BAB III

METODOLOGI PENELITIAN

3.1 Metode Penelitian

Penelitian ini menggunakan pendekatan kuantitatif dengan metode eksperimen untuk mengembangkan *dataset* sintetis dengan mengunakan varian model *Generative Adversarial Networks (GAN)* yaitu *Conditional Tabular Generative Adversarial Networks (CTGAN)* dan *Conditional Tabular Generative Adversarial Networks (CTGAN)*. Dua model *GAN* tersebut digunakan untuk menggenerasi dua *dataset* sintetis yang berbeda, yaitu *dataset header* paket jaringan dan *dataset payload* jaringan.

Model *CTGAN* dipilih untuk menghasilkan data sintetis pada fitur *header* paket jaringan karena kemampuannya dalam menangani data tabular dengan atribut campuran numerik dan kategorikal secara efektif. Sedangkan untuk *payload query SQL*, digunakan *Conditional Wasserstein GAN (CWGAN)* yang mampu menghasilkan data sekuensial dan tekstual dengan stabilitas pelatihan yang lebih baik serta kualitas data sintetis yang lebih realistis.

3.2 Alur Penelitian

Pada penelitian ini dilakukan beberapa tahapan utama untuk mencapai tujuan pengembangan *dataset* sintetis. Di awali dengan mengumpulkan *dataset* asli. *Dataset* asli yang digunakan terdiri dari dua bagian utama, yaitu *header* paket jaringan dan *payload query SQL*. Data ini diperoleh dari penelitian yang membahas tentang serangan *SQL injection*. Selanjutnya yaitu pra-pemrosesan data *header*.

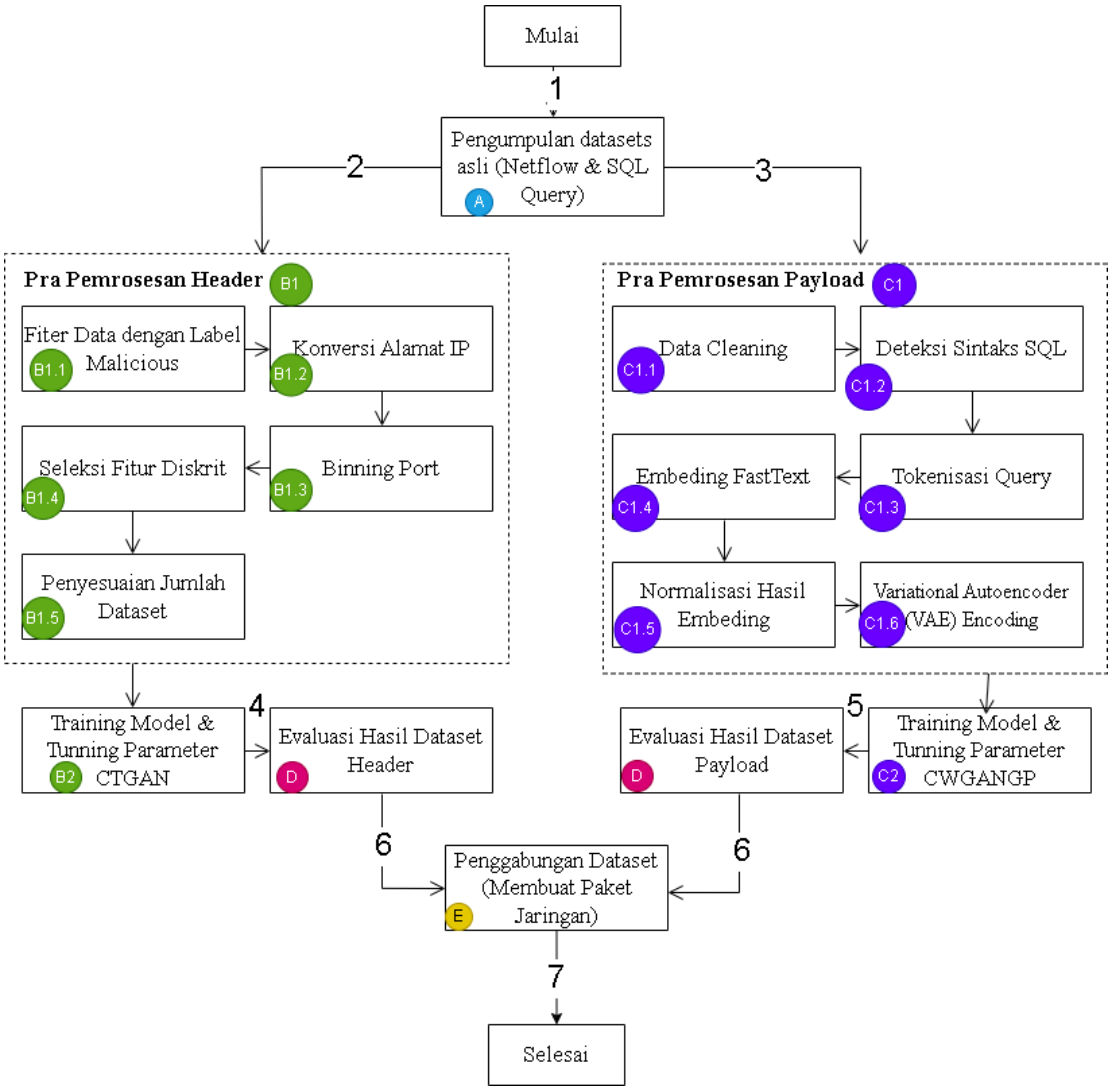
Data *header* paket jaringan diproses dengan melakukan pembersihan data, normalisasi fitur numerik, dan *encoding* fitur kategorikal seperti alamat *IP* dan *port*. Data yang sudah bersih kemudian dibagi menjadi batch untuk proses pelatihan model generatif. Pada sisi lain dilakukan pra-pemrosesan data *payload Payload query SQL* diproses dengan langkah tokenisasi untuk mengubah *query* menjadi token-token yang dapat dipahami oleh model.

