Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) p-ISSN: 2355-7699
Vol. 12, No. 5, Oktober 2025, hlm. 1037-1046 e-ISSN: 2528-6579

MENINGKATKAN DATASET CODEXGLUE DENGAN REPRESENTASI *ABSTRACT SYNTAX TREE* (AST) TER SERAGAM UNTUK ANALISIS KODE LINTAS BAHASA

Mardi Siswo Utomo[*1] , Ema Utami[2] , Kusrini [3] , Arief Setyanto[4]

[1] Universitas Stikubank Semarang, Semarang, [2,3,4] Universitas Amikom Yogyakarta, Yogyakarta
Email: [1] mardi@edu.unisbank.ac.id, [2] ema.u@amikom.ac.id, [3] kusrini@amikom.ac.id, [4] arief_s @amikom.ac.id
*Penulis Korespondensi

(Naskah masuk: 10 April 2025, diterima untuk diterbitkan: 30 Oktober 2025)

Abstrak

Dataset kode sumber populer seperti CodeXGLUE belum menyediakan representasi sintaksis yang diseragamkan untuk penelitian lintas bahasa pemrograman. Hal ini akan menyulitkan saat dilakukan penelitian yang berkaitan dengan analisis *syntax-aware*. Penelitian ini menyediakan representasi sintaksis yang diseragamkan untuk memperkaya dataset CodeXGLUE. Kami menghadirkan dataset CodeXGLUE-AST (*Abstract Syntax Tree)* seragam untuk enam bahasa pemrograman: Go, Java, JavaScript, Python, Ruby, dan PHP. AST diekstraksi menggunakan Tree-sitter dan disimpan dalam format JSON terstruktur. Untuk menjaga konsistensi antar bahasa, kemudian dilakukan klasifikasi dan pemetaan tipe node guna menyatukan representasi struktur AST. Evaluasi dataset menggunakan analisis kelengkapan struktur AST, pengukuran akurasi rekonstruksi kode menggunakan skor BLEU, serta pengujian ekstraksi *Data Flow Graph* (DFG) untuk menjaga ketergantungan antar variabel. Selain itu juga dilakukan pengujian pada tugas peringkasan kode menggunakan model CodeT5 yang menunjukkan peningkatan nilai BLEU, METEOR, ROUGE dan ROUGE-L hampir disemua percobaan saat menggunakan AST yang diseragamkan. Dengan representasi AST yang telah diseragamkan, diharapkan pengembangan model ML multi bahasa yang lebih andal dan sadar sintaksis untuk tugas-tugas seperti klasifikasi kode, pembuatan ringkasan kode, dan rekonstruksi program akan menjadi lebih berkembang.

Kata kunci: *abstract syntax tree, peringkasan kode, pembelajaran mesin, generasi kode, analisis sintaksis*

ENHANCING THE CODEXGLUE DATASET WITH UNIFORM ABSTRACT SYNTAX TREE (AST) REPRESENTATION FOR CROSS-LANGUAGE CODE ANALYSIS

Abstract

Popular source code datasets like CodeXGLUE have not yet provided a standardized syntactic representation for cross-programming language research. This data gap will complicate research related to syntax-aware analysis. This research provides a standardized syntactic representation to enrich the CodeXGLUE dataset. We present a uniform CodeXGLUE-AST (Abstract Syntax Tree) dataset for six programming languages: Go, Java, JavaScript, Python, Ruby, and PHP. The AST is extracted using Tree-sitter and stored in a structured JSON format. To maintain consistency across languages, classification and mapping of node types were then performed to unify the AST structure representation. The dataset evaluation used AST structure completeness analysis, code reconstruction accuracy measurement using BLEU scores, and Data Flow Graph (DFG) extraction testing to maintain variable dependencies. Additionally, testing was conducted on the code summarization task using the CodeT5 model, which showed an increase in BLEU, METEOR, ROUGE, and ROUGE-L scores in almost all experiments when using the standardized AST. With the standardized AST representation, it is hoped that the development of more reliable and syntax-aware multilingual ML models for tasks such as code classification, code summarization, and program reconstruction will become more advanced.

Keywords: *abstract syntax tree, code summarization, machine learning, code generation, syntax analysis.*

1. PENDAHULUAN

Ada kebutuhan mendesak alat analisis kode yang efektif dan otomatis karena semakin bertambah kompleksnya kode sumber perangkat lunak sekarang ini (Dehaerne et al., 2022). Kecerdasan buatan (AI) dan pembelajaran mesin (ML) telah membuka peluang baru dalam otomatisasi berbagai tugas rekayasa perangkat lunak (Frankish and Ramsey, 2014; Dehaerne et al., 2022; Siswo Utomo et al., 2024).

Salah satu metode untuk merepresentasikan kode sumber adalah *Abstract Syntax Tree* (AST)

(Shah, Patel and Terry, 2023; Alikhanifard and Tsantalis, 2024; Takerngsaksiri, Tantithamthavorn and Li, 2024), yang merupakan representasi hierarkis dari struktur sintaksis suatu program. AST memungkinkan pemahaman yang lebih mendalam tentang struktur kode dibandingkan dengan representasi berbasis token saja, dikarenakan struktur AST dapat mempertahankan hubungan antara elemen-elemen sintaksis (Cha, Taylor and Kang, 2019). Struktur AST banyak digunakan dalam berbagai aplikasi, seperti evaluasi kode (Duracik et al., 2020; Alikhanifard and Tsantalis, 2024), ringkasan kode (Sun et al., 2024),(Chen, Hu and Liu, 2019), (Kuang, Zhou and Yang, 2022), dan deteksi kerentanan keamanan perangkat lunak (Zhang et al., 2023).

CodeXGLUE (Lu et al., 2021a) (*Code Execution, Language Understanding, and Generation Benchmark*) adalah kumpulan data *benchmark* yang dirancang untuk berbagai tugas di bidang pemrosesan bahasa pemrograman. Dataset ini dikembangkan oleh Microsoft *Research* dan menyediakan berbagai dataset yang bertujuan untuk mengukur kinerja model dalam pemrosesan bahasa pemrograman. Dataset ini mencakup berbagai tugas, termasuk ringkasan kode, terjemahan kode, penyelesaian kode, dan tugas terkait lainnya yang berhubungan dengan kode program.

