan Graph Neural Network

Dalam sistem ini, *Abstract Syntax Tree* (AST) berperan sebagai representasi struktural kode program yang mampu mempertahankan logika dan susunan sintaks program meskipun terjadi perubahan pada tingkat permukaan. AST memungkinkan sistem untuk fokus pada struktur inti kode, sehingga efektif dalam mendeteksi plagiarisme *Type-1* (*copy-paste* kosmetik) dan *Type-2* (*rename identifier*).

Sementara itu, *Graph Neural Network* (GNN) berperan sebagai model pembelajaran mesin yang mempelajari pola kemiripan antar *graph* AST. Melalui mekanisme *message passing*, GNN mampu menangkap hubungan relasional antar *node* dalam *graph* dan menghasilkan representasi yang kaya konteks. Integrasi AST dan GNN memungkinkan sistem untuk membandingkan dua kode program berdasarkan kesamaan struktur logika, bukan hanya kesamaan teks.

3.3 Dataset dan Sumber Data

Dataset yang digunakan dalam penelitian ini berupa dataset kode program bahasa *Python* yang disusun dalam bentuk pasangan kode (*pairwise*) untuk mendukung proses deteksi plagiarisme kode program. Pemilihan bahasa *Python* didasarkan pada ketersediaan parser yang stabil untuk pembentukan *Abstract Syntax Tree* (AST), serta luasnya penggunaan *Python* dalam konteks akademik, sehingga relevan untuk studi deteksi plagiarisme kode program.

3.3.1 Sumber Dataset

Sumber data dalam penelitian ini terdiri atas dataset publik dan dataset hasil konstruksi (modifikasi). Dataset publik diperoleh dari repositori kode sumber terbuka, seperti platform *GitHub* dan sumber dataset kode program yang tersedia secara publik. Dataset ini berisi kumpulan kode *Python* yang valid dan dapat dijalankan, serta merepresentasikan berbagai struktur sintaks dan pola pemrograman. Dataset publik digunakan sebagai sumber kode program asli, yang menjadi dasar pembentukan pasangan data dalam penelitian ini.

Selain itu, penelitian ini menggunakan dataset hasil konstruksi, yaitu dataset yang dibentuk dengan cara memodifikasi sebagian kode program dari dataset publik secara terkontrol. Proses modifikasi dilakukan dengan tetap mempertahankan logika program, sehingga perubahan yang dilakukan mencerminkan praktik plagiarisme kode yang umum terjadi di lingkungan akademik. Dataset hasil konstruksi ini memungkinkan pembentukan pasangan kode yang merepresentasikan berbagai jenis plagiarisme secara sistematis dan terukur.

3.3.2 Skema Pairwise Dataset (Kode A vs Kode B)

Dataset dalam penelitian ini disusun menggunakan skema *pairwise*, di mana setiap data terdiri dari sepasang kode program, yaitu kode A dan kode B. Setiap pasangan kode digunakan sebagai satu unit data yang akan dianalisis tingkat kemiripannya oleh sistem.

Pada skema ini, pasangan kode yang memiliki kesamaan struktur dan logika diberi label plagiarisme. Sedangkan pasangan kode yang tidak memiliki kesamaan signifikan diberi label tidak plagiarisme.

Skema *pairwise* dipilih karena sesuai dengan tujuan sistem, yaitu membandingkan dua kode program secara langsung, sebagaimana proses pemeriksaan plagiarisme yang umum dilakukan dalam lingkungan akademik. Selain itu, skema ini memungkinkan sistem untuk menghasilkan skor kemiripan yang merepresentasikan tingkat kesamaan antar dua kode secara kuantitatif.

Jumlah data yang digunakan dalam penelitian ini adalah minimal 1000 pasangan kode, sehingga mencukupi untuk proses pelatihan dan evaluasi model berbasis pembelajaran mesin.

3.3.3 Jenis Plagiarisme yang Dimodelkan

Jenis plagiarisme kode program yang dimodelkan dalam penelitian ini meliputi.

1. Type-1 (Copy-Paste Kosmetik)

Jenis plagiarisme ini ditandai dengan kesamaan struktur kode secara keseluruhan, dengan perbedaan terbatas pada aspek permukaan seperti format penulisan, spasi, atau komentar. Jenis ini digunakan untuk menguji kemampuan sistem dalam mengenali kemiripan struktur kode yang identik.

2. Type-2 (Rename Identifier)

Jenis plagiarisme ini melibatkan perubahan pada nama variabel, fungsi, atau parameter, tanpa mengubah struktur dan alur logika program. Jenis ini digunakan untuk menguji kemampuan sistem dalam mendeteksi plagiarisme yang telah mengalami penyamaran melalui penggantian identifier.

3. Type-3 Ringan (Uji Ketahanan)

Jenis plagiarisme ini melibatkan perubahan struktur ringan, seperti variasi bentuk ekspresi atau konstruksi kontrol sederhana, namun tetap mempertahankan

fungsi utama program. Jenis ini tidak menjadi fokus utama penelitian, tetapi digunakan sebagai uji ketahanan (*robustness test*) untuk mengevaluasi sejauh mana sistem mampu menangani variasi plagiarisme yang lebih kompleks.

3.3.4 Pembagian Dataset

Dataset dibagi ke dalam tiga bagian utama, yaitu data latih, data validasi, dan data uji.

1. Data latih (*training set*), digunakan untuk melatih model *Graph Neural Network* dalam mempelajari pola kemiripan kode program.
2. Data validasi (*validation set*), digunakan untuk melakukan penyesuaian parameter model serta mencegah terjadinya overfitting selama proses pelatihan.
3. Data uji (*testing set*), digunakan untuk mengevaluasi kinerja akhir sistem deteksi plagiarisme kode program.

Pembagian dataset dilakukan secara proporsional dan terkontrol, sehingga setiap jenis plagiarisme terdistribusi secara seimbang pada masing-masing bagian data. Dengan demikian, evaluasi kinerja sistem dapat dilakukan secara objektif dan mencerminkan kemampuan model dalam mendeteksi berbagai jenis plagiarisme kode program.

3.4 Pra-pemrosesan Data

Pra-pemrosesan data merupakan tahapan penting dalam penelitian ini, karena bertujuan untuk menyiapkan data kode program agar dapat diproses secara efektif oleh model *Graph Neural Network* (GNN). Pada tahap ini, kode program mentah diubah menjadi representasi struktural yang sesuai dengan kebutuhan sistem, tanpa mengubah logika dasar program.

Tahapan pra-pemrosesan data dalam penelitian ini meliputi pembersihan kode, parsing kode *Python*, pembentukan *Abstract Syntax Tree* (AST), normalisasi kode, konversi AST menjadi *graph*, serta pelabelan data.

3.4.1 Pembersihan Kode Program

Tahap awal pra-pemrosesan adalah pembersihan kode program. Pada tahap ini, kode program diperiksa untuk memastikan bahwa kode bersifat valid dan dapat diparsing dengan benar. Proses pembersihan meliputi penghapusan elemen-elemen yang tidak relevan terhadap struktur sintaks, seperti komentar dan spasi berlebih, serta penyesuaian format dasar kode jika diperlukan. Pembersihan kode dilakukan untuk meminimalkan gangguan pada proses parsing dan pembentukan AST.

3.4.2 Parsing Kode Python

Setelah proses pembersihan, kode program *Python* diproses menggunakan *parser* bahasa *Python* untuk mengubah kode sumber menjadi bentuk struktur sintaks. Proses parsing ini bertujuan untuk mengidentifikasi elemen-elemen sintaks utama dalam kode program, seperti pernyataan, ekspresi, dan blok kontrol. Hasil dari tahap ini menjadi dasar pembentukan *Abstract Syntax Tree*.

3.4.3 Pembentukan Abstract Syntax Tree (AST)

Abstract Syntax Tree (AST) dibentuk dari hasil parsing kode program *Python*. AST merepresentasikan struktur sintaks kode program dalam bentuk hierarki node, di mana setiap node merepresentasikan konstruksi sintaks tertentu. Dengan menggunakan AST, detail permukaan seperti format penulisan dan komentar tidak lagi memengaruhi representasi kode, sehingga struktur logika program dapat direpresentasikan secara lebih jelas dan konsisten.

3.4.4 Normalisasi Kode

Untuk meningkatkan konsistensi representasi, dilakukan normalisasi kode pada tahap ini. Normalisasi mencakup penyeragaman elemen-elemen tertentu dalam AST, seperti penyamaan penamaan literal atau pengaburan (*abstraction*) terhadap nama variabel dan fungsi. Proses ini bertujuan untuk mengurangi pengaruh perbedaan penamaan yang tidak berkaitan dengan struktur logika program, khususnya pada kasus plagiarisme *Type-2* (*rename identifier*).

3.4.5 Konversi AST ke Graph

AST yang telah terbentuk kemudian dikonversi ke dalam bentuk *graph*, sehingga dapat diproses oleh model *Graph Neural Network*. Pada tahap ini, setiap *node* dalam AST dipetakan menjadi *node* pada *graph*, sedangkan hubungan hierarkis antar *node* direpresentasikan sebagai *edge*. Representasi *graph* ini memungkinkan pemodelan hubungan struktural antar elemen kode secara lebih fleksibel dan mendukung proses pembelajaran berbasis *graph*.

3.4.6 Pelabelan Data

Dalam pendekatan *supervised learning*, pelabelan data merupakan proses pemberian kelas (*ground truth*) pada setiap sampel untuk menjadi acuan dalam proses pelatihan model. Label digunakan untuk menghitung fungsi *loss* dan memperbarui parameter model, sehingga model dapat membentuk batas keputusan (*decision boundary*) yang membedakan antar kelas secara optimal.

Pada penelitian ini, pelabelan dilakukan pada dataset berbentuk pasangan kode (*pairwise dataset*), yaitu kode A dan kode B. Setiap pasangan diberi label biner sebagai berikut:

1. Label 1 (Plagiarisme) untuk pasangan kode yang memiliki kesamaan struktur dan logika signifikan, khususnya pada Type-1 (*copy-paste kosmetik*) dan Type-2 (*rename identifier*).
2. Label 0 (Tidak Plagiarisme) untuk pasangan kode yang tidak menunjukkan kesamaan struktural yang relevan.