Setelah dilakukan pra-pemrosesan data dilanjutkan dengan pengembangan *dataset* sintetis dengan dua model *Generative Adversarial Networks (GAN)* digunakan secara paralel untuk menghasilkan *dataset* sintetis:

1. *CTGAN* digunakan untuk menghasilkan data sintetis pada fitur *header* paket jaringan yang berbentuk data tabular dengan atribut numerik dan kategorikal.
2. *Conditional Wasserstein GAN (CWGAN)* digunakan untuk menghasilkan data sintetis *payload query SQL* yang bersifat sekuensial dan tekstual, dengan stabilitas pelatihan yang lebih baik dan kualitas data yang realistis.

Setelah proses generasi data sintetis dilakukan, khusus untuk *dataset SQL Injection* dilakukan detokenisasi *payload* sintetis. Hasil *output* dari model *CWGAN* berupa token-token disusun kembali menjadi *query SQL* melalui proses detokenisasi menggunakan kamus token yang telah dibuat sebelumnya. Di lanjutkan dengan penggabungan *dataset* sintetis. *Dataset header* sintetis hasil *CTGAN* dan *payload* sintetis hasil *WGAN* digabungkan untuk membentuk *dataset* lengkap yang merepresentasikan paket jaringan secara menyeluruh.

Evaluasi dilakukan dengan menganalisis hasil pelatihan model generatif dan kualitas *dataset* sintetis yang dihasilkan. Evaluasi ini menggunakan visualisasi berupa plot distribusi fitur, kurva *loss training*, dan perbandingan statistik antara data asli dan sintetis. Alur penelitian secara garis besar digambarkan pada gambar 3.1.



Gambar 3.1 Alur Penelitian

3.3 Pengumpulan Dataset Asli

Pada penelitian ini, proses pengumpulan dataset asli dilakukan dengan memanfaatkan dua sumber data yang berbeda sesuai dengan karakteristik fitur yang akan diolah. Dataset header paket jaringan diperoleh dari *SQL Injection Attack NetFlow Dataset* yang dikembangkan oleh (Ignacio & Campazas, 2022), sebagaimana telah dijelaskan pada Subbab 2.7. Dataset ini menyediakan fitur *NetFlow* yang merepresentasikan atribut header seperti alamat IP, port, protokol, dan karakteristik trafik lainnya dalam bentuk data tabular. Sementara itu, dataset payload berupa query SQL Injection diperoleh dari dataset yang dipublikasikan oleh Abu Syeed Sajid Ahmed (Kaggle, 2021), yang juga digunakan dalam penelitian (Dasari et al., 2025) untuk analisis dan pengembangan model generatif pada deteksi SQL Injection. Penggunaan dua dataset ini memungkinkan penelitian memisahkan karakteristik data tabular (header) dan data sekuensial (payload) secara sistematis sebelum diproses menggunakan arsitektur GAN yang sesuai.

3.4 Pra Pemrosesan Header

3.4.1 Filter Data dengan Label *Malicious*

Pada tahap awal proses penelitian, dilakukan penyaringan (filtering) dataset untuk memilih data dengan label malicious. Filter ini dilakukan untuk memastikan bahwa data yang digunakan pada proses pelatihan model hanya yang relevan dengan tujuan penelitian, yaitu mendeteksi dan mensintesis data serangan jaringan yang bersifat berbahaya. Dataset awal yang diperoleh memiliki jumlah total 400.003 sampel,

di mana data tersebut mencakup berbagai fitur header jaringan seperti unix_secs, unix_nsecs, sysuptime, exaddr, dpkts, dan kolom-kolom protokol jaringan lainnya.

Setelah proses filter menggunakan kondisi label malicious, jumlah sampel yang memenuhi kriteria berkurang menjadi 200.002. Pengurangan jumlah sampel ini bukan tanpa alasan, karena data non-malicious tidak dibutuhkan untuk membangun model deteksi serangan yang lebih spesifik dan akurat. Pada dataset hasil filter, distribusi fitur header juga diperhatikan untuk memastikan konsistensi dan validitas data yang akan digunakan lebih lanjut.

3.4.2 Konversi Alamat IP

Pada proses pra pemrosesan dataset header, salah satu tahap penting yang dilakukan adalah konversi alamat IP ke bentuk yang lebih mudah diproses oleh model. Alamat IP pada umumnya tersusun dari empat oktet yang masing-masing memiliki rentang nilai dari 0 hingga 255. Dalam penelitian ini, alamat IP pada kolom sumber (srcaddr) dan tujuan (dstaddr) dipecah menjadi empat kolom terpisah, yaitu src_oct1 hingga src_oct4 dan dst_oct1 hingga dst_oct4. Hal ini bertujuan agar model dapat mengenali pola distribusi pada setiap oktet secara independen serta meningkatkan akurasi pemodelan sintesis data.