Penelitian-penelitian sebelumnya yang menggunakan AST dari dataset CodeXGLUE berfokus pada representasi AST berbasis token yang belum diseragamkan, penelitian kami memperkenalkan dataset AST yang menyeragamkan istilah dan struktur sintaksis di berbagai bahasa pemrograman. Tidak seperti metode ekstraksi AST sebelumnya yang bervariasi berdasarkan parser, pendekatan kami memastikan representasi AST lintas bahasa yang konsisten secara struktural dan bermakna secara semantik (Duracik et al., 2020; Shi et al., 2021; Tipirneni, Zhu and Reddy, 2024). Penelitian GraphCodeBERT (Guo et al., 2021) memanfaatkan *Data Flow Graphs* (DFG) dari struktur AST akan tetapi tidak memiliki representasi AST yang ter-seragamkan di berbagai bahasa. Demikian pula StructCoder dan dataset lainnya yang telah diperkenalkan sebelumnya, mereka semua tidak menggunakan AST yang ter-seragam.

Kami akan memperkaya dataset CodeXGLUE dengan informasi yang sadar sintaksis, memungkinkan analisis kode terstruktur dan berfungsi sebagai dataset pertama yang menyediakan data AST secara konsisten di enam bahasa pemrograman (Cha, Taylor and Kang, 2019).

Untuk mengatasi kesenjangan ini, kami melakukan penelitian ini dengan tujuan menyeragamkan struktur AST untuk enam bahasa pemrograman, yaitu: Go, Java, JavaScript, Python, Ruby, dan PHP. AST yang telah diseragamkan diharapkan dapat mengurangi bias dari teknik ekstraksi yang tidak konsisten sebelumnya (Gao et al., 2023; Shah, Patel and Terry, 2023; Tipirneni, Zhu and Reddy, 2024) memungkinkan evaluasi model yang lebih adil, perbandingan yang konsisten, dan reprodusibilitas penelitian yang lebih baik.

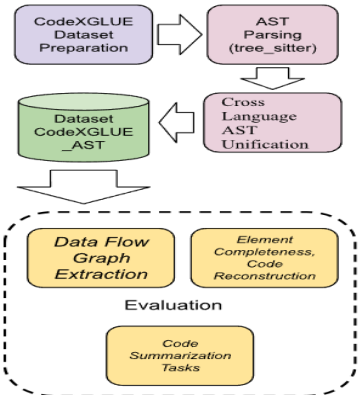
Penelitian kami berangkat dari penelitian sebelumnya yaitu tentang representasi AST terpadu (Wang et al., 2022), akan tetapi pendekatan yang kami lakukan berbeda dan kami fokus pada penyeragaman AST untuk ringkasan kode yang sadar sintaksis dan rekonstruksi AST-ke-kode daripada klasifikasi program. Kami menggunakan Tree-sitter (Anon., 2019) untuk mengekstrak AST kemudian mengkonversikannya ke format JSON. Dengan format JSON ini, elemen-elemen penting seperti tipe node, variabel, tipe data, ekspresi, dan struktur kontrol dapat terjaga dengan baik. Selanjutnya AST akan dianalisa kelengkapannya dan dilakukan juga rekonstruksi kode berbasis AST untuk memvalidasinya. Kami menggunakan skor BLEU, bersama dengan ekstraksi DFG (Guo et al., 2021) untuk memverifikasi ketergantungan variabel dan analisis *Root-Leaf Path* untuk memastikan integritas struktural.

Selain itu kami juga melakukan pengujian pada tugas rangkuman kode (*Code Summarization*) pada dataset yang dihasilkan. Kami membandingkan nilai BLUE, METEROR, ROUGE dan ROUGE-L pada performa model CodeT5 (Wang et al., 2023) yang telah di *fine tuning* dengan AST asli, AST ter-seragam dan kode sumber.

Penelitian ini berkontribusi terhadap analisis kode berbasis AI dengan menyediakan dataset AST yang diseragamkan untuk analisa kode program multi-bahasa. Sebelumnya tidak ada penelitian tentang penyeragaman AST lintas bahasa dalam bentuk JSON terstruktur untuk enam bahasa pemrograman. Hasil penelitian ini diharapkan dapat meningkatkan berbagai tugas analisis kode, menetapkan tolok ukur baru untuk model AI dalam pemahaman kode sintaksis. Selain itu, dataset ini memperkuat ketahanan model ringkasan kode terhadap serangan *adversarial* dengan meningkatkan pemahaman model transformer terhadap struktur yang sadar sintaksis.

2. METODE PENELITIAN

Metodologi penelitian ditunjukkan pada gambar 1, dimulai dengan persiapan dataset CodeXGLUE, yang mencakup enam bahasa pemrograman. Setelah pra-pemrosesan, Tree-sitter digunakan untuk ekstraksi AST hasilnya disimpan dalam format JSON. Untuk menguji kualitas AST yang diseragamkan, dilakukan empat penilaian utama: kelengkapan AST, rekonstruksi kode (skor BLEU) (Papineni et al., 2002a; Evtikhiev et al., 2023), ekstraksi DFG untuk memverifikasi ketergantungan variabel, dan pengujian dataset pada tugas peringkasan kode pada enam bahasa pemrograman tersebut.



Gambar 1 Metode penelitian penyeragaman AST lintas bahasa

2.1. Persiapan Dataset

Penelitian ini menggunakan dataset CodeXGLUE. Dataset ini dapat digunakan untuk tugas-tugas pemrosesan kode, seperti klasifikasi, ringkasan, dan rekonstruksi kode. Langkah pertama unduh dataset CodeXGLUE dan kemudian mengekstraknya untuk enam bahasa pemrograman hingga siap digunakan. Hasil akhir dari persiapan dataset akan menghasilkan tiga file jsonl, yaitu train.jsonl, valid.jsonl, dan test.jsonl untuk setiap bahasa pemrograman. Setiap file yang tidak terkompresi terdiri dari baris data yang setiap barisnya mewakili satu fungsi.

2.2. Parsing AST

AST diparsing / ekstrak menggunakan Tree-sitter, yang memungkinkan untuk dilakukan pemrosesan multi-bahasa yang seragam. Proses ini melibatkan parsing kode sumber dengan tata bahasa Tree-sitter karena dukungan bahasanya yang luas, konsistensi parsing, dan efisiensinya (Latif et al., 2023). Tidak seperti ANTLR (Parr, 2014), yang tidak memiliki dukungan Ruby bawaan, Tree-sitter memastikan ekstraksi AST yang seragam di enam bahasa pemrograman yang digunakan tanpa memerlukan banyak parser.

Parsing Tree-sitter mempertahankan akurasi AST meskipun ada perubahan kode kecil, sementara tata bahasa yang didorong oleh komunitas ANTLR sering menghasilkan pohon parse yang tidak konsisten. Selain itu, Tree-sitter langsung menghasilkan AST dalam format JSON, yang menyederhanakan integrasi ke dalam pipeline pembelajaran mesin, sedangkan ANTLR menghasilkan pohon parse yang lebih kompleks yang memerlukan transformasi tambahan menjadi AST.

