BAB II
TINJAUAN PUSTAKA

Penelitian oleh Gandhi dkk. (2024) mengkaji deteksi plagiarisme pada tugas pemrograman Python dengan pendekatan *machine learning* berbasis SVM. Masalah diformulasikan sebagai klasifikasi pasangan berkas. Dua berkas dinilai apakah mirip atau tidak. Fitur yang dipakai sederhana dan bisa dihitung di CPU. Contohnya kemiripan *token*, kemiripan urutan, serta *features* dari *syntax structure*. Fitur dinormalisasi agar perbedaan format tidak mendominasi. Model utama yang dipilih adalah SVM karena stabil di dimensi fitur sedang dan relatif mudah dijelaskan kontribusi fiturnya. Evaluasi dilakukan secara *supervised learning* dan dibandingkan dengan alat klasik seperti JPlag atau MOSS. Studi ini menegaskan bahwa pemilihan fitur yang tepat sudah cukup memberi kinerja baik tanpa model besar [4].

Awale dkk. (2022) menggabungkan fitur n-gram dan gaya penulisan untuk mengklasifikasikan pasangan kode tugas kuliah. Mereka membandingkan beberapa model klasik seperti SVM dan *boosting*. Skema evaluasi memakai pemisahan data latih dan uji yang jelas. Hasilnya menunjukkan model *boosting* sering unggul pada kombinasi fitur leksikal dan gaya yang dirancang cermat. Pendekatan ini ringan dan cocok untuk laboratorium kampus. Komponen fiturnya juga dapat diinspeksi saat dosen melakukan verifikasi manual [5].

Pada ranah Java, Hosam dkk. (2022) mengevaluasi beberapa set fitur untuk klasifikasi tradisional. Mereka menyusun kelompok fitur seperti histogram struktur, fitur leksikal per kelas, dan hitungan elemen struktur program. *Dataset* yang digunakan adalah SOCO-14. Setelah tahap *preprocessing* dan penyusunan fitur, peneliti membandingkan kinerja beberapa set fitur klasifikasi pada model *machine learning* yang sama. Mereka menyoroti efisiensi saat memproses puluhan ribu pasangan. Temuan ini menunjukkan bahwa fitur yang menonjolkan isi program bisa meningkatkan kinerja tanpa biaya komputasi berlebih [6].

Penelitian oleh Mehsen dan Joshi (2024) menyajikan pendekatan pada C++ dengan *pipeline* sederhana. *Source code* diubah menjadi *token-based* TF-IDF

vector, kemudian diklasifikasikan memakai Random Forest. Strategi ini memanfaatkan kemudahan ekstraksi fitur berbasis teks dan ketangguhan Random Forest terhadap fitur yang kurang informatif. Evaluasi menggunakan skema *hold-out* atau *k-fold* yang umum. Hasil yang dilaporkan menunjukkan akurasi tinggi pada data C++. Walau rincian penyeimbangan kelas tidak dijelaskan secara detail, studi ini menunjukkan kombinasi *vektorisasi* teks dan model tradisional cukup kuat untuk *code similarity detection* [7].

Selain pendekatan tradisional tersebut, penelitian oleh Ebrahim dan Joy (2023) memperkenalkan penggunaan *embedding* model *pretrained* (CodePTMs) untuk mendeteksi plagiarisme *source code* [8]. Tiga model PLBART, UnixCoder, dan CodeBERTa digunakan untuk menghasilkan representasi vektor yang memuat aspek *syntax* dan semantik kode. Cosine similarity antar *embedding* dipakai sebagai fitur untuk klasifikasi, sementara *AutoML* digunakan untuk memilih model terbaik. Pengujian menggunakan *Dataset* SOCO pada skenario Java dan C/C++ menunjukkan bahwa pendekatan ini mampu mengungguli *baseline* seperti JPlag dan MOSS, khususnya pada metrik *precision*. Penelitian ini menyoroti potensi model *embedding* modern dalam menangani plagiarisme dengan variasi struktur dan gaya.