3.4.3 *Binning Port*

Binning atau dapat disebut juga dengan Bucketing merupakan rekayasa atau teknik untuk mengelompokan sub rentang numerik yang berbeda kedalam kelompok-kelompok atau bucket. Pada penelitian ini, proses *Binning* dilakukan pada kolom srcport dan dstport, karena kedua kolom tersebut memiliki rentang nilai yang sangat

luas, yaitu 1 sampai 65535. Proses Binning pada fitur ini dilakukan karena setelah percobaan untuk training model dan generate dataset tanpa proses tersebut hasilnya model tidak dapat meniru sebaran atau rentang nilai yang sesuai dengan dataset asli dikarenakan model kesulitan untuk mengenali pola secara efektif pada rentang nilai tersebut.

3.4.4 Seleksi Fitur Diskrit

Seleksi fitur diskrit merupakan tahap penting dalam pra pemrosesan dataset yang bertujuan untuk memilih fitur-fitur yang bersifat kategorikal atau diskrit dan relevan dalam proses pelatihan model generatif. Pada konteks penelitian ini, fitur diskrit yang digunakan berasal dari hasil konversi data header serta pengelompokan nilai numerik menjadi kategori diskrit, seperti hasil proses binning untuk fitur port.

Fitur diskrit yang dipilih terdiri dari beberapa kolom, antara lain nilai binning untuk srcport dan dstport, segmen segmen oktet alamat IP sumber dan tujuan, serta fitur protokol seperti tcp_flags, tos, dan prot. Seleksi fitur ini didasarkan pada pertimbangan efektivitas pemodelan, di mana model CTGAN memiliki keunggulan dalam menangani fitur diskrit secara khusus dengan menerapkan conditional vector sampling dan mode-specific normalization.

Jumlah fitur diskrit yang dipilih sebanyak 13 fitur, yang diharapkan dapat merepresentasikan karakteristik penting dari data asli tanpa menghilangkan informasi signifikan. Pemilihan fitur-fitur ini juga memastikan bahwa model dapat belajar distribusi data diskrit dengan lebih baik, sehingga hasil sintesis data menjadi lebih realistis dan sesuai distribusi asli.

3.4.5 Penyesuaian Jumlah Dataset

Setelah proses seleksi fitur dan pra pemrosesan data selesai dilakukan, tahap selanjutnya adalah penyesuaian jumlah *dataset* agar sesuai dengan kebutuhan pelatihan model CTGAN. Penyesuaian ini penting dilakukan untuk memastikan agar jumlah sampel yang digunakan merupakan kelipatan dari *batch size* yang telah ditentukan, sehingga proses *training* dapat berjalan optimal dan stabil.

Dataset hasil filter dengan label malicious awalnya memiliki 200.002 sampel. Namun untuk proses pelatihan, jumlah sampel disesuaikan menjadi 16.000 sampel. Penyesuaian ini merupakan langkah yang wajar mengingat saat pelatihan model generatif, terutama dengan data berdimensi besar, keberadaan jumlah batch yang tepat akan mempengaruhi efisiensi komputasi serta performa konvergensi model.

Selain itu, penyesuaian jumlah dataset ini juga dilakukan agar distribusi data tetap terjaga dan representatif. Pendekatan sampling secara acak (random sampling) digunakan untuk memilih data sekaligus mempertahankan keseimbangan distribusi fitur diskrit yang telah diseleksi sebelumnya.

3.5 *Training* Model *CTGAN*

Dalam proses pelatihan, model CTGAN dilatih menggunakan fungsi fit() dari library CTGAN dengan parameter utama seperti batch_size, PAC (Packing Parameter), dan epoch. Jumlah sampel yang digunakan disesuaikan agar memenuhi hubungan matematis, dan dapat dilihat pada persamaan (6)

$$\text{total\_data} = \text{batch\_size} \times \text{PAC} \times n \quad (6)$$

Dimana persamaan (6) n adalah jumlah iterasi (batch) yang pasti, sehingga total data dapat terbagi habis ke dalam batch-batch tanpa meninggalkan sisa. Penyesuaian ini penting agar training berjalan efektif secara computational dan stabil pada model.

Teknik PAC berasal dari konsep PacGAN yang menggabungkan beberapa sampel menjadi satu input untuk Discriminator, yang bertujuan mengurangi risiko mode *collapse*, yaitu kondisi dimana Generator hanya mengeluarkan variasi data terbatas sehingga kualitas sintesis menurun. Dengan memasukkan PAC sebagai hyperparameter, pelatihan menjadi lebih stabil dan pola data dapat dipelajari secara lebih representatif.

3.5.1 *Training* Model *CTGAN*

Penginisialisasian model menggunakan jumlah epoch sebanyak 300 menyatakan bahwa seluruh dataset akan dilalui secara penuh sebanyak 300 kali selama proses training berlangsung. Fungsi ctgan.fit() menerima dataset akhir (df_final) yang berisi fitur diskrit lengkap, dengan parameter discrete_columns yang memberitahukan kolom apa saja yang bersifat diskrit.

Penggunaan vektor kondisi (conditional vector) di CTGAN memungkinkan model mengoneksikan fitur diskrit dalam sampling kondisi yang lebih terstruktur, sehingga distribusi data sintetis yang dihasilkan lebih akurat dalam merepresentasikan data asli. Seluruh proses ini dicatat waktu pelatihannya menggunakan modul *time.time*() untuk evaluasi efisiensi waktu *training*.