Gambar 2 menampilkan hasil ekstraksi AST dari kode program sederhana y = x + 2 * 3 yang ditulis dalam enam bahasa pemrograman. Setiap AST dihasilkan menggunakan Tree-sitter, yang mem-parsing sintaks setiap bahasa ke dalam bentuk pohon sintaks abstrak. Terlihat pada gambar 2 Struktur AST yang dihasilkan berbeda-beda tergantung pada grammar masing-masing bahasa pemrograman.

```
------------------------
go code:
y := x + z * 3
------------------------
Abstract Syntax Tree:
- source_file: y := x + z * 3
  - assignment: y := x + z * 3
    - expression_list: y
      - variable: y
    - :=: :=
    - expression_list: x + z * 3
      - expression: x + z * 3
        - variable: x
        - +: +
        - expression: z * 3
          - variable: z
          - *: *
          - int_literal: 3
  - ;
```

```
------------------------
python code:
y = x + z * 3
------------------------
Abstract Syntax Tree:
- module: y = x + z * 3
  - expression: y = x + z * 3
    - assignment: y = x + z * 3
      - variable: y
      - =: =
      - binary_operator: x + z * 3
        - variable: x
        - +: +
        - binary_operator: z * 3
          - variable: z
          - *: *
          - integer: 3
```

```
------------------------
php code:
$y = $x + $z * 3;
------------------------
Abstract Syntax Tree:
- program: $y = $x + $z * 3;
  - expression: $y = $x + $z * 3;
    - assignment: $y = $x + $z * 3
      - variable_name: $y
        - $: $
        - name: y
      - =: =
      - expression: $x + $z * 3
        - variable_name: $x
          - $: $
          - name: x
        - +: +
        - expression: $z * 3
          - variable_name: $z
            - $: $
            - name: z
          - *: *
          - integer: 3
    - ;: ;
```

```
------------------------
java code:
y = x + z * 3;
------------------------
Abstract Syntax Tree:
- program: y = x + z * 3;
  - expression: y = x + z * 3;
    - assignment: y = x + z * 3
      - variable: y
      - =: =
      - expression: x + z * 3
        - variable: x
        - +: +
        - expression: z * 3
          - variable: z
          - *: *
          - decimal_integer_literal: 3
    - ;: ;
```

```
------------------------
ruby code:
y = x + z * 3
------------------------
Abstract Syntax Tree:
- program: y = x + z * 3
  - assignment: y = x + z * 3
    - variable: y
    - =: =
    - binary: x + z * 3
      - variable: x
      - +: +
      - binary: z * 3
        - variable: z
        - *: *
        - integer: 3
```

```
------------------------
javascript code:
y = x + z * 3;
------------------------
Abstract Syntax Tree:
- program: y = x + z * 3;
  - expression: y = x + z * 3;
    - assignment: y = x + z * 3
      - variable: y
      - =: =
      - expression: x + z * 3
        - variable: x
        - +: +
        - expression: z * 3
          - variable: z
          - *: *
          - number: 3
    - ;: ;
```

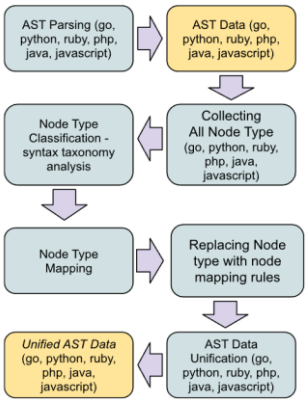
Gambar 2 Struktur AST sebelum diseragamkan

Perbedaan ini terlihat dalam beberapa aspek utama yaitu: 1) Setiap bahasa menggunakan penamaan node yang berbeda, 2) Struktur pohon berbeda. 3) Representasi Literal dan Variabel. Gambar 2 menegaskan perlunya proses penyeragaman AST lintas bahasa karena adanya beberapa perbedaan baik struktur maupun terminologi node dapat menghambat analisis kode lintas bahasa. Karena meskipun kode yang diuji identik, representasi AST yang dihasilkan bisa berbeda-beda.

2.3. Penyeragaman AST Lintas Bahasa

Penyeragaman AST Lintas Bahasa digunakan dalam studi ini untuk memastikan bahwa struktur AST dari berbagai bahasa pemrograman ditampilkan dengan cara yang sama. Penyeragaman AST diterapkan dengan menyeragamkan terminologi tipe node menjadi delapan kategori berdasarkan fungsi sintaksis (Sebesta, 2012), memastikan struktur AST yang seragam terlepas dari bahasa pemrograman.

Gambar 3 Metode penyeragaman AST lintas bahasa



Metode untuk menyeragamkan AST dari berbagai bahasa pemrograman ditunjukkan pada Gambar 3. Proses dimulai dengan parsing AST, di mana kode sumber dari enam bahasa diubah menjadi bentuk data AST. Setelah data AST dikumpulkan, proses pengumpulan semua jenis node dilakukan, yang melibatkan identifikasi semua jenis node yang muncul dalam berbagai bahasa. Untuk menyelaraskan jenis node, kami mengklasifikasikan jenis node ke dalam delapan kategori dengan metode analisis taksonomi sintaksis bahasa pemrograman (Sebesta, 2012).

Dengan analisis taksonomi juga, setiap tipe node akan dipetakan ke delapan kategori tadi. Setelah pemetaan selesai, tahap berikutnya adalah menyeragamkan terminologi tipe node menggunakan data pemetaan yang dihasilkan. Pada tahap berikutnya, semua data AST yang telah dimodifikasi digabungkan. Proses penggabungan ini menghasilkan data AST yang telah di seragamkan, yang siap untuk analisis lintas bahasa.

2.4. Validasi Grafik Aliran Data (DFG).

Validasi AST dengan DFG memastikan bahwa hubungan semantik antara variabel dan fungsi tetap utuh selama transformasi AST. Hal ini sangat penting untuk pelatihan *adversarial*, memungkinkan modifikasi sintaksis sambil mempertahankan ketergantungan data. DFG, yang direpresentasikan sebagai grafik terarah (G = (V, E)), menghubungkan node AST yang terkait dengan eksekusi data, di mana (V) mewakili variabel, konstanta, fungsi, atau ekspresi, dan (E) menunjukkan ketergantungan mereka. Konversi AST ke DFG diformulasikan seperti yang ditunjukkan dalam rumus 1.

$$DFG(G) = \{(V, E)|V = f(A), E = g(A)\} \quad (1)$$

Di mana:

$f(A)$ adalah fungsi yang mengekstrak node yang relevan dari AST A.