Eppa dan Murali (2022) mengevaluasi berbagai algoritma *machine learning* dan *deep learning*, termasuk KNN, SVM, *Decision Tree*, *Recurrent Neural Network*, dan *Transformer*, untuk mendeteksi plagiarisme pada tugas pemrograman C [9]. *Dataset* berisi 1241 pasangan kode, di mana fitur yang digunakan mencakup 65 fitur statis MILEPOST GCC serta lima fitur berbasis kesamaan, seperti *edit distance* dan jumlah baris umum. SVM menunjukkan kinerja terbaik dengan akurasi 99.59%, sedangkan model *deep learning* juga memberikan hasil yang stabil pada kasus *obfuscation*. Penelitian ini menunjukkan bahwa kombinasi fitur statis dan fitur kemiripan mampu memberikan performa yang sangat kompetitif dan lebih stabil dibandingkan pendekatan tekstual seperti MOSS atau JPlag.

Penelitian oleh Fokam dan Ajoodha (2021) mengembangkan pendekatan *deep learning* berbasis struktur program melalui modifikasi model *Abstract*

Syntax Tree Neural Network (ASTNN) menggunakan *contrastive learning* [10]. ASTNN memanfaatkan representasi pohon *syntax* (AST) untuk menangkap struktur dan urutan logika program secara lebih akurat dibandingkan tokenisasi biasa. Dalam studinya, model ASTNN asli diubah menjadi arsitektur *Siamese* yang dilatih dengan *contrastive loss*, sehingga model dapat mempelajari jarak antar *embedding* kode secara lebih diskriminatif. *Dataset* yang digunakan berasal dari *Open Judge System* dan mencakup lebih dari 26.000 pasangan kode C dengan proporsi *clone pair* yang tidak seimbang. Hasil eksperimen menunjukkan bahwa model ASTNN berbasis *contrastive learning* (ASTNN-c) mampu meningkatkan kinerja F1-score dari 0.935 menjadi 0.983, sekaligus mencapai *recall* hingga 99.2%. Temuan ini menegaskan bahwa representasi *syntax* berbasis AST dan pembelajaran *kontrastif* mampu menangani variasi perubahan struktur kode yang kompleks, serta memberikan performa yang stabil pada tugas *code similarity detection* berskala besar.

Enam fitur kemiripan yang digunakan dalam penelitian ini disusun dengan merujuk pada temuan-temuan penelitian terdahulu. Kemiripan token dan normalisasi teks mengacu pada pendekatan yang digunakan oleh Gandhi dkk. [4] serta Mehsen dan Joshi [7], yang menunjukkan bahwa representasi token yang dinormalisasi efektif untuk membedakan kemiripan kode. Kemiripan urutan logika dan kemiripan baris logika diadopsi dari gagasan kesamaan struktur dan alur program sebagaimana dibahas oleh Hosam dkk. [6] serta Fokam dan Ajoodha [10]. Kemiripan komentar dan pola penamaan variabel merujuk pada fitur gaya penulisan dan karakteristik kode yang digunakan oleh Awale dkk. [5] dan Eppa serta Murali [9]. Dengan merangkum dan menyederhanakan fitur-fitur yang telah terbukti efektif tersebut, penelitian ini merancang enam fitur kemiripan yang relevan, efisien, dan sesuai untuk mendeteksi plagiarisme pada tugas pemrograman mahasiswa.