Pelabelan dilakukan secara terkontrol berdasarkan skema konstruksi dataset, di mana pasangan plagiarisme dihasilkan melalui modifikasi terarah tanpa mengubah logika program, sedangkan pasangan tidak plagiarisme berasal dari kode yang tidak saling berkaitan.

Label ini digunakan untuk melatih model Graph Neural Network (GNN) dalam mempelajari pola kemiripan antar graph Abstract Syntax Tree (AST), sehingga sistem dapat menghasilkan skor kemiripan dan keputusan klasifikasi berdasarkan ambang batas (*threshold*) yang ditentukan.

3.5 Representasi Graph Abstract Syntax Tree

Pada penelitian ini, *Abstract Syntax Tree* (AST) yang dihasilkan dari proses parsing kode program direpresentasikan ke dalam bentuk *graph* agar dapat diproses oleh *Graph Neural Network* (GNN). Representasi *graph* dipilih karena mampu memodelkan hubungan struktural antar elemen kode secara eksplisit, sehingga informasi sintaks dan logika program dapat dipertahankan dan dipelajari secara lebih efektif.

AST pada dasarnya memiliki struktur hierarkis yang secara alami dapat dimodelkan sebagai *graph* berarah. Dengan mengubah AST menjadi *graph*, setiap elemen sintaks kode program dan hubungan antar elemen tersebut dapat direpresentasikan secara formal dalam bentuk *node* dan *edge*.

3.5.1 Definisi Node dan Edge

Dalam representasi *graph* AST, node merepresentasikan elemen sintaks dari kode program, sedangkan *edge* merepresentasikan hubungan struktural antar elemen tersebut. Setiap node pada graph berkorespondensi langsung dengan *node* pada AST, seperti pernyataan, ekspresi, atau konstruksi kontrol alur.

Sementara itu, *edge* digunakan untuk merepresentasikan hubungan hierarkis antar node, khususnya hubungan *parent–child* yang menunjukkan bahwa suatu elemen sintaks merupakan bagian dari elemen yang lebih besar. Dengan definisi ini, *graph* AST dapat dinyatakan sebagai struktur $G = (V, E)$ di mana V merupakan himpunan *node* dan E merupakan himpunan *edge* yang menghubungkan node-node tersebut.

3.5.2 Jenis Node Abstract Syntax Tree yang Digunakan

Jenis *node* AST yang digunakan dalam penelitian ini mencerminkan konstruksi sintaks utama pada bahasa pemrograman Python. Node-node tersebut meliputi:

1. node pernyataan (*statement*), seperti *assignment* dan *return*,
2. node ekspresi (*expression*),
3. node kontrol alur, seperti *if*, *for*, dan *while*,
4. node operasi, seperti *binary operation*,
5. node pemanggilan fungsi (*function call*).

Pemilihan jenis node ini bertujuan untuk menangkap struktur dan alur logika program secara menyeluruh, tanpa bergantung pada detail permukaan seperti nama variabel atau format penulisan kode.

Hubungan antar *node* dalam *graph* AST dibangun berdasarkan relasi struktural yang terdapat pada AST. Relasi utama yang digunakan adalah hubungan induk–anak (*parent–child*), yang mencerminkan struktur hierarkis kode program. Hubungan ini memungkinkan sistem untuk memahami bagaimana suatu konstruksi program tersusun dari sub-konstruksi yang lebih kecil.

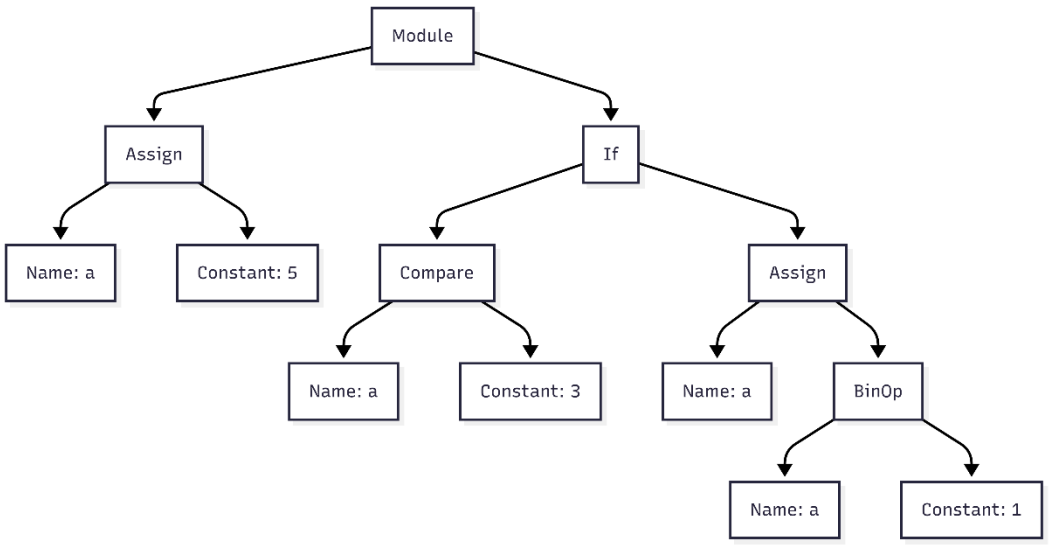
Relasi ini sangat penting karena mencerminkan alur logika dan ketergantungan antar bagian kode, sehingga kesamaan struktur antar dua kode program dapat dikenali meskipun terjadi perubahan pada tingkat permukaan.

3.5.3 Ilustrasi Graph Abstract Syntax Tree

Untuk memperjelas konsep representasi graph AST, digunakan potongan kode Python sederhana sebagai ilustrasi berikut:

```
a = 5
if a > 3:
    a = a + 1
```

Potongan kode tersebut kemudian diparsing untuk menghasilkan *Abstract Syntax Tree* (AST), yang selanjutnya direpresentasikan dalam bentuk *graph*. Pada *graph* AST, setiap konstruksi sintaks seperti *assignment*, *if statement*, *comparison*, dan *binary operation* direpresentasikan sebagai *node*, sedangkan hubungan hierarkis antar konstruksi tersebut direpresentasikan sebagai *edge*.



Gambar 3. 2 Representasi *Abstract Syntax Tree* dalam bentuk *graph* pada potongan kode *Python*.

Ilustrasi *graph* AST dari potongan kode di atas ditunjukkan pada Gambar 3.2, yang menggambarkan hubungan struktural antar elemen kode secara visual. Ilustrasi ini menunjukkan bagaimana struktur logika program direpresentasikan dalam bentuk graph yang dapat diproses oleh model berbasis graph.

3.5.4 Alasan Pemilihan Representasi Graph AST

Representasi AST dalam bentuk graph dipilih karena beberapa pertimbangan utama. Pertama, *graph* mampu mempertahankan struktur sintaks dan relasi antar elemen kode, sehingga perubahan kosmetik seperti format penulisan atau komentar tidak memengaruhi representasi inti kode program. Kedua, representasi *graph* memungkinkan *Graph Neural Network* untuk mempelajari pola kemiripan struktur kode melalui mekanisme pertukaran informasi antar node.

Dengan menggunakan representasi *graph* AST, sistem deteksi plagiarisme dalam penelitian ini dapat membandingkan dua kode program berdasarkan kesamaan struktur dan alur logika, bukan hanya kesamaan teks. Pendekatan ini

membuat sistem lebih robust dalam mendeteksi plagiarisme *Type-1* (*copy-paste* kosmetik) dan *Type-2* (*rename identifier*), serta memungkinkan pengujian ketahanan terhadap variasi plagiarisme Type-3 ringan.

3.6 Arsitektur Model Graph Neural Network (GNN)

Pada penelitian ini, *Graph Neural Network* (GNN) digunakan untuk mempelajari representasi struktur kode program yang telah direpresentasikan dalam bentuk *graph Abstract Syntax Tree* (AST). GNN dipilih karena kemampuannya dalam memodelkan data berbentuk *graph* serta menangkap hubungan relasional antar node melalui mekanisme *message passing*.

Model GNN pada penelitian ini bekerja dengan cara mengagregasi informasi dari *node-node* tetangga untuk membentuk representasi *node* yang semakin kaya pada setiap lapisan (l*ayer*). Proses ini dilakukan secara berulang sehingga informasi struktural dari seluruh *graph* AST dapat terdistribusi ke setiap node.

3.6.1 Mekanisme Message Passing pada Graph Neural Network (GNN)

Secara umum, proses pembaruan representasi node pada lapisan ke-k dalam GNN dapat dinyatakan sebagai berikut:

Rumus 3.1 Persamaan Pembaruan Representasi Node (*Message Passing*).

$$h_v^{(k)} = \sigma\left(W^{(k)} \cdot AGG\left(\left\{ h_u^{(k-1)} \mid u \in \mathcal{N}(v) \right\}\right)\right)$$

Tabel 3. 1 Simbol Message Passing GNN.

Simbol	Keterangan
$h_v^{(k)}$	Vektor representasi (*embedding*) node v pada lapisan ke-k.
$h_u^{(k-1)}$	Vektor representasi node tetangga u pada lapisan sebelumnya.
$\mathcal{N}(v)$	Himpunan node tetangga dari node v
$W^{(k)}$	Matriks bobot yang dipelajari pada lapisan ke-k.
$AGG(\cdot)$	Fungsi agregasi (misalnya penjumlahan atau rata-rata).
$\sigma(\cdot)$	Fungsi aktivasi non-linear.
k	Indeks lapisan pada model GNN.

Tabel ini bertujuan untuk memastikan bahwa setiap simbol matematis yang digunakan dapat dipahami dengan jelas dan konsisten.

3.6.2 Pembentukan Representasi Graph (Graph Embedding)

Setelah proses *message passing* pada seluruh *node* selesai, representasi *graph* AST diperoleh dengan melakukan agregasi terhadap seluruh *embedding node*. Representasi *graph* ini mencerminkan struktur keseluruhan kode program dan digunakan sebagai dasar perhitungan kemiripan antar dua kode program.

