BAB 3
METODOLOGI PENELITIAN

3.1 Studi Literatur

Pada tahap awal penelitian ini dilakukan studi literatur dengan menelaah dan memahami berbagai sumber referensi yang relevan, seperti artikel ilmiah, jurnal, serta buku yang berkaitan dengan topik *Federated Learning* dan *Homomorphic Encryption*. Studi literatur ini bertujuan untuk memperoleh pemahaman mendalam mengenai metode dan algoritma yang digunakan dalam penelitian, khususnya konsep *Federated Learning* sebagai pendekatan *machine learning* yang dapat melatih beberapa model secara bersamaan tanpa memindahkan data mentah dari lokal ke server, serta mekanisme *Homomorphic Encryption* yang memungkinkan proses komputasi dilakukan langsung pada data terenkripsi untuk menjaga privasi.

Studi literatur ini juga mencakup peninjauan terhadap berbagai penelitian terdahulu yang membahas penerapan *Federated Learning* dalam bidang kesehatan, terutama pada kanker payudara. Melalui analisis terhadap penelitian-penelitian sebelumnya, diperoleh pemahaman mengenai tantangan privasi pada data medis, potensi kebocoran informasi selama proses agregasi model, serta solusi yang ditawarkan melalui integrasi teknik enkripsi seperti *Segmented CKKS Homomorphic Encryption*.

3.2 Arsitektur Model

3.2.1 Spesifikasi Perangkat

Model dalam penelitian ini dijalankan pada perangkat dengan spesifikasi sebagai berikut:

- Sistem Operasi: Ubuntu 22.04.5 LTS (64-bit)
- Memori (RAM): 32 GB
- Prosesor: Intel® Xeon® Silver 4208 CPU @ 2.10 GHz × 16 core
- Kapasitas Penyimpanan: 2 TB
- *Windows*: X11
- Kartu Grafis: llvmpipe (LLVM 15.0.7, 256 bits)

3.2.2 Komponen Utama Model

1. *Server (Global Model)*

   Server berperan sebagai model global. Pada setiap epoch atau pelatihan (training round), server melakukan *aggregation* menggunakan FedAvg kemudian mengirimkan *weights* model global yang sudah di *aggregate* ke seluruh klien. Setelah menerima *weights* yang terenkripsi hasil dari *training* lokal dari setiap klien, server melakukan proses agregasi secara langsung pada ciphertext tanpa mendekripsinya. Hal ini dimungkinkan dengan metode Homomorphic Encryption yang mendukung operasi matematis pada data terenkripsi.

2. *Client (Local Model)*

   Setiap klien diasumsikan sebagai rumah sakit yang memiliki data lokalnya sendiri. Klien menerima *updated weights* global dari server, *training* menggunakan dataset lokal, kemudian mengenkripsi *weights* hasil *training* menggunakan *public key* sebelum mengirimkannya kembali ke server atau *global model* yang memastikan privasi data tetap terjaga karena hanya *weights* terenkripsi yang dikirimkan.

3. *Segmented CKKS Homomorphic Encryption*

   Komponen ini berfungsi untuk melakukan proses enkripsi, dekripsi, serta operasi matematis terhadap parameter model menggunakan metode Segmented CKKS. Berbeda dengan enkripsi CKKS biasa, pendekatan segmented digunakan ketika ukuran tensor melebihi kapasitas slot.

3.2.3 Threat Model

Threat model pada penelitian ini dirancang untuk mendefinisikan batasan keamanan dan asumsi ancaman yang relevan dengan sistem *federated learning* untuk klasifikasi *image* medis. Sistem yang dianalisis terdiri dari beberapa klien yang melakukan pelatihan model secara lokal menggunakan data medis masing-masing, serta sebuah server pusat yang bertugas mengagregasikan parameter model tanpa memiliki akses langsung ke data mentah klien.