Setelah inisialisasi, pelatihan dimulai dengan mencatat waktu awal menggunakan time.time(), kemudian fungsi ctgan.fit() dipanggil dengan parameter df_final sebagai dataset hasil prapemrosesan yang berisi fitur-fitur diskrit. Parameter discrete_columns=discrete_features digunakan untuk memberitahu CTGAN kolom mana saja yang bersifat diskrit, sehingga model dapat membentuk vektor kondisi (conditional vector) yang sesuai selama proses pembelajaran. Proses ini memungkinkan CTGAN mempelajari distribusi data dengan lebih baik.

3.5.2 *Tunning* Parameter

Untuk mendapatkan hasil yang paling baik dan optimal, pada penelitian ini dilakukan satu langkah penting yaitu dengan melakukan tunning atau penyesuaian nilai parameter. Proses ini dilakukan untuk menentukan kombinasi nilai parameter yang menghasilkan performa model terbaik. Dalam penelitian ini parameter pelatihan yang dituning meliputi jumlah epoch, ukuran batch, dan PAC (Packing Parameter).

Dalam penelitian yang dilakukan oleh Lei Xu dengan judul "Modeling Tabular Data using Conditional GAN", model CTGAN dilatih dengan batch size sebesar 500 dan 300 epoch, sebagai konfigurasi standar pada framework benchmark mereka. Hal ini bertujuan untuk memastikan bahwa model memiliki cukup iterasi untuk belajar distribusi data secara optimal. Epoch merupakan representasi satu siklus penuh ketika seluruh dataset digunakan untuk memperbarui bobot jaringan. Jumlah pac pada penelitian tersbut tidak disebutkan, maka pada penelitian ini dicoba beberapa jumlah pac berbeda dengan batch size 500 (Xu et al., 2019).

3.6 Pra Pemrosesan *Dataset Payload*

3.6.1 Data Cleaning

Tahap pertama dalam prapemrosesan dataset payload adalah data cleaning yang sangat penting untuk menjamin integritas dan kualitas data sebelum memasuki proses pemodelan lebih lanjut. Proses ini meliputi identifikasi, pembersihan, dan penghapusan data yang tidak memenuhi kriteria validitas, duplikat, ataupun data noisy yang dapat mengganggu performa model.

Dataset awal yang digunakan dalam penelitian ini memuat 22.000 query dari data mentah hasil pengumpulan sebelumnya. Data tersebut mengandung berbagai macam bentuk query, termasuk query dengan struktur yang tidak valid atau salah ketik yang harus dihapus agar tidak menimbulkan bias analisis. Oleh karena itu, proses cleaning berjalan dengan memeriksa kelengkapan kolom, validitas sintaks string, dan deteksi anomali seperti query kosong atau karakter yang tidak sesuai.

Setelah dilakukan pembersihan, sampel yang tersisa berkurang menjadi 21.935 baris, dengan total 65 query dihapus karena dianggap outliers atau data tidak lengkap. Pengurangan ini mengindikasikan bahwa sekitar 0,3% data merupakan data yang tidak layak diproses lebih lanjut. Penghapusan data tersebut dilakukan secara selektif agar dataset tetap mencerminkan distribusi data yang sebenarnya tanpa membawa noise berlebihan.

3.6.2 Deteksi Sintaks *SQL*

Tahap deteksi sintaks SQL merupakan proses krusial dalam prapemrosesan dataset payload yang bertujuan untuk mengidentifikasi apakah sebuah query mengandung unsur SQL atau tidak. Pada penelitian ini, pendefinisian kumpulan kata kunci SQL (*sql_keywords*) dan simbol khusus SQL (*sql_symbols*) dilakukan secara eksplisit agar dapat mengenali pola-pola umum pada query SQL maupun pola serangan seperti SQL Injection.

Kata kunci *SQL* yang dipilih meliputi perintah dasar seperti *SELECT, INSERT, UPDATE, DELETE*, serta konstruksi lanjutan seperti *JOIN, GROUP BY*, dan *ORDER BY*. Daftar ini diperluas dengan memasukkan kata kunci yang umum dipakai dalam berbagai query valid dan berpotensi disalahgunakan oleh penyerang.

Simbol khusus *SQL* yang masuk dalam kumpulan ini terdiri atas operator logika dan aritmatika seperti =, !=, <, >, +, -, serta tanda baca yang sering muncul dalam query seperti (,), ,, dan ;, termasuk juga simbol komentar SQL seperti -- dan pola /* … */ yang umum digunakan dalam penyusupan SQL Injection.

Untuk proses deteksi, fungsi contains_sql_syntax() diimplementasikan dengan tiga mekanisme utama:

1. Pencocokan ekspresi reguler (regex) terhadap kata kunci SQL dalam himpunan *sql_keywords*. Fungsi ini mencari keberadaan kata-kata tersebut dalam teks query secara case-insensitive.

2. Pencarian keberadaan simbol khusus dalam query dengan memeriksa setiap karakter terhadap daftar *sql_symbols* yang telah didefinisikan.
3. Aturan heuristik sederhana untuk mengenali pola khas serangan, misalnya ekspresi seperti SELECT * FROM yang digunakan secara eksplisit, atau pola logika 1=1 yang umum dipakai dalam eksploitasi SQL Injection.

Hasil deteksi ini kemudian diberi label biner dan disimpan pada kolom baru bernama Has_SQL_Syntax, dengan nilai 1 menandakan query mengandung sintaks SQL sedangkan 0 berarti tidak.