Pada penelitian ini, proses pembentukan *embedding graph* dilakukan secara terpisah untuk kode A dan kode B, sehingga dihasilkan dua buah *embedding graph* yang dapat dibandingkan.

3.6.3 Perhitungan Kemiripan Antar Graph

Embedding graph dari kode A dan kode B kemudian dibandingkan untuk menghasilkan skor kemiripan. Skor ini merepresentasikan tingkat kesamaan struktur antar dua *graph* AST. Berdasarkan skor tersebut, sistem selanjutnya menentukan label keputusan, yaitu plagiarisme atau tidak plagiarisme, sesuai dengan ambang batas yang telah ditetapkan.

3.6.4 Alasan Pemilihan Arsitektur GNN

Arsitektur GNN dipilih karena mampu mempelajari pola kemiripan struktur kode secara efektif melalui pemodelan hubungan antar *node*. Dengan mengombinasikan representasi *graph* AST dan mekanisme pembelajaran GNN, sistem dalam penelitian ini mampu mendeteksi plagiarisme kode program berdasarkan kesamaan struktur logika, bukan hanya kesamaan teks.

3.7 Proses Pelatihan Model

Proses pelatihan model dalam penelitian ini bertujuan untuk melatih *Graph Neural Network* (GNN) agar mampu mempelajari pola kemiripan struktur kode program yang direpresentasikan dalam bentuk *graph Abstract Syntax Tree* (AST). Pelatihan dilakukan menggunakan pendekatan *supervised learning*, di mana setiap pasangan kode program telah diberi label kelas, yaitu plagiarisme dan tidak plagiarisme.

3.7.1 Skema Pelatihan Model

Model dilatih menggunakan skema pairwise, di mana setiap data pelatihan terdiri dari dua *graph* AST, yaitu *graph* dari kode A dan graph dari kode B. Kedua *graph* tersebut diproses melalui arsitektur GNN yang sama (*shared weights*) untuk menghasilkan *embedding graph* masing-masing. *Embedding* yang dihasilkan kemudian dibandingkan untuk memprediksi tingkat kemiripan dan label klasifikasi.

Pelatihan dilakukan secara iteratif dalam beberapa *epoch*, di mana pada setiap iterasi model memperbarui bobotnya berdasarkan kesalahan prediksi yang dihasilkan pada data latih.

3.7.2 Fungsi Loss dan Optimisasi

Untuk melatih model, digunakan fungsi *loss* klasifikasi yang mengukur perbedaan antara label prediksi dan label sebenarnya. Fungsi *loss* ini digunakan sebagai dasar untuk memperbarui bobot model selama proses pelatihan. Proses optimisasi dilakukan menggunakan algoritma optimisasi standar, seperti *Adam* optimizer, yang umum digunakan dalam pelatihan model pembelajaran mendalam karena stabilitas dan efisiensinya.

Pada penelitian ini, pemilihan fungsi *loss* dan *optimizer* bertujuan untuk memastikan proses pelatihan berjalan secara konvergen dan menghindari permasalahan seperti *overfitting* atau *underfitting*.

3.7.3 Pengaturan Parameter Pelatihan

Beberapa parameter pelatihan yang digunakan dalam penelitian ini meliputi:

1. jumlah *epoch* pelatihan,
2. ukuran *batch*,

3. laju pembelajaran (*learning rate*),
4. jumlah lapisan pada model GNN.

Nilai parameter tersebut ditentukan melalui proses eksperimen dan penyesuaian menggunakan data validasi, sehingga diperoleh konfigurasi yang memberikan kinerja terbaik tanpa menyebabkan *overfitting*.

3.7.4 Penggunaan Data Validasi

Selama proses pelatihan, data validasi digunakan untuk memantau kinerja model pada data yang tidak digunakan dalam pelatihan langsung. Evaluasi pada data validasi dilakukan secara berkala untuk memastikan bahwa model tidak hanya menghafal data latih, tetapi juga mampu melakukan generalisasi dengan baik.

Jika terjadi penurunan kinerja pada data validasi, proses pelatihan dapat dihentikan atau parameter model disesuaikan untuk menjaga performa model.

3.7.5 Hasil Pelatihan Model

Hasil dari proses pelatihan model berupa model GNN terlatih yang mampu menghasilkan *embedding graph* untuk setiap kode program dan memprediksi tingkat kemiripan antar pasangan kode. Model terlatih ini selanjutnya digunakan pada tahap evaluasi untuk mengukur kinerja sistem deteksi plagiarisme kode program menggunakan data uji.

Dengan proses pelatihan yang terstruktur dan terkontrol, model diharapkan mampu mempelajari pola kemiripan struktur kode secara efektif dan memberikan hasil prediksi yang akurat pada tahap evaluasi.

3.8 Metode Evaluasi

Metode evaluasi digunakan untuk mengukur kinerja sistem deteksi plagiarisme kode program yang dikembangkan dalam penelitian ini. Evaluasi

dilakukan untuk menilai sejauh mana model *Graph Neural Network* (GNN) mampu mendeteksi kemiripan kode program secara akurat pada skema *pairwise* (kode A vs kode B). Hasil evaluasi diperoleh dari pengujian model menggunakan data uji yang tidak terlibat dalam proses pelatihan.

3.8.1 Skema Evaluasi

Pada penelitian ini, sistem menghasilkan dua jenis keluaran, yaitu:

1. Skor kemiripan, yang merepresentasikan tingkat kesamaan antara dua kode program.
2. Label klasifikasi, yaitu plagiarisme atau tidak plagiarisme, yang ditentukan berdasarkan skor kemiripan dan ambang batas (*threshold*) tertentu.

Evaluasi dilakukan terhadap hasil klasifikasi dua kelas dengan membandingkan label prediksi sistem dan label sebenarnya pada data uji.

3.8.2 Confusion Matrix

Untuk mengevaluasi hasil klasifikasi, digunakan *confusion matrix* yang terdiri dari empat komponen, yaitu:

1. *True Positive* (TP): pasangan kode yang merupakan plagiarisme dan diprediksi sebagai plagiarisme.
2. *True Negative* (TN): pasangan kode yang bukan plagiarisme dan diprediksi sebagai tidak plagiarisme.
3. *False Positive* (FP): pasangan kode yang bukan plagiarisme tetapi diprediksi sebagai plagiarisme.
4. *False Negative* (FN): pasangan kode yang merupakan plagiarisme tetapi diprediksi sebagai tidak plagiarisme.

Nilai-nilai tersebut berfungsi sebagai fondasi utama dalam penghitungan metrik evaluasi untuk mengukur performa model yang dikembangkan. Adapun beberapa metrik evaluasi yang diterapkan secara spesifik dalam penelitian ini meliputi.

1. Akurasi (Accuracy)

Akurasi mengukur proporsi prediksi yang benar terhadap seluruh data uji.

Rumus 3.2 Akurasi (*Accuracy*).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

2. Presisi (Precision)

Presisi mengukur ketepatan sistem dalam memprediksi pasangan kode sebagai plagiarisme.

Rumus 3.4 Presisi (*Precision*).

$$Precision = \frac{TP}{TP + FP}$$

3. Recall

Recall mengukur kemampuan sistem dalam mendeteksi seluruh pasangan kode yang benar-benar merupakan plagiarisme.

Rumus 3.5 *Recall*.

$$Recall = \frac{TP}{TP\ + FN}$$

4. F1-Score

F1-score merupakan nilai harmonik antara presisi dan recall yang digunakan untuk menilai keseimbangan antara keduanya.

Rumus 3.6 *F1-Score*.

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

3.8.3 Evaluasi Skor Kemiripan

Selain evaluasi berbasis klasifikasi, skor kemiripan yang dihasilkan oleh sistem juga dianalisis untuk melihat distribusi nilai kemiripan antara pasangan kode plagiarisme dan tidak plagiarisme. Analisis ini bertujuan untuk memastikan bahwa skor kemiripan yang dihasilkan mampu membedakan kedua kelas secara konsisten.

Ambang batas (*threshold*) digunakan untuk mengonversi skor kemiripan menjadi label keputusan. Nilai *threshold* ditentukan berdasarkan hasil pengujian pada data validasi agar menghasilkan keseimbangan yang optimal antara presisi dan recall.

3.8.3 Tujuan Evaluasi

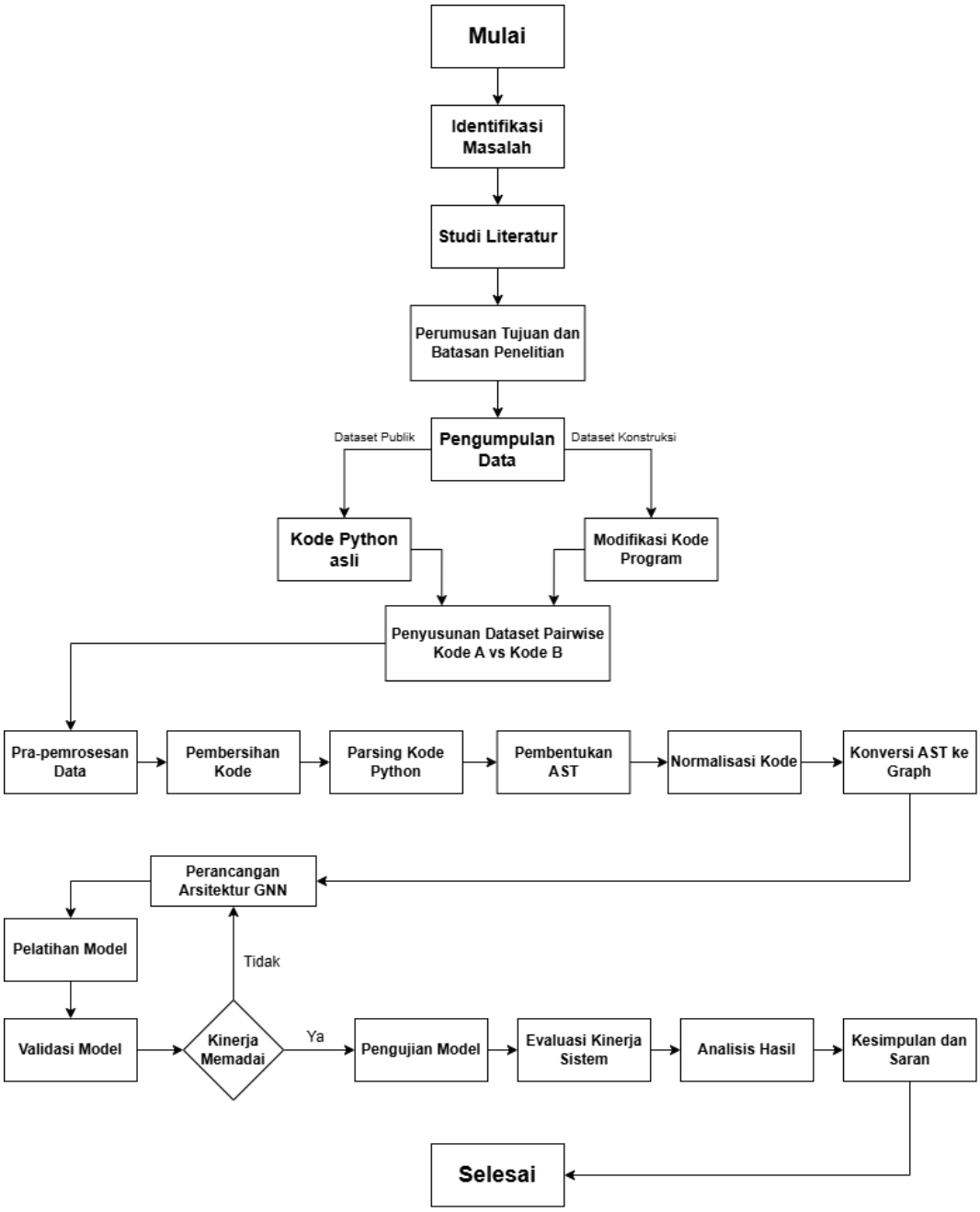
Melalui metode evaluasi ini, kinerja sistem dapat diukur secara objektif dan terukur. Hasil evaluasi digunakan untuk:

1. menilai efektivitas pendekatan AST dan GNN dalam mendeteksi plagiarisme kode program,
2. membandingkan performa sistem pada berbagai jenis plagiarisme,
3. serta menjadi dasar analisis dan pembahasan pada BAB selanjutnya.

3.9 Alur Penelitian

Diagram alir penelitian ini disusun sebagai kerangka kerja sistematis guna memetakan seluruh tahapan eksperimen dari fase perencanaan hingga penarikan kesimpulan. Representasi visual ini membantu peneliti dalam memahami setiap tingkatan proses yang harus dilalui seperti ekstraksi struktur melalui Abstract

Syntax Tree, pemrosesan data menggunakan Graph Neural Network, hingga tahap evaluasi nilai kemiripan. Dengan adanya skema yang terstruktur ini, seluruh proses deteksi plagiarisme dapat berjalan secara terarah serta memastikan hasil analisis yang objektif dan mudah dipahami.



Gambar 3. 3 Diagram Alir Penelitian

Untuk memperjelas tahapan penelitian secara keseluruhan, alur penelitian ditampilkan dalam bentuk diagram alur penelitian seperti yang terlihat pada Gambar 3.4. Alur penelitian pada penelitian ini disusun secara sistematis untuk memastikan bahwa setiap tahapan penelitian dilakukan secara terstruktur dan saling berkesinambungan. Alur penelitian dimulai dari tahap perumusan masalah hingga tahap evaluasi dan analisis hasil, sesuai dengan tujuan penelitian yaitu mengembangkan sistem deteksi plagiarisme kode program berbasis *Abstract Syntax Tree* (AST) dan *Graph Neural Network* (GNN).

Tahapan awal penelitian diawali dengan identifikasi masalah dan studi literatur, yang bertujuan untuk memahami permasalahan plagiarisme kode program serta pendekatan-pendekatan yang telah dikembangkan pada penelitian terdahulu. Hasil studi literatur digunakan sebagai dasar dalam perumusan tujuan dan batasan penelitian.

Selanjutnya, dilakukan pengumpulan dan penyusunan dataset, yang bersumber dari dataset publik dan dataset hasil konstruksi. Dataset disusun dalam skema *pairwise* (kode A vs kode B) dan diberi label sesuai dengan jenis plagiarisme yang dimodelkan.

Pada tahap berikutnya, dilakukan pra-pemrosesan data, yang meliputi pembersihan kode, *parsing* kode *Python*, pembentukan Abstract Syntax Tree (AST), normalisasi kode, serta konversi AST ke dalam bentuk *graph*. *Graph* AST yang dihasilkan kemudian digunakan sebagai masukan bagi model *Graph Neural Network*.

Tahap selanjutnya adalah perancangan dan pelatihan model GNN, di mana model dilatih menggunakan data latih dan divalidasi menggunakan data validasi.

Setelah proses pelatihan selesai, model diuji menggunakan data uji untuk memperoleh hasil evaluasi.

Tahap akhir penelitian adalah evaluasi dan analisis hasil, yang dilakukan dengan menggunakan metrik evaluasi yang telah ditentukan. Hasil evaluasi dianalisis untuk menilai kinerja sistem serta menjawab rumusan masalah penelitian.

BAB IV

HASIL DAN PEMBAHASAN

4.1 Implementasi Sistem

Pada penelitian ini, sistem deteksi plagiarisme kode program dikembangkan dengan memanfaatkan pendekatan berbasis struktur menggunakan *Abstract Syntax Tree* (AST) dan *Graph Neural Network* (GNN). Sistem dirancang untuk mampu membandingkan dua kode program dan menghasilkan nilai kemiripan (*similarity score*) yang digunakan sebagai dasar dalam menentukan tingkat plagiarisme.

Secara umum, alur kerja sistem dimulai dari proses input dua kode program, yang kemudian diproses melalui beberapa tahapan utama, yaitu parsing kode menjadi AST, konversi AST ke dalam bentuk *graph*, ekstraksi fitur *graph*, dan perhitungan tingkat kemiripan menggunakan model GNN. Hasil akhir dari sistem berupa nilai *similarity* yang diinterpretasikan ke dalam kategori plagiarisme, yaitu rendah (*low*), sedang (*medium*), dan tinggi (*high*).

4.1.1 Arsitektur Sistem

Arsitektur sistem yang dikembangkan terdiri dari beberapa komponen utama yang saling terintegrasi. Tahapan proses dalam sistem dalam dijelaskan sebagai berikut:

1. Input Kode Program

Pengguna memasukkan dua buah kode program dalam bahasa Python melalui antarmuka sistem. Kode yang dimasukkan kemudian diproses untuk dianalisis tingkat kemiripannya.

2. Parsing ke Abstract Syntax Tree (AST)

Setiap kode program diubah menjadi representasi AST. AST merepresentasikan struktur sintaksis kode dalam bentuk pohon, sehingga informasi struktural dari program dapat dipertahankan tanpa terpengaruh oleh format penulisan.

3. Konversi AST ke Graph

Struktur AST kemudian dikonversi menjadi *graph*, di mana setiap *node* merepresentasikan elemen sintaksis, dan *edge* merepresentasikan hubungan antar *node*.

4. Tranformasi ke PyTorch Geometric (PyG)

Graph yang dihasilkan diubah menjadi format yang dapat diproses oleh GNN menggunakan pustaka *PyTorch Geometric*. Pada tahap ini, fitur *node* diekstraksi dan direpresentasikan dalam bentuk vektor numerik.

5. Pemrosesan Menggunakan Graph Neural Network (GNN)

Dua *graph* yang dihasilkan dari masing-masing kode program diproses menggunakan model GNN berbasis arsitektur *Siamese Network*. Model ini menghasilkan representasi embedding dari masing-masing *graph*, yang kemudian dibandingkan untuk menghitung kemiripan.

6. Perhitungan Similarity Score

Output dari model berupa nilai *logit* yang kemudian diubah menggunakan fungsi *sigmoid* menjadi nilai similarity dalam rentang 0 hingga 1.

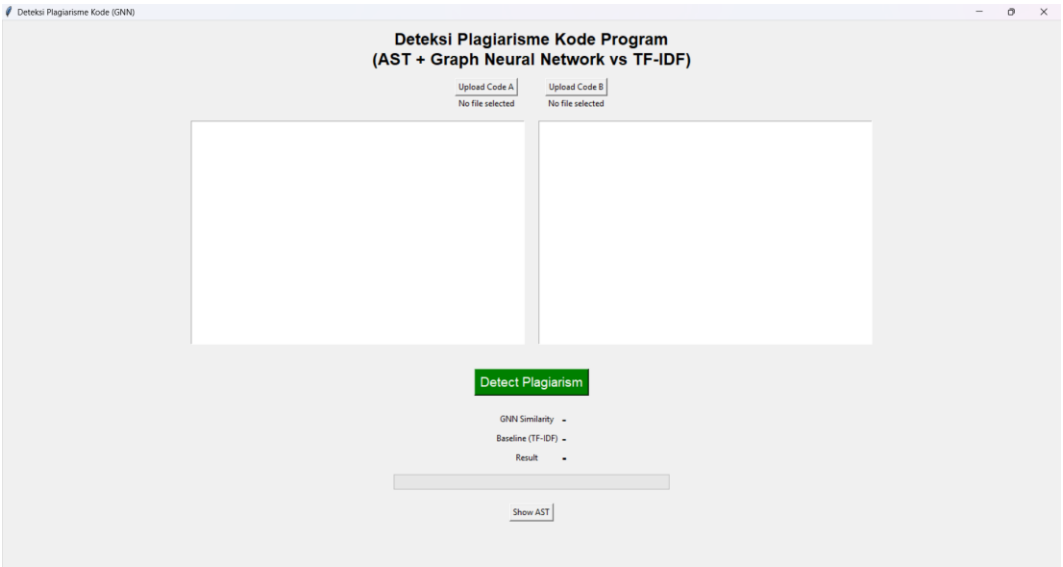
7. Interpretasi Hasil

Kategori nilai *similarity* dalam sistem ini terbagi menjadi tiga tingkatan utama, di mana nilai yang berada diatas 0,85 diklasifikasikan sebagai Plagiarisme

Tinggi (*High*). Sementara itu, untuk rentang nilai antara 0,06 hingga 0,85 masuk ke dalam kategori Plagiarisme Sedang (*Medium*), dan Nilai yang berada di bawah 0,06 dianggap sebagai Tidak Plagiarisme (*Low*).