Aset utama yang dilindungi dalam sistem ini adalah data *image* medis lokal yang bersifat sensitif, serta informasi yang secara implisit terkandung dalam parameter dan pembaruan model selama proses pelatihan federated. Kebocoran

informasi dari parameter model berpotensi mengungkap karakteristik data medis klien, sehingga perlu dicegah.

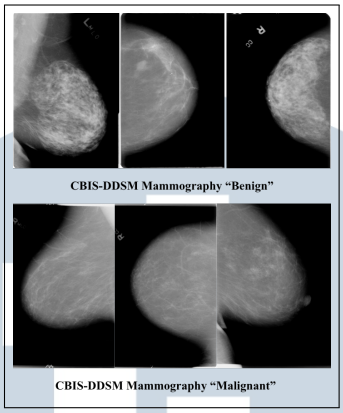
Pada implementasi penelitian ini, skema *Homomorphic Encryption* menggunakan *shared key pair*, di mana satu pasangan *public key* dan *secret key* digunakan bersama oleh seluruh klien. Pendekatan ini mengikuti implementasi FedSHE yang masih bersifat simulatif. Konfigurasi ini, *threat model* yang digunakan adalah *honest-but-curious*, di mana klien dan server diasumsikan menjalankan protokol dengan benar dan tidak melakukan kolusi. Penggunaan *shared secret key* berpotensi menimbulkan risiko kebocoran informasi apabila terjadi kolusi antar klien atau kompromi pada sisi klien. Mekanisme ini tidak dimaksudkan untuk merepresentasikan *deployment federated learning* pada lingkungan produksi, melainkan sebagai simulasi untuk mengevaluasi dampak penggunaan *Homomorphic Encryption* terhadap performa model dan overhead komputasi.

Ancaman yang dipertimbangkan dalam penelitian ini mencakup upaya inferensi data atau kebocoran informasi melalui parameter model yang dikirimkan selama proses agregasi. Pengatasan ancaman tersebut, dilakukan dengan menggunakan skema *homomorphic encryption* berbasis *Segmented* CKKS sehingga parameter model tetap berada dalam bentuk terenkripsi selama transmisi dan agregasi.

3.3 Dataset dan Preprocessing Data

3.3.1 Dataset CBIS-DDSM

Dataset yang digunakan dalam penelitian ini adalah *Curated Breast Imaging Subset of DDSM* (CBIS-DDSM), yang diunduh melalui *Kaggle* [9]. Dataset ini merupakan hasil kurasi dari *Digital Database for Screening Mammography* (DDSM) dan dikembangkan oleh tim *Cancer Imaging Archive* (TCIA). CBIS-DDSM menyediakan *mammography image* digital yang telah diberi anotasi oleh ahli radiologi untuk membantu pengembangan model deteksi kanker payudara berbasis *machine learning*.



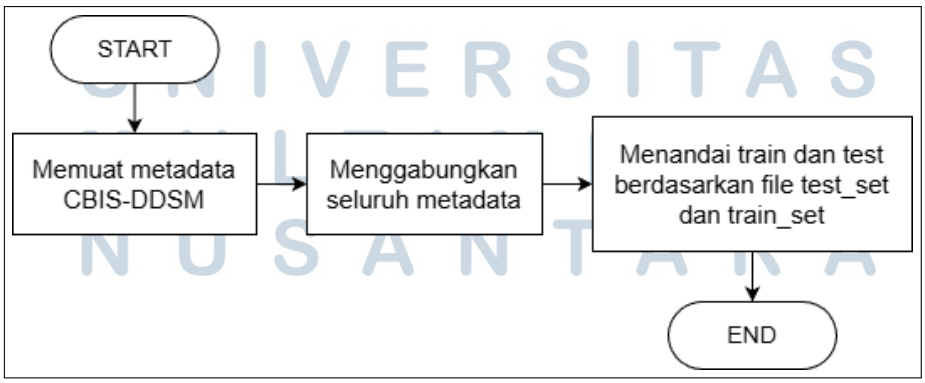
Gambar 3.1. *Image Dataset CBIS-DDSM Mammography*

Gambar 3.1 menunjukan contoh *mammography image* pada dataset CBIS-DDSM. Dataset ini terdiri dari dua kategori utama:

1. Benign (jinak) – menunjukkan adanya kelainan non-kanker pada jaringan payudara.
2. Malignant (ganas) – menunjukkan adanya sel kanker pada jaringan payudara.