Selain pendeteksian sintaks, tahap ini juga melakukan standarisasi kolom *Label*. Jika label sudah dalam bentuk numerik (0 dan 1), maka langsung dikonversi ke tipe integer menggunakan fungsi astype(int) untuk memastikan konsistensi. Jika masih berupa teks seperti "benign" dan "malicious", maka dilakukan pemetaan dengan fungsi map() agar seluruh label menjadi representasi numerik yang siap untuk digunakan dalam pelatihan model. Jumlah query deteksi sintaks dapat dilihat pada Tabel 4.10.

3.6.3 Tokenisasi *Query*

Pada tahap ini dilakukan proses tokenisasi terhadap query SQL Injection agar setiap elemen dalam query dapat diuraikan menjadi satuan unit yang lebih kecil atau token. Tujuan dari tokenisasi adalah mengubah query yang semula berupa string panjang menjadi urutan token yang lebih terstruktur, sehingga pola sintaks maupun indikasi serangan dapat dipelajari oleh model dengan lebih baik.

Tokenisasi dilakukan menggunakan ekspresi reguler (*regex*) yang dirancang secara khusus agar sesuai dengan karakteristik sintaks SQL. Pola regex yang digunakan mampu mengenali berbagai komponen query, antara lain:

1. Komentar SQL seperti -- dan /* ... */ yang kerap digunakan dalam query ataupun pola serangan SQL Injection.
2. String literal yang ditulis dengan tanda petik tunggal '...' maupun petik ganda "...".
3. Kata kunci dan identifier seperti *SELECT, FROM, WHERE*, serta nama tabel atau kolom.
4. Simbol dan operator seperti =, !=, <, >, +, -, serta tanda kurung.

Penggunaan regex memungkinkan pemisahan token menjadi unit logis yang merepresentasikan arti dan fungsi query secara lengkap, sehingga detail struktur query dapat dianalisis secara efektif.

Setelah tokenisasi, dilakukan penyaringan berdasarkan panjang token untuk mengeliminasi query yang memiliki jumlah token terlalu sedikit atau sangat panjang agar input ke model seragam dan representatif. Filter ini juga membantu menjaga kestabilan proses *embedding* dan pelatihan model selanjutnya.

3.6.4 *Embeding FastText*

Setelah proses tokenisasi selesai, tahap selanjutnya adalah pembuatan embedding dengan menggunakan model FastText. Embedding merupakan teknik untuk mengubah token token yang berupa kata atau simbol menjadi representasi vektor numerik

berdimensi tetap yang dapat dipahami oleh model pembelajaran mesin. FastText dipilih karena kemampuannya menghasilkan embedding kata dengan memperhatikan sub-kata, sehingga efektif untuk menangani variasi token yang kompleks dan jarang ditemukan dalam domain query SQL Injection.

Proses pelatihan FastText dilakukan pada seluruh tokenized query yang telah disiapkan, sehingga model dapat belajar asosiasi konteks antar token secara menyeluruh. Training ini memungkinkan FastText membangun vocabulary sebanyak 22.410 token unik dari dataset. Vocabulary yang besar ini mencerminkan keragaman token yang muncul pada payload query dan berbagai pola serangan.

Representasi embedding yang dihasilkan memiliki dimensi 50, yang merupakan kompromi antara kapasitas representasi dan efisiensi komputasi. Embedding ini menyandikan karakteristik semantik dan sintaks token secara numerik, memudahkan model generatif untuk mengolah dan mensintesis query dengan kompleksitas pola yang tinggi.

Selama proses pelatihan model *FastText*, dilakukan visualisasi reduksi dimensi dengan menggunakan *Principal Component Analysis* (PCA) untuk memeriksa distribusi vektor embedding dalam ruang berdimensi rendah. Visualisasi ini tersimpan dalam bentuk interaktif yang memperlihatkan klaster token yang serupa secara semantik.

3.6.5 Normalisasi Hasil *Embedding*

Setelah berhasil dibuat embedding token menggunakan FastText, tahap berikutnya adalah melakukan normalisasi terhadap embedding tersebut. Normalisasi ini diperlukan untuk mengatur skala nilai vektor embedding agar berada dalam rentang konsisten sehingga memudahkan proses pelatihan model generatif serta mencegah nilai numerik yang terlalu besar atau kecil yang dapat mengganggu kestabilan model.

Embedding yang dihasilkan memiliki dimensi (N, T, E) yaitu 21.253 query, dengan panjang token maksimum 50 dan dimensi embedding 50. Masing-masing mbedding token dikonversi menjadi vektor bernilai floating point. Namun nilai-nilai tersebut secara langsung dapat memiliki rentang yang bervariasi tergantung kata dan kontekstualisasinya dalam FastText.

Selain itu, teknik padding juga diterapkan untuk menyamakan panjang setiap sequence token menjadi 50, di mana embedding token yang kurang pada query pendek dilengkapi dengan vektor nol (zero padding) agar input data memiliki format seragam. Penyesuaian ini penting karena kebanyakan model pembelajaran mesin memerlukan input berdimensi tetap.

3.6.6 Variational Autoencoder (VAE) Encoding

Pada tahap encoding menggunakan *Variational Autoencoder* (VAE), data query SQL Injection yang telah melalui proses embedding dengan FastText diterjemahkan ke dalam representasi laten berdimensi lebih kecil namun tetap memuat informasi penting struktur query. VAE memiliki keunggulan dalam mengkompresi data sambil

mempertahankan fitur distribusi asli yang esensial, sehingga memudahkan proses pembelajaran oleh model generatif berikutnya.