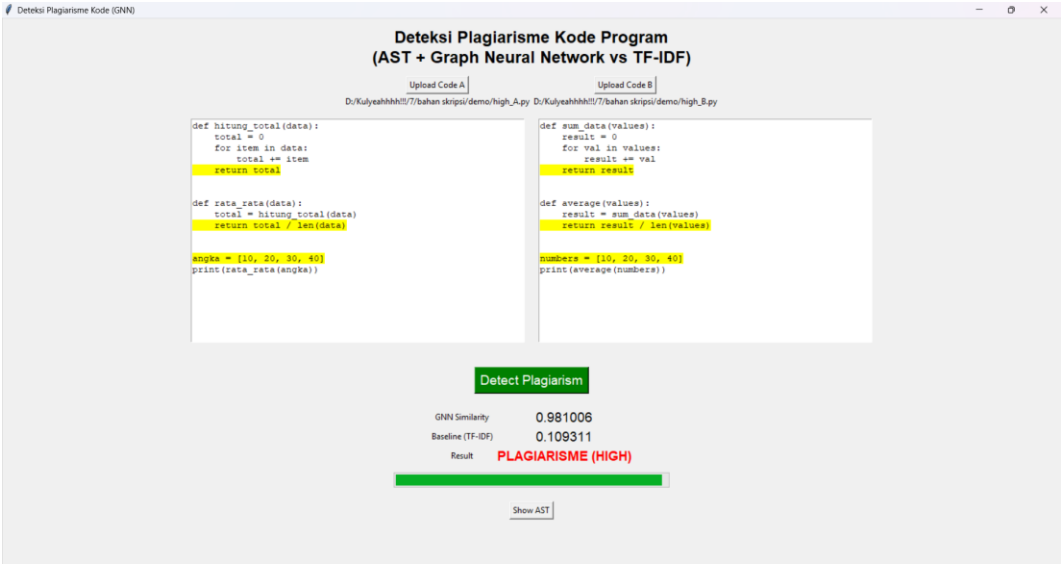
4.1.2 Implementasi Antarmuka Pengguna (GUI)

Sistem yang dikembangkan dilengkapi dengan antarmuka pernggguna berbasis *Graphical User Interface* (GUI) yang dibangun menggunakan pustaka Tkinter. Antarmuka ini dirancang untuk mempermudah pengguna dalam melakukan proses deteksi plagiarisme tanpa perlu berinteraksi dengan kode program.



Gambar 4. 1 Tampilan Utama GUI.

Gambar 4.1 menunjukkan tampilan utama sistem yang terdiri dari dua area utama untuk menampilkan kode program yang diunggah oleh pengguna. Masing-masing area merepresentasikan kode program pertama dan kedua, sehingga pengguna dapat melakukan perbandingan secara visual. Selain itu, tersedia tombol untuk mengunggah file kode program serta tombol untuk menjalankan proses deteksi plagiarisme.



Gambar 4. 2 Hasil Deteksi dan Score.

Setelah proses deteksi dilakukan, sistem akan menampilkan nilai kemiripan (*similarity score*) yang dihasilkan oleh model *Graph Neural Network* (GNN). Selain itu, sistem juga menampilkan nilai kemiripan dari metode baseline berbasis TF-IDF sebagai pembanding. Hasil deteksi kemudian diklasifikasikan ke dalam kategori plagiarisme, yaitu rendah, sedang, atau tinggi, yang ditampilkan dengan perbedaan warna untuk memudahkan interpretasi. Nilai *similarity* juga divisualisasikan dalam bentuk grafik serta indikator *progress bar*.

Selain menampilkan hasil numerik, sistem juga menyediakan fitur penyorotan (*highlight*) pada bagian kode yang memiliki tingkat kemiripan tinggi. Fitur ini memungkinkan pengguna untuk secara langsung melihat bagian-bagian kode yang dianggap mirip oleh sistem, sehingga dapat membantu dalam proses analisis lebih lanjut. Penyorotan dilakukan dengan membandingkan setiap baris kode dan menandai bagian yang memiliki tingkat kemiripan tertentu.

Dengan adanya berbagai fitur tersebut, antarmuka pengguna yang dikembangkan tidak hanya berfungsi sebagai alat untuk menjalankan sistem, tetapi

juga sebagai media visualisasi yang membantu pengguna dalam memahami hasil deteksi plagiarisme secara lebih mendalam.

4.1.3 Integrasi Model dan Sistem

Model *Graph Neural Network* (GNN) yang digunakan dalam penelitian ini merupakan hasil pelatihan yang telah disimpan dan kemudian diintegrasikan ke dalam sistem melalui proses pemuatan (*loading model*). Dengan demikian, model dapat digunakan secara langsung pada tahap deteksi tanpa memerlukan pelatihan ulang.

Integrasi model dilakukan dengan menghubungkan seluruh tahapan pemrosesan, mulai dari input kode program hingga menghasilkan nilai kemiripan. Kode yang diunggah oleh pengguna diproses menjadi *Abstract Syntax Tree* (AST), kemudian dikonversi ke dalam bentuk *graph* dan direpresentasikan sebagai input bagi model GNN. Model selanjutnya menghasilkan nilai *similarity score* yang digunakan untuk menentukan tingkat plagiarisme.

Sistem dirancang agar proses tersebut berjalan secara otomatis, sehingga pengguna hanya perlu memasukkan kode program tanpa harus memahami mekanisme internal yang kompleks. Selain itu, sistem juga mengintegrasikan metode baseline berbasis TF-IDF sebagai pembanding, sehingga hasil yang diperoleh tidak hanya bersifat deteksi, tetapi juga memberikan evaluasi komparatif.

Dengan integrasi ini, sistem mampu bekerja secara *end-to-end*, mulai dari tahap input hingga menghasilkan output berupa nilai kemiripan dan interpretasi hasil, sehingga menunjukkan bahwa model yang diusulkan telah diimplementasikan secara nyata dan dapat digunakan secara langsung.

4.2 Hasil Pelatihan Model

Pada tahap ini, dilakukan proses pelatihan model *Graph Neural Network* (GNN) untuk mendeteksi kemiripan antar kode program berdasarkan representasi struktur yang dihasilkan dari *Abstract Syntax Tree* (AST). Proses pelatihan bertujuan untuk menghasilkan model yang mampu membedakan pasangan kode yang termasuk plagiarisme dan yang tidak, dengan tingkat akurasi yang tinggi.

Pelatihan model dilakukan menggunakan dataset pasangan kode program yang telah disiapkan sebelumnya. Dataset tersebut terdiri dari pasangan kode dengan label plagiarisme dan non-plagiarisme, sehingga model dapat mempelajari pola kesamaan dan perbedaan struktur kode secara efektif. Hasil dari proses pelatihan ini kemudian dievaluasi menggunakan beberapa metrik performa untuk mengukur kualitas model yang dihasilkan.

4.2.1 Dataset dan Pembagian Data

Dataset yang digunakan dalam penelitian ini terdiri pasangan kode program yang telah melalui proses pengumpulan dan praproses. Data diperoleh dari kombinasi data nyata yang bersumber dari berbagai kode program publik serta data hasil rekayasa (*synthetic data*) yang dihasilkan melalui modifikasi kode, seperti perubahan nama variabel, struktur, dan logika program untuk mensimulasikan berbagai tingkat plagiarisme.

Secara keseluruhan, dataset berjumlah 1304 pasangan kode program yang masing-masing diberi label sebagai plagiarisme (1) dan non-plagiarisme (0). Distribusi dataset berdasarkan label dapat dilihat pada Tabel 4.1.

Tabel 4. 1 Distribusi Dataset.

No	Kategori	Label	Jumlah Data
1.	Plagiarisme	1	978
2.	Non-Plagiarisme	0	326
Total		-	1034

Berdasarkan Tabel 4.1, dataset memiliki distribusi yang tidak seimbang, di mana jumlah data plagiarisme lebih dominan dibandingkan dengan non-plagiarisme. Kondisi ini tetap dipertahankan untuk mencerminkan variasi kasus yang lebih realistis dalam proses pembelajaran model.

Dataset kemudian dibagi menjadi data pelatihan (*training data*) dan data pengujian (*testing data*) dengan perbandingan 80:20 secara acak. Pembagian ini bertujuan untuk melatih model serta mengevaluasi kemampuannya dalam melakukan generalisasi terhadap data yang belum pernah dilihat sebelumnya. Distribusi pembagian dataset ditunjukan pada Tabel 4.2.

Tabel 4. 2 Pembagian dataset.

No	Jenis Dataset	Jumlah Data
1.	Training	1034
2.	Testing	261
Total		1304

Dengan pembagian tersebut, proses pelatihan dan evaluasi dapat dilakukan secara terpisah dan objektif, sehingga model yang dihasilkan diaharapkan memiliki performa yang baik dalam mendeteksi plagiarisme kode program.

4.2.2 Hasil Pelatihan Model

Proses pelatihan model dilakukan dengan menggunakan dataset yang telah dibagi menjadi data pelatihan (*training data*) dan data pengujian (*testing data*). Model yang digunakan merupakan *Graph Neural Network* (GNN) berbasis arsitektur *Siamese Network* yang dirancang untuk mempelajari tingkat kemiripan antara dua representasi *graph* dari kode program.

Selama proses pelatihan, model belajar untuk mengidentifikasi pola kesamaan dan perbedaan struktur kode program berdasarkan representasi *Abstract Syntax Tree* (AST) yang telah dikonversi menjadi *graph*. Proses ini memungkinkan model untuk memahami hubungan struktural antar elemen kode, sehingga tidak hanya bergantung pada kesamaan teks secara langsung.

Setelah proses pelatihan selesai, performa model dievaluasi menggunakan beberapa metrik evaluasi yaitu *accuracy*, *precision*, dan *F1-Score*. Hasil evaluasi model dapat dilihat pada Tabel 4.3.

Tabel 4. 3 Hasil Evaluasi Model GNN.