$g(A)$ adalah fungsi yang membangun hubungan ketergantungan antara node berdasarkan aliran data.

2.5. Rekonstruksi Kode

Kami memvalidasi rekonstruksi kode untuk memastikan AST yang diekstraksi secara akurat mewakili kode sumber. Proses ini menilai seberapa baik informasi sintaksis dipertahankan selama konversi dan rekonstruksi AST. Menggunakan penelusuran pre-order, setiap node AST diubah kembali menjadi urutan token yang mewakili kode asli. Kode yang direkonstruksi kemudian dibandingkan dengan yang asli menggunakan skor BLEU (Evtikhiev et al., 2023; Papineni et al., 2002a) yang ditunjukkan dalam Formula 2, dengan mengukur kesamaan tekstual. Karena AST adalah representasi terstruktur, skor BLEU ideal sebesar 100 mengonfirmasi bahwa transformasi kode → AST → kode mempertahankan integritas sintaksis penuh.

$$BLEU = BP \cdot \exp(\sum_{n=1}^{N} W_n \cdot ogp_n) \quad (2)$$

Di mana:

pn: Presisi n-gram, yaitu jumlah n-gram yang cocok antara hasil prediksi dan referensi dibagi dengan total jumlah n-gram yang ada dalam hasil prediksi.

wn: Bobot untuk setiap n-gram. Biasanya, wn = 1/N, yang berarti setiap n-gram diberikan bobot yang sama.

The Brevity Penalty (BP) menghukum prediksi yang terlalu pendek dibandingkan dengan referensi.

N adalah jumlah n-gram yang dipertimbangkan (biasanya hingga 4-gram, jadi N = 4).

3. HASIL DAN PEMBAHASAN

3.1. Persiapan Dataset

Dataset CodeXGLUE diunduh dan diproses sebelumnya mengikuti pedoman yang ditetapkan dalam laman resmi dataset CodeXGLUE (Lu et al., 2021a). Langkah pra-pemrosesan memastikan bahwa semua fungsi diparse dan diformat dengan

benar, menghilangkan fungsi yang tidak lengkap atau tidak dapat diparse untuk menjaga kualitas dataset.

Dataset akhir terdiri dari file JSONL untuk enam bahasa pemrograman. Proses persiapan dimulai dengan mengkloning *repository* resmi CodeXGLUE dari GitHub (Lu et al., 2021b) ke folder lokal. Unduhan dilakukan dengan mengunduh file zip dari GitHub resmi mereka dan mengekstraknya ke folder lokal.

3.2. Ekstraksi dan Unifikasi AST

AST diekstraksi menggunakan Tree-sitter (Anon., 2019), memastikan representasi JSON yang seragam di seluruh bahasa pemrograman. Seperti yang disoroti oleh studi-studi sebelumnya (Takerngsaksiri, Tantithamthavorn and Li, 2024; Tipirneni, Zhu and Reddy, 2024) struktur AST bervariasi secara signifikan antar bahasa karena strategi parsing yang berbeda. Metode penyeragaman AST Lintas Bahasa kami memastikan representasi yang seragam, mengurangi ketidak konsistenan yang ditemukan dalam penelitian - penelitian sebelumnya (Duracik et al., 2020; Wang et al., 2022).

Jumlah total tipe node di semua bahasa mencapai 596, dengan kategori yang memastikan pemetaan yang seragam, seperti yang ditunjukkan dalam Tabel 1. Setiap node dalam AST yang dihasilkan berisi informasi penting seperti tipe node, hubungan induk-anak, dan atribut sintaksis. Untuk memastikan analisis yang lebih konsisten di berbagai bahasa, proses normalisasi tipe node dilakukan, di mana nama-nama node yang berbeda dalam setiap bahasa dipetakan ke dalam struktur yang seragam.

Tabel 1 Jumlah Tipe Node Setiap Bahasa

Bahasa	Jumlah Tipe Node
Go	94
Python	96
Java	98
Javascript	108
Ruby	84
PHP	116
Total	596

Gambar 4 adalah *Word Cloud* untuk frekuensi kemunculan node dalam AST yang diekstrak dari enam bahasa pemrograman. Semakin besar ukuran kata dalam awan kata, semakin sering node tersebut muncul dalam dataset.

Gambar 4 Frekuensi Word Cloud untuk Tipe-tipe Node yang muncul

Dalam gambar 4, beberapa jenis node dominan terlihat jelas, seperti: *identifier* (terbesar), menunjukkan bahwa banyak elemen dalam AST merujuk pada nama variabel, fungsi, atau obyek dalam kode sumber, *expression statement* dan *binary expression*: menunjukkan jumlah operasi ekspresi dalam kode sumber.

Jenis node dalam AST yang dihasilkan dalam studi ini telah diklasifikasikan ke dalam delapan kategori untuk memastikan keseragaman dalam berbagai bahasa pemrograman. Tabel 2 berisi klasifikasi jenis node dalam AST. Klasifikasi menggunakan pendekatan analisis taksonomi sintaksis dengan menempatkan setiap jenis node ke dalam salah satu dari delapan kategori perintah.

Tabel 2 Klasifikasi / Ontologi tipe node

Kategori	Contoh tipe node
function_nodes	function, function_definition, method, class_declaration
variable_nodes	identifier, name
assignment_nodes	assignment, assignment_expression
expression_nodes	expression, expression_statement
call_nodes	call, call_expression
control_flow_nodes	if, for, while
type_nodes	int, string
ignore_nodes	call, call_expression

Dengan pemetaan ini, dipastikan bahwa dataset AST yang dihasilkan akan lebih seragam, mudah digunakan dalam model pembelajaran mesin, dan dapat digunakan untuk berbagai tugas analisis kode sintaksis dan semantik.

Setiap kategori mencakup tipe node yang memiliki peran serupa dalam berbagai bahasa pemrograman, memungkinkan representasi yang lebih seragam. Misalnya, semua jenis node yang terkait dengan deklarasi fungsi (seperti "*function*", "*function_definition*", "*method_declaration*") ditempatkan dalam kategori *function_nodes*, sementara node yang terkait dengan pemanggilan fungsi dikategorikan dalam *call_nodes*.

Pada Tabel 3 ditunjukkan distribusi jumlah node dalam setiap kategori dalam enam bahasa pemrograman dengan tipe node AST yang diseragamkan untuk memastikan representasi sintaksis seragam di seluruh bahasa pemrograman. Meskipun struktur AST bervariasi, banyak elemen sintaksis memiliki arti yang sama tetapi berbeda dalam penamaan.