3.7 *Training* model *CWGAN-GP*

Pada tahap awal pelatihan model Conditional Wasserstein GAN with Gradient Penalty (CWGAN-GP), dilakukan konfigurasi parameter dan persiapan lingkungan eksekusi yang bertujuan mengoptimalkan proses pelatihan. Parameter yang ditetapkan mencakup dimensi latent vector, jumlah epoch, ukuran batch, frekuensi pembaruan critic, serta parameter optimasi seperti learning rate dan faktor momentum.

3.7.1 Konfigurasi Awal

Dimensi vektor laten sebesar 100 digunakan sebagai input awal ke generator. Vektor ini merepresentasikan ruang abstrak dari data asli dalam bentuk angka acak yang kemudian dipetakan menjadi data payload sintetik berstruktur. Pelatihan dilakukan selama 100 epoch, yaitu 100 siklus penuh data dilalui agar generator mampu memahami pola distribusi dari payload hasil encoding. Ukuran batch sebanyak 64 sampel dipilih sebagai kompromi antara kecepatan pelatihan dan kestabilan model.

Pada karakteristik WGAN, critic diperbarui sebanyak lima kali lebih sering dibanding generator (n_critic=5). Hal ini memastikan kualitas umpan balik dari critic selalu terkini dan mempercepat konvergensi model.

Optimasi menggunakan algoritma Adam dengan learning rate 0.0002 dan parameter beta_1=0.5, beta_2=0.9. Kombinasi ini memberikan kestabilan pelatihan lebih baik dan mengurangi loss selama proses iterasi.

3.7.2 Dataset Hasil *Encoding*

Pada tahap ini, dataset payload hasil encoding yang diperoleh dari proses Variational Autoencoder (VAE) dimuat sebagai input untuk pelatihan model CWGAN-GP. Dataset ini berisi representasi numerik berdimensi tinggi dari query SQL Injection berlabel malicious, yang sebelumnya telah melalui serangkaian prapemrosesan seperti tokenisasi, embedding menggunakan FastText, dan encoding menjadi vektor laten oleh VAE.

Dataset terdiri dari dua elemen utama, yaitu encoded_data yang merupakan matriks berisi vektor laten hasil encoding, dan labels yang menyimpan informasi kelas atau tipe dari query SQL Injection tersebut. Dimensi vektor laten ini—biasanya berkisar antara 128 hingga 256—merupakan ukuran ruang abstrak yang cukup untuk menangkap fitur penting struktur query secara efisien. Sedangkan jumlah kelas label unik digunakan untuk mengatur kondisi pada generator dan critic dalam model, memastikan fitur kondisional pada data sintetik yang dihasilkan.

Analisis distribusi kelas dalam dataset penting dilakukan untuk memastikan proporsi sampel antar kelas tetap seimbang. Seimbangnya distribusi kelas ini sangat berpengaruh agar CWGAN-GP dapat menghasilkan data sintetis yang merepresentasikan berbagai tipe serangan SQL Injection secara adil tanpa bias ke kelas mayoritas.

3.7.3 Fungsi Loss dan Gradient Penalty

Pada tahap ini salah satu komponen terpenting dalam implementasi Conditional Wasserstein GAN with Gradient Penalty (CWGAN-GP) adalah mendefinisikan atau membuat fungsi loss serta gradient penalty untuk ditambahkan sebagai bentuk regularisasi atau pembatasan. Hal ini dilakukan agar model tidak berperilaku terlalu ekstrem ketika mempelajari data.

1. Wasserstein Loss

Perbedaan mendasar antara WGAN dan GAN biasa terletak pada fungsi discriminator. Pada GAN klasik, discriminator menghasilkan probabilitas (antara 0–1) dengan aktivasi sigmoid. Namun pada WGAN, discriminator diganti istilahnya menjadi critic, yang menghasilkan skor kontinu (real-valued) tanpa batasan rentang nilai. Skor ini digunakan untuk mengukur seberapa realistis suatu data.

Dengan demikian, generator berusaha memaksimalkan skor data palsu, sementara critic berusaha membedakan keduanya dengan memberi skor lebih tinggi pada data asli. Dalam impelementasi kode, fungsi loss ini cukup sederhana, pada tabel 3.1.

Tabel 3.1 Peseudocode Fungsi Loss

```
Fungsi wasserstein_loss(y_true, y_pred):
    return rata-rata(y_true × y_pred)
```

Pada Tabel 3.1 label diberikan dengan tanda data asli = -1 dan data palsu = +1. Efek akhirnya tetap sesuai dengan prinsip WGAN, yaitu memperbesar jarak skor data asli dan data palsu.

2. Gradient Penalty

Pada WGAN versi awal, *weight clipping* digunakan untuk menjaga syarat *1-Lipschitz continuity* pada critic, namun cara ini sering membuat pelatihan tidak stabil. Untuk memperbaikinya, memperkenalkan metode *gradient penalty* (GP). Ide utamanya adalah memastikan bahwa gradien critic terhadap input selalu memiliki norma mendekati 1. Dengan begitu, critic tidak menghasilkan gradien yang terlalu besar (*exploding gradient*) atau terlalu kecil (*vanishing gradient*). Implementasi sederhana dalam tabel 3.2.