No	Metrik	Nilai
1.	Accuracy	0.9945848375451264
2.	Precision	0.9948979591836735
3.	Recall	0.9974424552429667
4.	F1-Score	0.9961685823754789

Berdasarkan Tabel 4.3, model menunjukkan performa yang sangat tinggi pada seluruh metrik evaluasi. Nilai *accuracy* sebesar 0.9946 menunjukkan bahwa model mampu mengklasifikasikan sebagian besar pasangan kode program dengan

benar. Nilai Precision yang tinggi menunjukkan bahwa prediksi plagiarisme yang dihasilkan oleh model memiliki tingkat keakuratan yang sangat baik.

Selain itu, nilai *recall* yang mendekati 1 menunjukkan bahwa model mampu mendeteksi hampir seluruh kasus plagiarisme yang terdapat dalam dataset. Hal ini penting dalam konteks deteksi plagiarisme, di mana kegagalan dalam mendeteksi kasus plagiarisme (*false negative*) perlu diminimalkan.

Nilai *F1-Score* yang tinggi juga menunjukkan bahwa model memiliki keseimbangan yang baik antara *precision* dan *recall*, sehingga mampu memberikan performa yang stabil dalam berbagai kondisi data.

Secara keseluruhan, hasil pelatihan menunjukkan bahwa model GNN yang dikembangkan mampu mempelajari pola kemiripan kode program secara efektif dan memberikan performa yang sangat baik dalam mendeteksi plagiarisme.

4.2.3 Evaluasi Performa Model

Berdasarkan hasil pelatihan, model Graph Neural Network (GNN) menunjukkan performa yang sangat baik dalam mendeteksi plagiarisme kode program, yang ditunjukkan oleh nilai *accuracy*, *precision*, *recall*, dan *F1-Score* yang tinggi.

Nilai *accuracy* yang mendekati 1 menunjukkan bahwa model mampu mengklasifikasikan sebagian besar pasangan kode dengan benar. Selain itu, *precision* yang tinggi mengindikasikan bahwa prediksi plagiarisme yang dihasilkan model memiliki tingkat keakuratan yang baik. Sementara itu, nilai *recall* yang tinggi menunjukkan bahwa model mampu mendeteksi hampir seluruh kasus plagiarisme, sehingga meminimalkan kesalahan berupa *false negative*. Nilai *F1-*

score yang tinggi juga menunjukkan keseimbangan yang baik antara *precision* dan *recall*.

Tingginya performa model dipengaruhi oleh penggunaan representasi *Abstract Syntax Tree* (AST) yang mampu menangkap struktur kode secara lebih mendalam, serta pemanfaatan *Graph Neural Network* (GNN) yang efektif dalam mempelajari hubungan antar elemen dalam *graph*. Selain itu, penggunaan dataset yang beragam juga membantu model dalam mengenali berbagai pola plagiarisme.

Namun demikian, model masih memiliki keterbatasan, di mana pada beberapa kasus model cenderung memberikan nilai kemiripan yang tinggi pada kode dengan struktur serupa meskipun secara logika berbeda. Hal ini menunjukkan bahwa model lebih berfokus pada kesamaan struktur dibandingkan pemahaman semantik secara mendalam.

Secara keseluruhan, model yang dikembangkan telah menunjukkan performa yang sangat baik dan memiliki kemampuan yang baik dalam mendeteksi plagiarisme kode program, meskipun masih terdapat ruang untuk pengembangan lebih lanjut.

4.3 Hasil Pengujian dan Evaluasi

Setelah model selesai dilatih, tahap selanjutnya adalah melakukan pengujian untuk mengevaluasi kinerja model dalam mendeteksi plagiarisme kode program. Pengujian dilakukan menggunakan data pengujian (*testing data*) yang tidak digunakan selama proses pelatihan, sehingga hasil yang diperoleh dapat mencerminkan kemampuan model dalam melakukan generalisasi terhadap data baru.

Pada tahap ini, model mengolah pasangan kode program dan menghasilkan nilai kemiripan (*similarity score*) yang digunakan untuk menentukan kategori plagiarisme. Hasil pengujian kemudian dianalisis untuk melihat sejauh mana model mampu membedakan antara kode plagiarisme dan non-plagiarisme secara akurat.

Selain itu, evaluasi juga dilakukan dengan mengamati hasil prediksi model pada berbagai kondisi, sehingga dapat diketahui perilaku model dalam menghadapi variasi kasus plagiarisme. Dengan demikian, tahap ini tidak hanya menunjukkan performa model, tetapi juga memberikan gambaran nyata mengenai cara kerja model dalam proses deteksi.

4.3.1 Hasil Pengujian Model pada Dataset

Hasil pengujian model diperoleh dari evaluasi terhadap data pengujian (*testing data*) yang tidak digunakan selama pelatihan. Pada tahap ini, setiap pasangan kode program diproses oleh model untuk menghasilkan nilai *similarity score*, yang kemudian digunakan untuk menentukan prediksi plagiarisme.

Secara umum, model *Graph Neural Network* (GNN) mampu memberikan nilai kemiripan yang tinggi pada pasangan kode plagiarisme dan nilai yang sangat rendah pada pasangan kode non-plagiarisme. Hal ini menunjukkan bahwa model memiliki kemampuan yang baik dalam membedakan kedua kelas berdasarkan nilai kemiripan yang dihasilkan.

Tabel 4. 4 Contoh Hasil Pengujian Model.

No	Label Asli	GNN Score	Baseline Score	Pred GNN	Pred Baseline
1.	1	0.9912	1.0000	1	1
2.	1	0.9934	0.7848	1	1
3.	1	0.9879	0.3002	1	0
4.	1	0.9795	0.2330	1	0
5.	1	0.9899	0.2101	1	0
6.	1	0.9911	0.8876	1	1
7.	0	0.0000	0.1133	0	0
8.	0	0.0000	0.0213	0	0
9.	0	0.0002	0.1100	0	0
10.	0	0.0001	0.0102	0	0

Dapat dilihat pada Tabel 4.3, bahwa model GNN memberikan nilai *similarity score* yang tinggi pada data plagiarisme (umumnya di atas 0.97) dan nilai yang sangat rendah pada data non-plagiarisme. Selain itu, prediksi yang dihasilkan model juga konsisten dengan label asli, yang menunjukkan tingkat akurasi yang tinggi.

Jika dibandingkan dengan metode baseline, terlihat bahwa pendekatan berbasis teks cenderung menghasilkan nilai yang kurang konsisten, terutama pada beberapa kasus plagiarisme yang tidak berhasil terdeteksi. Hal ini menunjukkan bahwa pendekatan berbasis struktur menggunakan GNN lebih efektif dalam mendeteksi kemiripan kode program.

Dengan demikian, hasil pengujian ini menunjukkan bahwa model yang dikembangkan mampu bekerja secara efektif pada data pengujian serta memiliki kemampuan yang baik dalam membedakan plagiarisme dan non-plagiarisme.

4.3.2 Analisis Hasil Berdasarkan Kasus

Analisis hasil pengujian dilakukan untuk memahami perilaku model dalam mendeteksi plagiarisme kode program pada berbagai kondisi. Pengujian mencakup beberapa kategori kasus, yaitu plagiarisme tinggi, sedang, rendah, serta kasus yang menunjukkan keterbatasan model.

Pada kasus plagiarisme tinggi (*high plagiarism*), model mampu memberikan nilai *similarity score* yang sangat tinggi dan konsisten. Hal ini disebabkan oleh kesamaan struktur kode yang signifikan, sehingga model dapat dengan mudah mengenali pola kemiripan yang ada.

Pada kasus plagiarisme sedang (*medium plagiarism*), model cenderung tetap memberikan nilai kemiripan yang tinggi meskipun terdapat perbedaan pada logika program. Hal ini menunjukkan bahwa model memiliki sensitivitas tinggi terhadap kesamaan struktur dasar, seperti penggunaan pola perulangan dan percabangan.

Sementara itu, pada kasus non-plagiarisme (*low plagiarism*), model mampu memberikan nilai kemiripan yang sangat rendah. Hal ini menunjukkan bahwa model memiliki kemampuan yang baik dalam membedakan kode program yang tidak memiliki kesamaan struktur.

Namun demikian, terdapat beberapa kasus yang menunjukkan keterbatasan model, yaitu ketika dua kode program memiliki struktur yang sangat mirip tetapi berbeda dalam logika atau operasi. Dalam kondisi ini, model tetap memberikan

nilai kemiripan yang tinggi karena pendekatan yang digunakan lebih berfokus pada kesamaan struktur dibandingkan pemahaman semantik secara mendalam.

Sebagai ilustrasi, contoh analisis hasil pengujian berdasarkan kategori kasus ditunjukkan pada Tabel 4.4.

Tabel 4. 5 Analisis Hasil Berdasarkan Kasus.

No	Kategori Kasus	Deskripsi	GNN Score	Interpretasi
1.	High Plagiarism	Struktur kode sangat mirip	0.99	Terdeteksi dengan sangat baik
2.	Medium Plagiarism	Struktur mirip, logika berbeda	0.97	Cenderung terdeteksi tinggi
3.	Low Plagiarism	Struktur berbeda	0.0001	Terdeteksi dengan baik
4.	Error Case	Struktur sama, operasi berbeda	0.98	Tidak membedakan semantik

Berdasarkan Tabel 4.4, terlihat bahwa model memiliki kemampuan yang sangat baik dalam mendeteksi plagiarisme berbasis struktur, namun masih memiliki keterbatasan dalam membedakan perbedaan semantik yang halus. Secara keseluruhan, model menunjukkan performa yang efektif dalam berbagai kondisi pengujian, meskipun masih terdapat ruang untuk pengembangan lebih lanjut.

4.4 Perbandingan dengan Metode Baseline

Pada penelitian ini, dilakukan perbandingan antara model yang diusulkan dengan metode baseline untuk mengevaluasi keunggulan pendekatan yang

digunakan. Metode baseline merupakan metode pembanding yang umumnya digunakan sebagai acuan dasar dalam suatu penelitian, sehingga dapat memberikan gambaran mengenai peningkatan kinerja yang dihasilkan oleh model yang dikembangkan.