Secara keseluruhan, dataset mencakup 3103 *mammography image* dari berbagai pasien. Setiap *image* disertai label diagnosis serta metadata pendukung seperti posisi gambar (kiri/kanan), jenis pandangan (CC/MLO), dan informasi pasien anonim. *Images* yang digunakan memiliki format .jpg, dengan resolusi tinggi yang memungkinkan proses ekstraksi fitur tekstur dan struktur jaringan. *Dataset* ini memiliki 1.739 data yang dikategorikan sebagai "*benign*" dan 1.364 yang dikategorikan sebagai "*malignant*".

3.3.2 Data Collecting

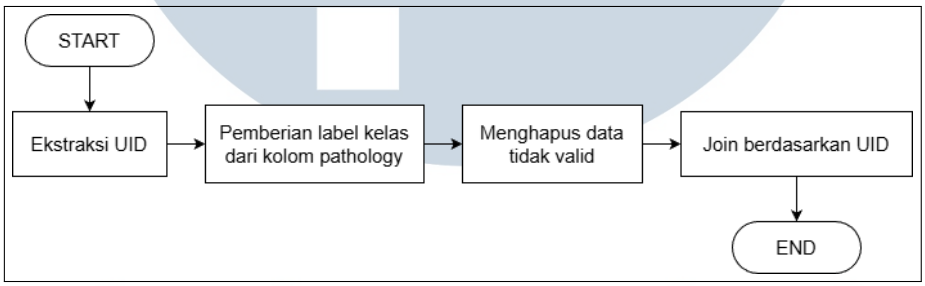


Gambar 3.2. Alur kerja data collecting

Alur kerja *Data Collecting* dapat dilihat pada gambar 3.2 yang diawali dengan memuat metadata dataset CBIS-DDSM yang terdiri dari empat file CSV resmi, yaitu `mass_case_description_train_set.csv`, `mass_case_description_test_set.csv`, `calc_case_description_train_set.csv`, dan `calc_case_description_test_set.csv`. Keempat file ini mencakup data kasus *mass* dan *calcification* pada subset *training* dan *testing*.

Seluruh file CSV digabungkan ke dalam satu *dataframe* terintegrasi, di mana setiap baris merepresentasikan satu *image* mammogram. Kolom *csv_split* ditambahkan untuk menandai pembagian asli dataset, dengan nilai *train* untuk data pelatihan dan *test* untuk data pengujian, guna menjaga *official test split* CBIS-DDSM.