Tabel 3.2 Pseudocode Proses *Gradient Penalty*

```
Fungsi gradient_penalty(model, real, fake, real_labels):

    alpha ← random nilai antara 0 dan 1
    diff ← fake - real
    interpolated ← real + alpha × diff

    Aktifkan GradientTape
        pred ← model(interpolated, real_labels)
        grads ← gradien pred terhadap interpolated

    grads_sqr ← grads²
    grads_sqr_sum ← jumlah(grads_sqr per sampel)
    grad_l2 ← akar(grads_sqr_sum + ε)

    penalty ← rata-rata( (grad_l2 - 1)² )

    return penalty
```

Pada Tabel 3.2 dalam proses perhitungan *gradient penalty*, terlebih dahulu dibangkitkan nilai alpha yang berfungsi sebagai faktor interpolasi acak untuk menentukan posisi titik di antara data asli dan data palsu. Nilai alpha ini kemudian

digunakan untuk menghasilkan interpolated sample, yaitu kombinasi linear antara data asli dan data yang dihasilkan oleh generator. Setelah diperoleh sampel interpolasi tersebut, dilakukan perhitungan gradien keluaran critic terhadap input interpolasi menggunakan *tape.gradient*. Gradien ini digunakan untuk mengukur sensitivitas critic terhadap perubahan kecil pada input. Selanjutnya dihitung nilai penalti, yaitu selisih antara norm gradien dengan nilai 1 yang kemudian dikuadratkan. Tahapan ini bertujuan memastikan norm gradient tetap mendekati 1 sebagaimana disyaratkan oleh WGAN-GP agar proses pelatihan model menjadi lebih stabil.

Dengan cara ini, critic dilatih agar gradiennya tetap stabil. Hasilnya, proses training menjadi lebih seimbang: generator bisa terus belajar menghasilkan data yang makin realistis, dan critic bisa menilai dengan konsisten tanpa overfitting.

3.7.4 Pembangunan Arsitektur Generator Critic dan Training Loop

Pada tahap ini, dilakukan pembangunan dua komponen utama pada arsitektur model Conditional Wasserstein GAN dengan Gradient Penalty (CWGAN-GP), yaitu Generator dan Critic, sekaligus implementasi proses pelatihan (training loop) yang menjadi inti pembelajaran model.

Generator bertugas menghasilkan data payload sintetis yang menyerupai distribusi data asli. Generator menerima dua input diantaranya yaitu vektor noise acak (z) berdimensi 100 yang memberikan variasi random, serta vektor label kondisi berupa one-hot encoding (y) yang mengarahkan jenis data yang dihasilkan. Kedua input digabung dan diproses melalui berlapis jaringan saraf terhubung (fully connected

layers) dengan fungsi aktivasi ReLU pada lapisan tersembunyi. Lapisan output menggunakan aktivasi linear agar nilai keluaran bisa mengambil nilai positif maupun negatif sesuai ruang laten hasil encoding. Output Generator memiliki dimensi yang sama dengan data encoding asli, sehingga bisa diterima Critic sebagai input.

Critic berfungsi membedakan data asli dari data sintetis, dengan input berupa kombinasi data dan label kondisi. Data ini juga diproses dalam beberapa lapisan dense dengan jumlah neuron yang berkurang secara bertahap agar fitur esensial ter-ekstrak sebelum menghasilkan satu skor kontinu sebagai respons. Critic tidak menggunakan aktivasi sigmoid, karena skor real-valued ini diperlukan untuk menghitung fungsi loss Wasserstein yang lebih stabil dibandingkan GAN konvensional.

Proses pelatihan menggunakan training loop di mana Critic diperbarui beberapa kali lebih sering dibanding Generator (misalnya, 3–5 kali critic update per update generator) untuk memastikan kemampuan Critic dalam menilai kualitas data yang dihasilkan Generator. Dalam tiap iterasi:

1. Critic dilatih untuk memaksimalkan perbedaan skor antara data asli dan sintetis sambil mempertahankan syarat Lipschitz melalui penerapan gradient penalty, menjaga kestabilan gradien agar tidak melewati batas kritis.
2. Generator dilatih untuk meminimalkan skor yang dikeluarkan Critic terhadap data sintetis, sehingga secara bertahap meningkatkan kualitas dan realisme data yang dihasilkan.

Fungsi loss yang diterapkan adalah Wasserstein loss, yang menggunakan skor real-valued dari Critic—berbeda dengan fungsi loss GAN klasik yang menggunakan

probabilitas biner. Gradient penalty ditambahkan sebagai regularisasi untuk mengontrol norma gradien Critic agar dekat dengan 1, mencegah pelatihan menjadi tidak stabil.

3.7.5 Generasi Data Sintetis

Setelah proses pelatihan selesai, tahap selanjutnya adalah generasi *data payload SQL Injection sintetik*. Tahap ini menjadi tujuan utama dari penerapan *CWGAN-GP*, yaitu memproduksi dataset tambahan yang realistis dan bervariasi untuk digabungkan dengan dataset header.

Proses generasi dimulai dengan menyiapkan dua jenis input untuk generator. Pertama, noise acak berupa vektor berukuran 100 yang diambil dari distribusi normal standar. Vektor ini berfungsi sebagai representasi laten awal yang akan diubah oleh generator menjadi data baru. Kedua, label kondisi yang menunjukkan kelas payload yang ingin dihasilkan. Dalam penelitian ini, label dipilih dengan mengikuti distribusi asli pada dataset, sehingga proporsi data sintetis tetap mencerminkan kondisi nyata. Sebagai contoh, apabila payload jenis *UNION SELECT* lebih banyak dibandingkan Boolean-based, maka jumlah data hasil generator juga akan mengikuti pola tersebut.