Metode baseline yang digunakan dalam penelitian ini adalah pendekatan berbasis teks menggunakan *Term Frequency–Inverse Document Frequency* (TF-IDF) yang dikombinasikan dengan *cosine similarity*. Metode ini dipilih karena merupakan salah satu pendekatan sederhana yang umum digunakan dalam mengukur kemiripan dokumen, termasuk dalam konteks kode program.

Perbandingan antara metode *baseline* dan model *Graph Neural Network* (GNN) bertujuan untuk menunjukkan perbedaan kemampuan kedua pendekatan dalam mendeteksi plagiarisme kode program. Pendekatan *baseline* cenderung berfokus pada kesamaan teks, sedangkan model GNN memanfaatkan representasi struktur kode melalui *Abstract Syntax Tree* (AST), sehingga diharapkan mampu menangkap kemiripan yang lebih kompleks.

Dengan adanya perbandingan ini, dapat diketahui sejauh mana model yang diusulkan memberikan peningkatan kinerja dibandingkan metode konvensional, serta memperkuat validitas hasil penelitian yang dilakukan.

4.4.1 Metode Baseline (TF-IDF dan Cosine Similarity)

Metode baseline yang digunakan dalam penelitian ini adalah pendekatan berbasis teks menggunakan *Term Frequency–Inverse Document Frequency* (TF-IDF) yang dikombinasikan dengan *cosine similarity*. Metode ini digunakan sebagai pembanding karena memiliki pendekatan yang sederhana dan umum digunakan dalam mengukur tingkat kemiripan antar dokumen.

TF-IDF merupakan teknik representasi teks yang mengubah dokumen menjadi vektor numerik berdasarkan frekuensi kemunculan kata. Komponen *term frequency* (TF) mengukur seberapa sering suatu kata muncul dalam dokumen, sedangkan *inverse document frequency* (IDF) memberikan bobot lebih tinggi pada kata yang jarang muncul di seluruh dokumen. Dengan demikian, TF-IDF mampu merepresentasikan karakteristik teks secara kuantitatif.

Setelah kode program direpresentasikan dalam bentuk vektor TF-IDF, tingkat kemiripan antara dua kode dihitung menggunakan *cosine similarity*. *Cosine similarity* mengukur sudut antara dua vektor dalam ruang vektor, dengan nilai berkisar antara 0 hingga 1. Nilai yang mendekati 1 menunjukkan tingkat kemiripan yang tinggi, sedangkan nilai mendekati 0 menunjukkan perbedaan yang signifikan.

Dalam konteks penelitian ini, kode program diperlakukan sebagai teks, sehingga pendekatan TF-IDF hanya mempertimbangkan kemunculan token atau kata tanpa memperhatikan struktur kode. Hal ini menyebabkan metode *baseline* memiliki keterbatasan dalam mendeteksi plagiarisme yang melibatkan perubahan struktur, seperti penggantian nama variabel atau modifikasi alur program.

Meskipun demikian, metode ini tetap digunakan sebagai pembanding untuk menunjukkan perbedaan performa antara pendekatan berbasis teks dan pendekatan berbasis struktur yang digunakan oleh model *Graph Neural Network* (GNN).

4.4.2 Perbandingan Hasil Kinerja Model

Perbandingan kinerja dilakukan antara model *Graph Neural Network* (GNN) yang diusulkan dengan metode baseline berbasis TF-IDF dan *cosine similarity*. Evaluasi dilakukan menggunakan data pengujian dengan mengukur beberapa metrik, yaitu *accuracy*, *precision*, *recall*, dan *F1-score*.

Hasil perbandingan kinerja kedua metode dapat dilihat pada Tabel 4.5.

Tabel 4. 6 Perbandingan Kinerja Model.

Metode	Accuracy	Precision	Recall	F1-Score
GNN	0.9946	0.9949	0.9974	0.9962
TF-IDF ditambah Cosine Similarity	0.7392	0.9940	0.6343	0.7744

Berdasarkan Tabel 4.5, terlihat bahwa model GNN memiliki performa yang jauh lebih baik dibandingkan metode baseline pada hampir seluruh metrik evaluasi. Nilai *accuracy* pada model GNN mencapai 0.9946, yang menunjukkan tingkat ketepatan klasifikasi yang sangat tinggi, sedangkan metode baseline hanya mencapai 0.7392.

Perbedaan yang paling signifikan terlihat pada nilai *recall*, di mana model GNN mencapai nilai 0.9974, sementara metode baseline hanya sebesar 0.6343. Hal ini menunjukkan bahwa metode baseline gagal mendeteksi sejumlah kasus plagiarisme, sedangkan model GNN mampu mengidentifikasi hampir seluruh kasus yang ada.

Selain itu, nilai *F1-score* pada model GNN juga jauh lebih tinggi dibandingkan baseline, yang menunjukkan bahwa model memiliki keseimbangan yang baik antara *precision* dan *recall*. Meskipun nilai *precision* pada kedua metode relatif tinggi, metode baseline memiliki kelemahan dalam menangkap seluruh kasus plagiarisme.

Dengan demikian, hasil perbandingan ini menunjukkan bahwa pendekatan berbasis struktur menggunakan GNN lebih efektif dibandingkan pendekatan berbasis teks dalam mendeteksi plagiarisme kode program.

4.4.3 Analisis Perbandingan

Berdasarkan hasil perbandingan kinerja pada subbab sebelumnya, terlihat bahwa model *Graph Neural Network* (GNN) memiliki performa yang lebih unggul dibandingkan metode baseline berbasis TF-IDF dan *cosine similarity*. Perbedaan ini terutama disebabkan oleh perbedaan pendekatan yang digunakan dalam merepresentasikan kode program.

Metode baseline berbasis TF-IDF hanya memandang kode program sebagai kumpulan teks, sehingga kemiripan diukur berdasarkan kesamaan token atau kata yang muncul. Pendekatan ini tidak mempertimbangkan struktur kode, sehingga rentan terhadap perubahan sederhana seperti penggantian nama variabel, perubahan urutan kode, atau modifikasi sintaks. Akibatnya, metode baseline cenderung gagal mendeteksi plagiarisme yang melibatkan perubahan tersebut, yang tercermin dari rendahnya nilai *recall*.

Sebaliknya, model GNN memanfaatkan representasi *Abstract Syntax Tree* (AST) yang mengubah kode program menjadi struktur graph. Pendekatan ini memungkinkan model untuk memahami hubungan antar elemen dalam kode, sehingga mampu menangkap pola kesamaan yang lebih kompleks. Dengan demikian, meskipun terjadi perubahan pada nama variabel atau bentuk penulisan kode, model tetap dapat mengenali kesamaan struktur program.

Selain itu, kemampuan GNN dalam melakukan *message passing* pada graph memungkinkan model untuk mengekstraksi informasi dari konteks yang lebih luas, sehingga menghasilkan representasi yang lebih kaya dibandingkan pendekatan berbasis teks. Hal ini berkontribusi pada tingginya nilai *recall* dan *F1-score* yang diperoleh.

Dengan demikian, dapat disimpulkan bahwa pendekatan berbasis struktur menggunakan GNN lebih efektif dalam mendeteksi plagiarisme kode program dibandingkan metode berbasis teks. Hasil ini menunjukkan bahwa pemanfaatan struktur kode merupakan faktor penting dalam meningkatkan akurasi deteksi plagiarisme.

4.5 Analisis Hasil

Berdasarkan hasil pengujian dan perbandingan yang telah dilakukan, model *Graph Neural Network* (GNN) menunjukkan performa yang sangat baik dalam mendeteksi plagiarisme kode program. Hal ini ditunjukkan oleh nilai metrik evaluasi yang tinggi serta kemampuan model dalam membedakan berbagai tingkat kemiripan kode.

Pada kasus plagiarisme tinggi (*high plagiarism*), model mampu mendeteksi kemiripan dengan sangat baik, ditunjukkan oleh nilai *similarity score* yang tinggi dan konsisten. Hal ini menunjukkan bahwa model efektif dalam mengenali kesamaan struktur kode yang signifikan.

Pada kasus plagiarisme sedang (*medium plagiarism*), model cenderung memberikan nilai kemiripan yang tetap tinggi meskipun terdapat perbedaan pada logika program. Kondisi ini menunjukkan bahwa model memiliki sensitivitas tinggi terhadap kesamaan struktur, namun belum sepenuhnya mampu membedakan perbedaan semantik secara mendalam.

Sementara itu, pada kasus non-plagiarisme (*low plagiarism*), model mampu memberikan nilai kemiripan yang sangat rendah, sehingga menunjukkan kemampuan yang baik dalam menghindari kesalahan deteksi pada kode yang tidak memiliki kesamaan struktur.

Namun demikian, terdapat beberapa keterbatasan yang ditemukan pada model. Pada beberapa kasus, model menghasilkan prediksi yang kurang tepat, seperti *false positive* dan *false negative*. *False positive* terjadi ketika kode yang tidak memiliki plagiarisme terdeteksi sebagai plagiarisme, sedangkan *false negative* terjadi ketika kasus plagiarisme tidak berhasil terdeteksi oleh model.

Keterbatasan ini umumnya terjadi pada kasus di mana dua kode program memiliki struktur yang mirip tetapi berbeda dalam logika atau tujuan program. Hal ini menunjukkan bahwa model lebih berfokus pada kesamaan struktur dibandingkan pemahaman semantik secara menyeluruh.

Secara keseluruhan, hasil penelitian ini menunjukkan bahwa pendekatan berbasis struktur menggunakan GNN sangat efektif dalam mendeteksi plagiarisme kode program. Namun, untuk meningkatkan performa lebih lanjut, diperlukan pengembangan yang mampu mengintegrasikan pemahaman semantik kode program agar model dapat membedakan kesamaan yang bersifat superficial dan yang benar-benar menunjukkan plagiarisme.

4.6 Keterbatasan Sistem

Meskipun model Graph Neural Network (GNN) yang dikembangkan menunjukkan performa yang sangat baik, terdapat beberapa keterbatasan yang perlu diperhatikan.

Pertama, model belum mampu memahami aspek semantik kode secara mendalam dan lebih berfokus pada kesamaan struktur yang direpresentasikan melalui *Abstract Syntax Tree* (AST). Akibatnya, kode dengan struktur serupa tetapi logika berbeda masih dapat terdeteksi sebagai plagiarisme.