Proses ini menyatukan istilah ke dalam kategori umum, memungkinkan analisis lintas bahasa. Misalnya, deklarasi fungsi (misalnya, `*function_definition*`, `*method_declaration*`, `*def*`, `*lambda*`) diseragamkan sebagai `*function*`. Sementara itu untuk pemanggilan fungsi (`*member_call_expression*``*method_invocation*`, `*call_expression*`) dipetakan ke `*call*`. Tabel 4 adalah contoh daftar pemetaan dari jenis-jenis node pada penelitian ini. Untuk peta lengkap tersedia di *repository* GitHub penelitian ini (Utomo et al., 2025) dengan menggunakan standar lisensi MIT.

3.3. Kelengkapan Informasi AST

Kelengkapan informasi AST sangat penting untuk mengevaluasi representasi sintaksis dalam analisis kode sumber. AST berkualitas tinggi harus menangkap semua elemen sintaksis kunci, termasuk struktur kendali, deklarasi variabel, pemanggilan fungsi, dan ekspresi.Memastikan setiap node AST diteliti. Tabel 3 juga menunjukkan distribusi jumlah node dalam setiap kategori pada enam bahasa pemrograman. Kolom terakhir adalah jumlah jenis node unik dalam setiap kategori.mempertahankan atribut lengkap menjaga integritas kode sumber selama ekstraksi dan transformasi.

Tabel 3. Distribusi jumlah node pada setiap kategori

Kategori	Go	Python	Ruby	PHP	Java	Javascript	Unique
function_nodes	5	7	4	5	3	7	24
variable_nodes	7	10	17	10	11	15	62
assignment_nodes	2	2	2	3	1	2	8
expression_nodes	24	26	29	29	27	29	117
call_nodes	1	1	4	3	1	4	12
control_flow_nodes	35	44	26	50	32	38	126
type_nodes	18	3	2	12	14	9	50
ignore_nodes	2	3	3	5	5	4	10
Total	94	96	84	116	98	108	409

Nama-nama node yang berbeda dipetakan ke tipe-tipe umum untuk konsistensi. untuk enam bahasa pemrograman. Hasilnya menunjukkan kategori node yang tinggi dan merata, menunjukkan cakupan sintaksis yang merata di berbagai bahasa.

Tabel 4 Contoh pemetaan tipe node

Tipe Node	Dipetakan menjadi
function_nodes	function, function_definition, method, class_declaration
variable_nodes	identifier, name
assignment_nodes	assignment, assignment_expression
call_nodes	call, call_expression
control_flow_nodes	if, for, while
ignore_nodes	call, call_expression

Tabel 5 menyajikan persentase kemunculan kategori node dalam dataset CodeXGLUE setelah menormalkan dan menyeragamkan tipe node AST. Kategori ASSIGNMENT_NODES menunjukkan variasi minimal antar bahasa dengan Ruby memiliki persentase lebih rendah (81,2%) karena deklarasi variabelnya yang fleksibel. Sementara Python dan JavaScript memiliki persentase lebih tinggi karena model penugasan eksplisit dalam AST mereka. Secara keseluruhan, hasilnya mengonfirmasi ekstraksi AST yang konsisten dan normalisasi tipe node di berbagai bahasa.

Meskipun terdapat variasi dalam ASSIGNMENT_NODES dan TYPE_NODES, akan tetapi dataset ini mempertahankan cakupan sintaksis yang komprehensif dan seragam, memastikan analisis lintas bahasa yang merata dan akurat untuk penelitian pembelajaran mesin.

Tabel 5 Persentase kemunculan kategori-kategori node

Kategori	Go (%)	Python (%)	Ruby (%)	PHP (%)	Java (%)	Javascript (%)
FUNCTION_NODES	100	100	100	100	100	100
VARIABLE_NODES	100	100	100	100	100	100
ASSIGNMENT_NODES	93.3	100	81.2	94.7	89	97.3
EXPRESSION_NODES	100	100	100	99.9	99.9	100
CALL_NODES	100	100	100	100	100	100
CONTROL_FLOW_NODES	99	100	100	100	99.9	100
TYPE_NODES	95	95	95	95	95	99.8
IGNORE_NODES	100	100	100	99.8	100	100

3.4. Ekstraksi Grafik Aliran Data (DFG)

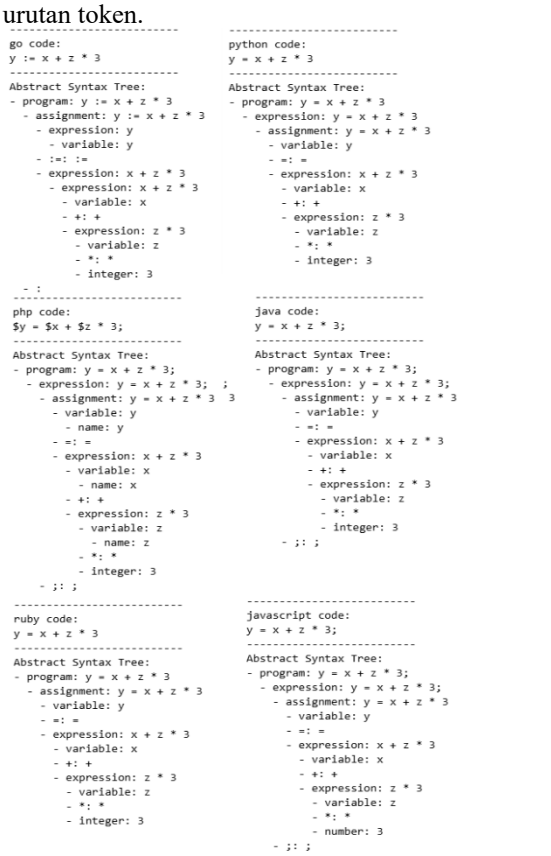
Aspek validasi krusial adalah memastikan bahwa AST yang diekstraksi mempertahankan ketergantungan data melalui analisis DFG. Ekstraksi DFG mengonfirmasi bahwa ketergantungan variabel dipertahankan di semua enam bahasa. Studi

sebelumnya (Guo et al., 2021; Shi et al., 2021) menekankan pentingnya validasi DFG dalam memastikan pemahaman kode semantik yang kuat. DFG diekstrak menggunakan pendekatan pencarian AST, di mana setiap ekspresi diperiksa untuk mengidentifikasi hubungan antara variabel dan operator dalam kode sumber.