Setelah noise dan label tersedia, keduanya dimasukkan ke dalam generator. Generator kemudian mengubah kombinasi tersebut menjadi data baru berupa vektor berdimensi *[NUM_SYNTHETIC_SAMPLES, encoded_dim]*. Setiap baris pada matriks ini merepresentasikan satu payload sintetis dalam bentuk hasil encoding. Pada

implementasi ini, jumlah data yang dihasilkan ditetapkan sebanyak 10.000 sampel, sesuai dengan parameter *NUM_SYNTHETIC_SAMPLES*.

3.7.6 Decoding Data SQL Injection

Dikarenakan hasil keluaran atau output dari generator pada tahap sebelumnya masih berupa representasi vektor laten dengan dimensi numerik, sehingga belum dapat langsung dibaca sebagai *query SQL Injection*. Oleh karena itu, diperlukan tahap decoding untuk mengubah representasi vektor laten menjadi *payload SQL Injection* dalam bentuk teks yang lebih mudah dianalisis. Proses decoding ini melibatkan beberapa komponen utama, yaitu *VAE decoder*, *fastText embedding* model, dan token mapping, seperti pada proses encoding pada tahapan pra pemrosesan data.

Fungsi *decode_latent_to_query()* menerima input berupa vektor laten dan mengembalikannya dalam bentuk query. Agar hasil tetap terkontrol, digunakan parameter max_len untuk membatasi panjang maksimum token dalam satu query. Selain itu ditambahkan mekanisme anti-looping untuk mencegah pengulangan token yang sama secara berlebihan. Apabila token yang sama muncul lebih dari lima kali berturut-turut, proses decoding pada vektor tersebut dihentikan dan query segera ditutup. Strategi ini penting untuk menghindari hasil sintesis yang tidak bermakna akibat generator menghasilkan embedding berulang.

3.8 Penggabungan *Header* dan *Payload*

Dalam penelitian ini, penggabungan dilakukan dengan menggunakan *Scapy*, sebuah *library Python* yang memungkinkan pembuatan, manipulasi, dan pengiriman paket jaringan secara fleksibel. Proses penggabungan dilakukan sebagai berikut:

3.8.1 Pembuatan *Header* Paket

Data *header* sintetis yang dihasilkan oleh model *CTGAN* digunakan untuk membangun lapisan *IP* dan *TCP/UDP* pada paket jaringan. Seperti atribut *srcaddr, dstaddr, srcport, dstport, prot,* dan *tcp_flags* dimasukkan ke dalam *header* paket menggunakan objek *IP()* dan *TCP()* di *Scapy*.

3.8.2 Penyisipan Payload Query SQL

Payload query SQL sintetis yang dihasilkan oleh model *WGAN* dimasukkan sebagai lapisan *Raw* pada paket. *Payload* ini berisi *query SQL* berbahaya yang telah diproses menjadi *string* yang valid.

3.8.3 Penggabungan *Header* dan *Payload*

Dengan Scapy, *header* dan *payload* digabungkan menggunakan operator / untuk membentuk paket lengkap yang merepresentasikan trafik jaringan dengan serangan *SQL Injection* secara utuh.

3.8.4 Validasi Paket

Scapy secara otomatis menghitung *checksum* dan parameter lain yang diperlukan untuk memastikan paket valid.

3.9 Evaluasi

Evaluasi dalam penelitian ini difokuskan pada penilaian kualitas *dataset* sintetis yang dihasilkan oleh model generatif *CTGAN* untuk *header* paket jaringan dan *WGAN* untuk *payload query SQL*. Metode yang digunakan untuk mengevaluasi kualitas data sintetis header menggunakan metode *Kolmogorov-Smirnov Test* (*KS test)* dan *Jensen-Shannon* Divergence (JSD), sedangkan untuk dataset *payload* menggunakaan *tuning* parameter dan *cosine similarity*.

3.9.1 Kolmogorov-Smirnov Test

Kolmogorov-Smirnov Test (KS test) digunakan untuk menguji apakah distribusi fitur numerik pada *dataset* sintetis memiliki kesamaan yang signifikan dengan distribusi fitur pada data asli. Secara teknis, KS test membandingkan fungsi distribusi kumulatif (CDF) dari dua sampel (data asli dan sintetis) dan menghitung nilai statistik maksimum dari perbedaan antara kedua CDF tersebut.

3.9.2 Jensen-Shannon Divergence

Jensen-Shannon Divergence (JSD) digunakan untuk mengukur kesamaan distribusi probabilitas antara data asli dan sintetis, terutama pada fitur kategorikal atau distribusi multivariant. JSD adalah metrik simetris yang mengkuantifikasi jarak antara dua distribusi probabilitas, dengan nilai 0 menunjukkan distribusi data sintetis sudah mendekati distribusi data asli dan dan nilai lebih besar menandakan distribusi masih jauh atau tidak mirip dengan distribusi data asli.

3.9.3 *Tuning* Parameter

Proses tuning dan evaluasi dilakukan untuk mengoptimalkan performa model CWGAN-GP dengan menyesuaikan beberapa parameter utama, yaitu learning rate untuk generator (glr) dan critic (clr), jumlah update critic terhadap generator dalam satu iterasi (n_critic), serta besarnya bobot gradient penalty (gp). Evaluasi hasil tuning dilakukan menggunakan tiga metrik sebagai ukuran performa model pada ruang laten

.