Kedua, model memiliki sensitivitas tinggi terhadap struktur kode, sehingga kurang optimal dalam membedakan variasi logika yang kompleks.

Ketiga, dataset yang digunakan masih terbatas dan tidak seimbang, yang berpotensi mempengaruhi kemampuan generalisasi model.

Keempat, penelitian ini hanya difokuskan pada bahasa pemrograman Python, sehingga hasilnya belum tentu dapat digeneralisasikan ke bahasa lain.

Dengan demikian, pengembangan selanjutnya diharapkan dapat meningkatkan pemahaman semantik, memperluas dan menyeimbangkan dataset, serta menguji model pada berbagai bahasa pemrograman.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil penelitian yang telah dilakukan, dapat disimpulkan bahwa model deteksi plagiarisme kode program berbasis *Abstract Syntax Tree* (AST) dan *Graph Neural Network* (GNN) berhasil dikembangkan dan diimplementasikan dalam skema *pairwise* (kode A vs kode B). Sistem yang dibangun mampu menghasilkan skor kemiripan serta klasifikasi plagiarisme secara otomatis dan objektif.

Model GNN yang digunakan menunjukkan performa yang sangat baik dalam mendeteksi plagiarisme kode program, khususnya pada kasus Type-1 (*copy-paste* kosmetik) dan Type-2 (*rename identifier*). Hal ini dibuktikan dengan nilai akurasi, *precision*, *recall*, dan *F1-score* yang tinggi pada data pengujian, yang menunjukkan bahwa model mampu mengenali pola kemiripan struktur kode secara efektif.

Dibandingkan dengan metode *baseline* berbasis TF-IDF dan *cosine similarity*, model GNN terbukti memiliki kinerja yang lebih unggul. Perbedaan yang paling signifikan terlihat pada kemampuan model dalam mendeteksi kasus plagiarisme yang melibatkan perubahan permukaan kode, seperti penggantian nama variabel. Hal ini menunjukkan bahwa pendekatan berbasis struktur menggunakan AST dan GNN lebih efektif dibandingkan pendekatan berbasis teks.

Selain itu, model juga menunjukkan kemampuan yang baik dalam membedakan kode plagiarisme dan non-plagiarisme berdasarkan struktur program. Namun, pada beberapa kasus, model masih mengalami keterbatasan dalam

membedakan kode dengan struktur yang mirip tetapi memiliki perbedaan logika, yang menunjukkan bahwa pemahaman semantik kode belum sepenuhnya optimal.

Secara keseluruhan, penelitian ini membuktikan bahwa pemanfaatan representasi AST yang dikombinasikan dengan Graph Neural Network (GNN) merupakan pendekatan yang efektif dalam mendeteksi plagiarisme kode program. Pendekatan ini mampu mengatasi keterbatasan metode konvensional berbasis teks serta memberikan hasil yang lebih akurat dan robust dalam berbagai kondisi pengujian.

5.2 Saran

Berdasarkan hasil penelitian yang telah dilakukan, terdapat beberapa saran yang dapat dipertimbangkan untuk pengembangan penelitian selanjutnya.

Pertama, pengembangan model perlu diarahkan pada peningkatan kemampuan dalam memahami aspek semantik kode program. Hal ini penting untuk mengatasi keterbatasan model yang masih cenderung berfokus pada kesamaan struktur, sehingga diharapkan mampu membedakan kode dengan struktur serupa tetapi memiliki logika yang berbeda.

Kedua, diperlukan penggunaan dataset yang lebih besar, beragam, dan seimbang untuk meningkatkan kemampuan generalisasi model. Selain itu, penambahan variasi jenis plagiarisme yang lebih kompleks, seperti Type-3 dan Type-4, dapat memberikan evaluasi yang lebih komprehensif terhadap performa sistem dalam kondisi yang lebih mendekati dunia nyata.

Ketiga, penelitian selanjutnya disarankan untuk menguji dan mengembangkan model pada berbagai bahasa pemrograman selain Python. Hal ini bertujuan untuk mengetahui kemampuan adaptasi model terhadap perbedaan

sintaks dan struktur pada bahasa pemrograman lain, sehingga meningkatkan fleksibilitas dan cakupan penggunaan sistem.

Keempat, pengembangan sistem dapat diarahkan pada integrasi dengan platform pembelajaran atau sistem manajemen pembelajaran (*Learning Management System*), sehingga dapat dimanfaatkan secara langsung sebagai alat bantu dalam proses evaluasi tugas pemrograman secara otomatis dan objektif.

Kelima, eksplorasi pendekatan yang mengombinasikan representasi struktur dan semantik, seperti model berbasis embedding atau pendekatan hibrida, dapat menjadi alternatif untuk meningkatkan akurasi dan robustitas sistem dalam mendeteksi plagiarisme kode program.

DAFTAR PUSTAKA

Büyük, O. O., & Nizam, A. (2023). Deep learning with class-level abstract syntax tree and code histories for detecting code modification requirements. *Journal of Systems and Software*, *206*, 111851. https://doi.org/10.1016/j.jss.2023.111851

Chen, C., Wu, Y., Dai, Q., Zhou, H. Y., Xu, M., Yang, S., Han, X., & Yu, Y. (2024). A Survey on Graph Neural Networks and Graph Transformers in Computer Vision: A Task-Oriented Perspective. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *46*(12), 10297–10318. https://doi.org/10.1109/TPAMI.2024.3445463

Dong, Z., Hu, Q., Zhang, Z., & Zhao, J. (2024). On the effectiveness of graph data augmentation for source code learning. *Knowledge-Based Systems*, *285*(October 2023), 111328. https://doi.org/10.1016/j.knosys.2023.111328

Ebrahim, F., & Joy, M. (2023). Source Code Plagiarism Detection with Pre-Trained Model Embeddings and Automated Machine Learning. *International Conference Recent Advances in Natural Language Processing, RANLP*, 301–309. https://doi.org/10.26615/978-954-452-092-2_034

Ebrahim, F., & Joy, M. (2024). Semantic Similarity Search for Source Code Plagiarism Detection: An Exploratory Study. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, *1*, 360–366. https://doi.org/10.1145/3649217.3653622

Eka Putra, I. G. A., & Supriana, I. W. (2022). Deteksi Plagiarisme Source Code Tugas Mahasiswa Menggunakan Algoritma Cosine Similarity Dan Pembobotan TF-IDF. *Jurnal Nasional Teknologi Informasi Dan Aplikasnya*, *1*(1), 575. https://ojs.unud.ac.id/index.php/jnatia/article/view/92871

Guo, J., Liu, J., Liu, X., Wan, Y., & Li, L. (2023). Summarizing source code with Heterogeneous Syntax Graph and dual position. *Information Processing and Management*, *60*(5). https://doi.org/10.1016/j.ipm.2023.103415

Koschke, R. (2007). Survey of Research on Software Clones. *Dagstuhl Seminar Proceedings*, *6301*.

Li, C., Sirikham, A., Konpang, J., & Wang, Y. (2024). Code Clone Detection With Self-Supervision on Dual Graphs. *Operational Research in Engineering Sciences: Theory and Applications*, *7*(4), 105–129. https://doi.org/10.5281/zenodo.15786583

Li, Z., Lei, H., Ma, Z., & Zhang, F. (2024). Code Similarity Prediction Model for Industrial Management Features Based on Graph Neural Networks. *Entropy*, *26*(6). https://doi.org/10.3390/e26060505

Maertens, R., Van Neyghem, M., Geldhof, M., Van Petegem, C., Strijbol, N.,

Dawyndt, P., & Mesuere, B. (2024). Discovering and exploring cases of educational source code plagiarism with Dolos. *SoftwareX*, *26*(February), 101755. https://doi.org/10.1016/j.softx.2024.101755

Roy, C. K., Cordy, J. R., & Koschke, R. (2009). Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, *74*(7), 470–495. https://doi.org/10.1016/j.scico.2009.02.007

Siddiqui, N. (2025). Real-Time Code Plagiarism Detection Using NLP and Machine Learning for Academic and Industry Applications. *International Research Journal of Engineering and Technology*, 576–585. www.irjet.net

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(1), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386

Yang, G., Jin, T., & Dou, L. (2023). Heterogeneous Directed Hypergraph Neural Network over abstract syntax tree (AST) for Code Classification. *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, *2023-July*, 274–279. https://doi.org/10.18293/SEKE2023-136

Yang, H., Li, Z., & Guo, X. (2023). A Novel Source Code Clone Detection Method Based on Dual-GCN and IVHFS. *Electronics (Switzerland)*, *12*(6). https://doi.org/10.3390/electronics12061315

Yu, D., Yang, Q., Chen, X., Chen, J., & Xu, Y. (2023). Graph-based code semantics learning for efficient semantic code clone detection. *Information and Software Technology*, *156*(December 2022), 107130. https://doi.org/10.1016/j.infsof.2022.107130

Yu, Q., Liu, X., Zhou, Q., Zhuge, J., & Wu, C. (2023). Code classification with graph neural networks: Have you ever struggled to make it work? *Expert Systems with Applications*, *233*(July), 120978. https://doi.org/10.1016/j.eswa.2023.120978

Zakeri-Nasrabadi, M., Parsa, S., Ramezani, M., Roy, C., & Ekhtiarzadeh, M. (2023). A systematic literature review on source code similarity measurement and clone detection: Techniques, applications, and challenges. *Journal of Systems and Software*, *204*, 111796. https://doi.org/10.1016/j.jss.2023.111796

Zhang, Y., Yang, J., & Ruan, O. (2024). Cross-language Source Code Clone Detection Based on Graph Neural Network. *ACM International Conference Proceeding Series*, *February*, 189–194. https://doi.org/10.1145/3673277.3673310

Zhang, Z., & Saber, T. (2025). Exploring the Boundaries Between LLM Code Clone Detection and Code Similarity Assessment on Human and AI-Generated Code. *Big Data and Cognitive Computing*, *9*(2), 1–19. https://doi.org/10.3390/bdcc9020041

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*, *1*(December 2020), 57–81. https://doi.org/10.1016/j.aiopen.2021.01.001