Secara khusus, validasi ini melibatkan identifikasi variabel dalam AST, pemetaan hubungan antara variabel dan ekspresi, serta analisis ketergantungan data, yang mencakup hubungan antara ekspresi aritmatika, operasi logika, dan pemanggilan fungsi yang melibatkan data yang sama. Metode kami berhasil mempertahankan integritas aliran variabel.

3.5. Evaluasi Rekonstruksi Kode

Untuk mengevaluasi sejauh mana informasi sintaksis dapat dipertahankan, dilakukan proses rekonstruksi kode dari AST yang diekstrak, kemudian dibandingkan dengan kode sumber aslinya menggunakan metrik BLEU score (Papineni et al., 2002b; Evtikhiev et al., 2023). Proses rekonstruksi dilakukan melalui traversal pre-order, di mana setiap node AST diubah kembali menjadi urutan token.



Gambar 5 Struktur AST setelah diseragamkan

Pada penelitian ini, hasil rekonstruksi menunjukkan BLEU score 100, yang secara teoritis menunjukkan kesamaan sangat tinggi antara hasil rekonstruksi dan kode sumber. Namun, skor ini perlu ditafsirkan dengan hati-hati. Hal ini dikarenakan pada struktur AST yang digunakan, khususnya hasil dari parser Tree-sitter, terdapat atribut eksplisit yang menyimpan salinan kode sumber asli secara lengkap.

Dengan demikian, nilai BLEU tinggi ini tidak sepenuhnya mencerminkan kemampuan rekonstruksi dari struktur AST itu sendiri, melainkan lebih pada fakta bahwa informasi kode sumber asli tersedia secara langsung dalam struktur AST.

3.6. Perbandingan dengan Dataset AST yang Ada

Pendekatan-pendekatan sebelumnya bergantung pada representasi AST yang tidak seragam, dataset kami dengan ketat menerapkan penyeragaman jenis node, untuk memastikan konsistensi lintas bahasa yang lebih baik. Tabel 6 menunjukkan struktur fitur dalam dataset CodeXGLUE_AST. Baris terakhir dalam Tabel 6 (code_ast) adalah struktur AST yang dihasilkan dari penelitian ini. Setiap entri dalam dataset ini mencakup informasi kode sumber, metadata fungsi, dan representasi AST dalam format JSON. Pada gambar 5 diperlihatkan contoh hasil AST yang ter-seragam untuk kode sumber yang sama di keenam bahasa.

Tabel 6 Fitur pada dataset CodeXGLUE_AST

Feature Name	Description
Repo	Repo/function owner
Path	Full path of the original file
func_name	Function name
original_string	Original text before tokenization
language	Programming language
Code	The original_string part which is the code
code_tokens	Token code version
docstring	Existing comment/summary on original_string
docstring_tokens	Token version docstring
code_ast*	Complete AST representation in json format

3.7. Pengujian Pada Tugas Peringkasan Kode

Untuk menguji dampak praktis dari dataset ini, kami melakukan eksperimen baseline pada tugas peringkasan kode (*code summarization)*. Pada percobaan ini digunakan *pretrained-model* CodeT5.

Tabel 7 Hasil Pengujian Dataset CodeXGLU AST Pada Tugas *Code Summarization*

Bahasa	Sumber Pelatihan	Nilai			
		BLEU	METEOR	ROUGE	ROUGE-L
go	ast-original	6.36	31.88	40.62	38.12
go	ast-unified	**6.72**	**32.2**	**40.77**	**38.19**
go	original	5.28	28.51	36.35	33.94
java	ast-original	4.66	19.85	34.74	30.09
java	ast-unified	**6.24**	**25.59**	**40.47**	**36.19**
java	original	1.98	11.68	22.63	19.68
javascript	ast-original	1.25	14.65	29.73	26.4
javascript	ast-unified	**1.36**	**14.71**	**29.79**	**26.34**
javascript	original	0.85	10.62	21.21	18.91
php	ast-original	1.06	20.12	36.09	33.79
php	ast-unified	**7.1**	**29.72**	**45.44**	**43.08**
php	original	1.04	14.3	24.87	22.94
python	ast-original	41.58	81.24	86.71	86.5
python	ast-unified	42.62	**81.92**	**87.22**	**87.03**

python	original	**44.91**	70.46	76.58	76.57
ruby	ast-original	2.43	15.34	26.57	22.63
ruby	ast-unified	**2.49**	**15.8**	**28.38**	**23.79**
ruby	original	1.58	11.71	20.58	17.55

Pada model CodeT5 dilakukan *fine-tuning* dengan data AST original (Belum diseragamkan), data AST yang diseragamkan dan dengan kode sumber. Dari ketiga model hasil dibandingkan performanya untuk tugas peringkasan kode. Tabel 7 menampilkan hasil akhir dari percobaan yang kami lakukan. Tabel 7 menunjukkan peningkatan nilai BLEU, METEOR, ROUGE dan ROUGE-L hampir pada semua percobaan saat menggunakan AST yang diseragamkan. Hanya pada bahasa pemrograman Python nilai BLEU kalah dengan pelatihan dengan kode sumber, meskipun nilai METEOR (Lavie and Denkowski, 2009), ROUGE dan ROUGE-L (Lin, 2004) masih unggul AST ter-seragam.

Dengan hasil evaluasi dan performa yang bagus tersebut, maka dataset CodeXGLUE_AST dapat memberikan kontribusi signifikan pada bidang analisis sintaksis berbasis pembelajaran mesin. Representasi AST yang diseragamkan memungkinkan eksplorasi lebih lanjut, termasuk integrasi dengan *Graph Neural Networks* (GNN) dan model berbasis transformer. Hasil penelitian ini juga membuka peluang untuk analisis lintas bahasa yang lebih akurat dan seragam dalam berbagai aplikasi kecerdasan buatan berbasis kode sumber.

4. DISKUSI

Penelitian ini telah berhasil menyeragamkan representasi *Abstract Syntax Tree* (AST) pada enam bahasa pemrograman utama: Go, Java, JavaScript, Python, Ruby, dan PHP. Proses penyeragaman ini dilakukan melalui klasifikasi tipe node ke dalam delapan kategori sintaksis guna memastikan konsistensi struktur AST lintas bahasa.

Dibandingkan dengan pendekatan sebelumnya seperti GraphCodeBERT yang fokus pada representasi *Data Flow Graph* (DFG), StructCoder yang menggabungkan AST dan DFG, maupun UAST dan SG-Trans yang menekankan kombinasi struktur lokal dan global, pendekatan kami memperluas cakupan dengan menyediakan representasi AST yang seragam secara eksplisit. Representasi ini sangat berguna untuk berbagai tugas pengolahan kode sumber seperti peringkasan kode, generasi kode, dan pelatihan model pembelajaran mesin yang sadar sintaksis.