Tabel 2. 1 Perbandingan Penelitian

Item Pembanding	**Penelitian**							
	Gandhi dkk	**Awale dkk**	**Hosam dkk**	**Mehsen & Joshi**	**Ebrahim & Joy**	**Eppa & Murali**	**Fokam & Ajoodha**	**Penelitian ini**
Domain	Python tugas mahasiswa	Tugas pemrograman umum	Java (SOCO-14)	C++ & Python	Java, C/C++ (SOCO)	C *programming*	C (*Open Judge*, pendidikan)	Java (Tugas pemrograman Mahasiswa)
Model/Metode	SVM (*pairwise*)	SVM, *Boosting*	Klasifikasi berbasis *feature set*	Random Forest, TF-IDF	*Pre-trained embeddings + AutoML*	SVM, KNN, DT, RNN, *Transformer*	ASTNN (*recursive NN*, *Siamese*, *contrastive loss*)	Linear SVM, dibandingkan, *Logistic Regression, Random Forest, Gradient Boosting*
Fitur	*Token*, urutan, *syntax*	N-gram, gaya penulisan	Histogram, struktur, leksikal	*TF-IDF tokens*, *code metrics*	Embedding (PLBART, UnixCoder, CodeBERT	65 *code static*, 5 *similarity*, *deep features*	*AST statement tree*, GRU, *embedding*	enam fitur: baris, urutan logika, komentar,

					a), *cosine similarity*			variabel, *token*
Dataset & Evaluasi	Pasangan tugas Python dengan pendekatan *supervised learning* dibandingkan alat klasik	*Train/test split*, *n-gram*/gaya	SOCO-14; skala besar	880 solusi (44 *problem*), *hold-out/k-fold*	SOCO: Java (12k *test*), C++ (19k *test*), multi-skenario	2482 program (1241 pasang), *balanced plagiarized/non-plagiarized*	26.000+ pasangan kode C, *Open Judge*, 60% *train*, 20% valid, 20% *test*	ConPlag Java + data sintetis, *group-aware split*
Metrik	*Precision*, *Recall*, F1	*Accuracy*, F1	F-measure, *Precision*, *Recall*	Akurasi, *Precision*, *Recall*, F1	*Precision*, *Recall*, F1-score	Akurasi, *Precision*, *Recall*, F1	*Precision*, *Recall*, F1-score	*Precision*, *Recall*, F1, AUC
Hasil Utama	Kinerja baik tanpa model besar	*Boosting* unggul	F-measure tinggi fitur terbaik	K-means 99.2%, RF 93.5%, unggul MOSS	F1 Java 0.69, C++ 0.49, lebih baik dari *baseline*	SVM 99.6%, KNN 98%, *Transformer*/RNN >90%	ASTNN-c (*contrastive*): F1=0.983, *Precision*=0.975, *Recall*=0.992, +5% dari ASTNN biasa	Linear SVM stabil, mudah dijelaskan dan siap digunakan

Kekurangan	Spesifik Python, data terbatas	Jumlah data tidak lengkap	Perlu uji *cross-Dataset* lain	Fokus C++/Python, belum tangani *obfuscation* berat	*Imbalanced Dataset*, limit *token embedding*	*Dataset* spesifik C, perlu uji ke bahasa lain	Perlu *resource* besar, waktu *training* lama (20 jam/model)	Bergantung pada fitur buatan, perlu uji lintas *dataset*

Tabel 2.1 menunjukkan perkembangan yang konsisten dalam deteksi plagiarisme *source code*. Fitur-fitur yang menonjolkan isi program dan normalisasi konten permukaan umumnya lebih efektif dalam menekan *false positive* akibat perbedaan format atau perubahan sederhana. Model tradisional seperti SVM, *Random Forest*, atau *Gradient boosting* mampu memberikan kinerja baik jika rekayasa fitur dilakukan secara cermat dan evaluasi menggunakan pemisahan data yang tepat. Di sisi lain, pendekatan berbasis *embedding* dan *deep learning* mutakhir, seperti ASTNN dengan *contrastive learning*, cenderung lebih tangguh menghadapi modifikasi struktur kode yang kompleks dan variasi *obfuscation*, meski memerlukan sumber daya komputasi yang lebih besar. *Dataset* terstandar seperti SOCO dan ConPlag menyediakan landasan uji yang relevan, terutama untuk bahasa Java dalam konteks tugas pemrograman di perkuliahan. Berlandaskan berbagai temuan tersebut, penelitian ini memilih strategi enam fitur kemiripan yang mudah dijelaskan, menerapkan *group-aware split* untuk evaluasi yang valid dan mencegah *data leakage*, serta mengembangkan aplikasi web yang memudahkan dosen untuk meninjau skor fitur dan hasil prediksi secara langsung.