Keunggulan utama dari pendekatan ini adalah kemampuannya untuk mengurangi bias dalam model pembelajaran mesin akibat perbedaan representasi AST antar bahasa. Dataset AST yang diseragamkan juga membuka peluang dalam pengembangan teknik perturbasi semantik untuk pelatihan *adversarial*. Dengan memanfaatkan struktur AST, dapat dibuat contoh kode *adversarial* yang mempertahankan integritas semantik sambil memodifikasi elemen seperti nama variabel, struktur fungsi, maupun alur kontrol. Teknik ini berpotensi meningkatkan ketahanan model ringkasan kode berbasis transformer terhadap serangan sintaksis.

4.1. Tantangan dan Keterbatasan Penelitian

Fokus utama dari studi ini adalah pada penyeragaman representasi AST. Oleh karena itu, dampak langsung dari dataset ini terhadap peningkatan performa model pembelajaran mesin belum dieksplorasi secara mendalam. Tujuan utama adalah menyediakan dataset AST yang konsisten sebagai landasan untuk riset lebih lanjut.

Beberapa tantangan masih perlu dicermati. Salah satu keterbatasan utama adalah belum adanya validasi komprehensif terhadap efektivitas representasi AST yang diseragamkan dalam skenario aplikasi nyata pembelajaran mesin. Selain itu, proses pemetaan berbagai jenis node ke dalam kategori umum meskipun meningkatkan konsistensi, dapat menyebabkan hilangnya informasi semantik spesifik yang mungkin berguna dalam konteks tertentu.

4.2. Arah Penelitian Masa Depan

Sebagai arah pengembangan selanjutnya, kami akan mengintegrasikan teknik perturbasi berbasis AST ke dalam pelatihan *adversarial* untuk model rangkuman kode berbasis transformer. Pendekatan ini akan melibatkan pembuatan contoh *adversarial* melalui modifikasi struktur AST secara sistematis dengan tetap menjaga semantik program.

Selain itu, perluasan representasi AST ke bahasa pemrograman lain dan integrasi dengan model semantik serta *Graph Neural Network* (GNN) juga akan menjadi fokus. Upaya ini diharapkan dapat meningkatkan pemahaman sintaksis dan semantik dalam pengolahan kode lintas bahasa secara lebih menyeluruh dan tangguh.

5. KESIMPULAN

Penelitian ini menunjukkan bahwa representasi *Abstract Syntax Tree* (AST) dapat digunakan secara efektif sebagai bentuk representasi sintaksis dalam pembelajaran mesin berbasis kode sumber. Dengan menyeragamkan struktur AST untuk enam bahasa pemrograman—Go, Java, JavaScript, Python, Ruby, dan PHP—penelitian ini berhasil menghasilkan dataset AST yang konsisten, lengkap, dan akurat dalam merepresentasikan informasi sintaksis.

Melalui evaluasi kelengkapan informasi, rekonstruksi kode, dan pengujian pada tugas peringkasan kode, dataset ini terbukti mampu meningkatkan performa model pembelajaran mesin, khususnya dalam tugas-tugas yang membutuhkan pemahaman struktur sintaksis, seperti *code summarization*. Hasil evaluasi juga menunjukkan bahwa representasi AST yang diseragamkan memberikan skor BLEU, METEOR, ROUGE, dan

ROUGE-L yang lebih baik dibandingkan AST asli dan kode sumber pada sebagian besar bahasa pemrograman yang diuji.

Kontribusi utama dari penelitian ini adalah penyediaan dataset AST yang seragam dalam format JSON terstruktur, yang sebelumnya belum tersedia dalam penelitian terdahulu. Dataset ini memiliki potensi besar untuk digunakan dalam riset lanjutan, terutama untuk pengembangan model transformer yang sadar sintaks, pelatihan *adversarial*, dan eksplorasi representasi kode yang lebih semantik.

Ke depan, penelitian ini dapat diperluas dengan mengintegrasikan model semantik serta memperluas cakupan bahasa pemrograman untuk menguji ketahanan dataset terhadap variasi sintaksis. Selain itu, integrasi dengan analisis *Data Flow Graph* (DFG) dan penerapan teknik augmentasi berbasis AST juga diharapkan dapat meningkatkan ketahanan dan generalisasi model dalam memahami kode sumber secara lebih mendalam.

DAFTAR PUSTAKA

ALIKHANIFARD, P. AND TSANTALIS, N., 2024. A Novel Refactoring and Semantic Aware Abstract Syntax Tree Differencing Tool and a Benchmark for Evaluating the Accuracy of Diff Tools. *ACM Trans. Softw. Eng. Methodol.* [online] https://doi.org/10.1145/3696002.

ANON. 2019. *Tree Sitter - Online Documentation*. [online] Available at: <https://github.com/tree-sitter>.

CHA, S., TAYLOR, R.N. AND KANG, K. EDS, 2019. *Handbook of Software Engineering*. [online] Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-00262-6.

CHEN, Q., HU, H. AND LIU, Z., 2019. Code Summarization with Abstract Syntax Tree. In: T. Gedeon, K.W. Wong and M. Lee, eds. *Neural Information Processing*. Cham: Springer International Publishing. pp.652–660. https://doi.org/10.1007/978-3-030-36802-9_69.

DEHAERNE, E., DEY, B., HALDER, S., DE GENDT, S. AND MEERT, W., 2022. Code Generation Using Machine Learning: A Systematic Review. *IEEE Access*, 10, pp.82434–82455. https://doi.org/10.1109/ACCESS.2022.3196347.

DURACIK, M., HRKUT, P., KRSAK, E. AND TOTH, S., 2020. Abstract Syntax Tree Based Source Code Antiplagiarism System for Large Projects Set. *IEEE Access*, 8, pp.175347–175359. https://doi.org/10.1109/ACCESS.2020.3026422.

EVTIKHIEV, M., BOGOMOLOV, E., SOKOLOV, Y. AND BRYKSIN, T., 2023. Out of the BLEU: How should we assess quality of the Code Generation models? *Journal of Systems and Software*, 203, p.111741. https://doi.org/10.1016/j.jss.2023.111741.

FRANKISH, K. AND RAMSEY, W.M. eds, 2014. *The Cambridge handbook of artificial intelligence*. Cambridge, UK: Cambridge University Press.

GAO, S., GAO, C., HE, Y., ZENG, J., NIE, L., XIA, X. AND LYU, M., 2023. Code Structure–Guided Transformer for Source Code Summarization. *ACM Transactions on Software Engineering and Methodology*, 32(1), p.23:1-23:32. https://doi.org/10.1145/3522674.

GUO, D., REN, S., LU, S., FENG, Z., TANG, D., LIU, S., ZHOU, L., DUAN, N., SVYATKOVSKIY, A., FU, S., TUFANO, M., DENG, S.K., CLEMENT, C., DRAIN, D., SUNDARESAN, N., YIN, J., JIANG, D. AND ZHOU, M., 2021. GRAPHCODEBERT: PRE-TRAINING CODE REPRESEN- TATIONS WITH DATA FLOW.

KUANG, L., ZHOU, C. AND YANG, X., 2022. Code comment generation based on graph neural network enhanced transformer model for code understanding in open-source software ecosystems. *Automated Software Engineering*, 29(2), p.43. https://doi.org/10.1007/s10515-022-00341-1.

LATIF, A., AZAM, F., ANWAR, M.W. AND ZAFAR, A., 2023. Comparison of Leading Language Parsers – ANTLR, JavaCC, SableCC, Tree-sitter, Yacc, Bison. In: *2023 13th International Conference on Software Technology and Engineering (ICSTE)*. [online] 2023 13th International Conference on Software Technology and Engineering (ICSTE). pp.7–13. https://doi.org/10.1109/ICSTE61649.2023.00009.

LAVIE, A. AND DENKOWSKI, M.J., 2009. The Meteor metric for automatic evaluation of machine translation. *Machine Translation*, 23(2), pp.105–115. https://doi.org/10.1007/s10590-009-9059-4.

LIN, C.-Y., 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In: *Text Summarization Branches Out*. [online] Barcelona, Spain: Association for Computational Linguistics. pp.74–81. Available at: <https://aclanthology.org/W04-1013> [Accessed 15 September 2024].

LU, S., GUO, D., REN, S., HUANG, J., SVYATKOVSKIY, A., BLANCO, A., CLEMENT, C., DRAIN, D., JIANG, D., TANG, D., LI, G., ZHOU, L., SHOU, L., ZHOU, L., TUFANO, M., GONG, M., ZHOU, M., DUAN, N., SUNDARESAN, N., DENG, S.K., FU, S. AND LIU, S., 2021a. *CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation*. https://doi.org/10.48550/arXiv.2102.04664

LU, S., REN, S., GUO, D., SVYATKOVSKIY, A. AND HUANG, J., 2021b. *CodeXGLUE Github Repository*. [online] Available at: <https://github.com/microsoft/CodeXGLUE>.

PAPINENI, K., ROUKOS, S., WARD, T. AND ZHU, W.-J., 2002a. Bleu: a Method for Automatic Evaluation of Machine Translation. In: P. Isabelle, E. Charniak and D. Lin, eds. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. [online] ACL 2002. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics. pp.311–318. https://doi.org/10.3115/1073083.1073135.

PAPINENI, K., ROUKOS, S., WARD, T. AND ZHU, W.-J., 2002b. Bleu: a Method for Automatic Evaluation of Machine Translation. In: P. Isabelle, E. Charniak and D. Lin, eds. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. [online] ACL 2002. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics. pp.311–318. https://doi.org/10.3115/1073083.1073135.

PARR, T., 2014. *The definitive ANTLR 4 reference*. Book version: P 2.0 edn. The pragmatic programmers. Dallas, Texas Raleigh, North Carolina: The Pragmatic Bookshelf.

SEBESTA, R.W., 2012. *Concepts of programming languages*. 10. ed edn. Always learning. Boston: Pearson.

SHAH, M., PATEL, R. AND TERRY, A., 2023. *Structural Syntax Network for Code Classification*. https://doi.org/10.20944/preprints202312.0805.v1.

SHI, T., KENESHLOO, Y., RAMAKRISHNAN, N. AND REDDY, C.K., 2021. Neural Abstractive Text Summarization with Sequence-to-Sequence Models. *ACM/IMS Trans. Data Sci.*, 2(1), p.1:1-1:37. https://doi.org/10.1145/3419106.

SISWO UTOMO, M., UTAMI, E., KUSRINI AND SETYANTO, A., 2024. Machine Learning Innovations in Code Generation: A Systematic Literature Review of Methods, Challenges and Directions. In: *2024 International Conference on Information Technology and Computing (ICITCOM)*. [online] 2024 International Conference on Information Technology and Computing (ICITCOM). pp.24–29. https://doi.org/10.1109/ICITCOM62788.2024.10762291.

SUN, W., FANG, C., CHEN, Y., ZHANG, Q., TAO, G., YOU, Y., HAN, T., GE, Y., HU, Y., LUO, B. AND CHEN, Z., 2024. An Extractive-and-Abstractive Framework for Source Code Summarization. *ACM Transactions on Software Engineering and Methodology*, 33(3), pp.1–39. https://doi.org/10.1145/3632742.

TAKERNGSAKSIRI, W., TANTITHAMTHAVORN, C. AND LI, Y.-F., 2024. Syntax-aware on-the-fly code completion. *Information and Software Technology*, [online] 165(C). https://doi.org/10.1016/j.infsof.2023.107336.

TIPIRNENI, S., ZHU, M. AND REDDY, C.K., 2024. *StructCoder: Structure-Aware Transformer for Code Generation*. Available at: <http://arxiv.org/abs/2206.05239> [Accessed 11 September 2024].

UTOMO, M., UTAMI, E., KUSRINI, K. AND SETYANTO, A., 2025. CodeXGLUE AST Dataset. Available at: <https://github.com/mardiutomo75/CodeXGLUE_AST>.

WANG, K., YAN, M., ZHANG, H. AND HU, H., 2022. Unified Abstract Syntax Tree Representation Learning for Cross-Language Program Classification. In: *2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC)*. [online] 2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC). pp.390–400. https://doi.org/10.1145/3524610.3527915.

WANG, Y., LE, H., GOTMARE, A.D., BUI, N.D.Q., LI, J. AND HOI, S.C.H., 2023. *CodeT5+: Open Code Large Language Models for Code Understanding and Generation*. https://doi.org/10.48550/arXiv.2305.07922

ZHANG, J., LIU, Z., HU, X., XIA, X. AND LI, S., 2023. Vulnerability Detection by Learning From Syntax-Based Execution Paths of Code. *IEEE Transactions on Software Engineering*, 49(8), pp.4196–4212. https://doi.org/10.1109/TSE.2023.3286586