3.3.3 Data Cleaning

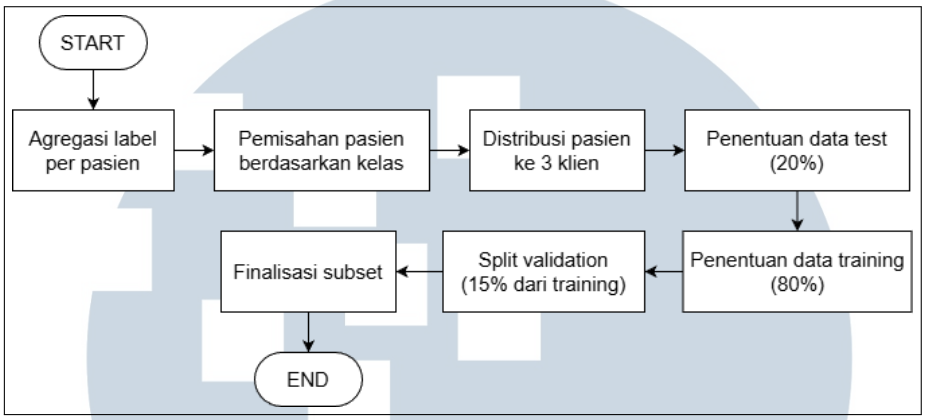


Gambar 3.3. Alur kerja data cleaning

Gambar 3.3 menunjukan tahap *Data Cleaning* diawali dengan ekstraksi `SeriesInstanceUID` dari *path* file DICOM dengan mengambil folder terakhir sebelum ekstensi `.dcm`. UID ini digunakan sebagai kunci utama untuk menghubungkan metadata dengan *image*. Pemberian label kelas dilakukan berdasarkan kolom `pathology`, di mana nilai `MALIGNANT` diklasifikasikan sebagai *malignant* dan nilai lainnya sebagai *benign*.

Data dengan `SeriesInstanceUID` kosong atau label yang tidak tersedia dihapus untuk memastikan validitas dataset. Integrasi *path image* dilakukan dengan memuat file `dicom_info.csv` dan melakukan *join* berdasarkan `SeriesInstanceUID` sehingga diperoleh kolom `image_path`. Hasil tahap ini berupa metadata terstruktur yang mencakup `patient_id`, `label`, `csv_split`, dan `image_path`.

3.3.4 Data Splitting



Gambar 3.4. Alur kerja data splitting

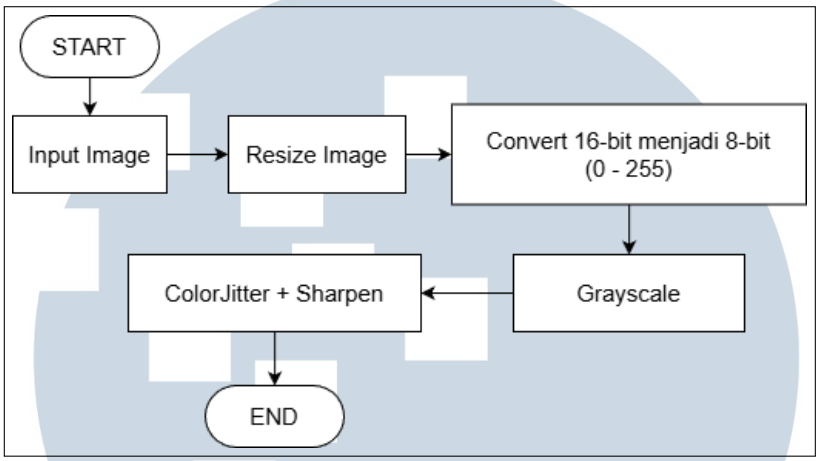
Tahap *Data Splitting* dilakukan berbasis pasien untuk mensimulasikan skenario *Federated Learning* yang dapat dilihat pada gambar 3.4. Label pasien ditentukan dengan mengagregasi seluruh *image* milik pasien yang sama, di mana pasien diklasifikasikan sebagai *malignant* apabila memiliki minimal satu *image malignant*, dan sebagai *benign* apabila seluruh *image*nya *benign*.

Pasien *benign* dan *malignant* diacak secara terpisah dan didistribusikan secara proporsional ke tiga klien, yaitu `site0`, `site1`, dan `site2`, dengan memastikan setiap pasien hanya muncul pada satu klien. Data *testing* mengikuti pembagian asli CBIS-DDSM berdasarkan `csv_split = test` dan tidak diacak ulang. Data *validation* dipilih sebesar 15% pasien dari data pelatihan, sementara sisa pasien digunakan sebagai data *training*, sehingga tidak terjadi *overlap* pasien antar subset. Tabel 3.1 menyajikan jumlah total data dan distribusi kelas *benign* serta *malignant* pada setiap klien sebagai hasil akhir proses *data splitting*.

Tabel 3.1. Distribusi Dataset CBIS-DDSM dan Pembagian Data pada Setiap Klien

Site	Total Data	Benign	Malignant	Train	Val	Test
Site 0	1.025	577	448	696	123	206
Site 1	1.069	587	482	721	128	220
Site 2	1.009	575	434	671	119	219

3.3.5 Image Preprocessing



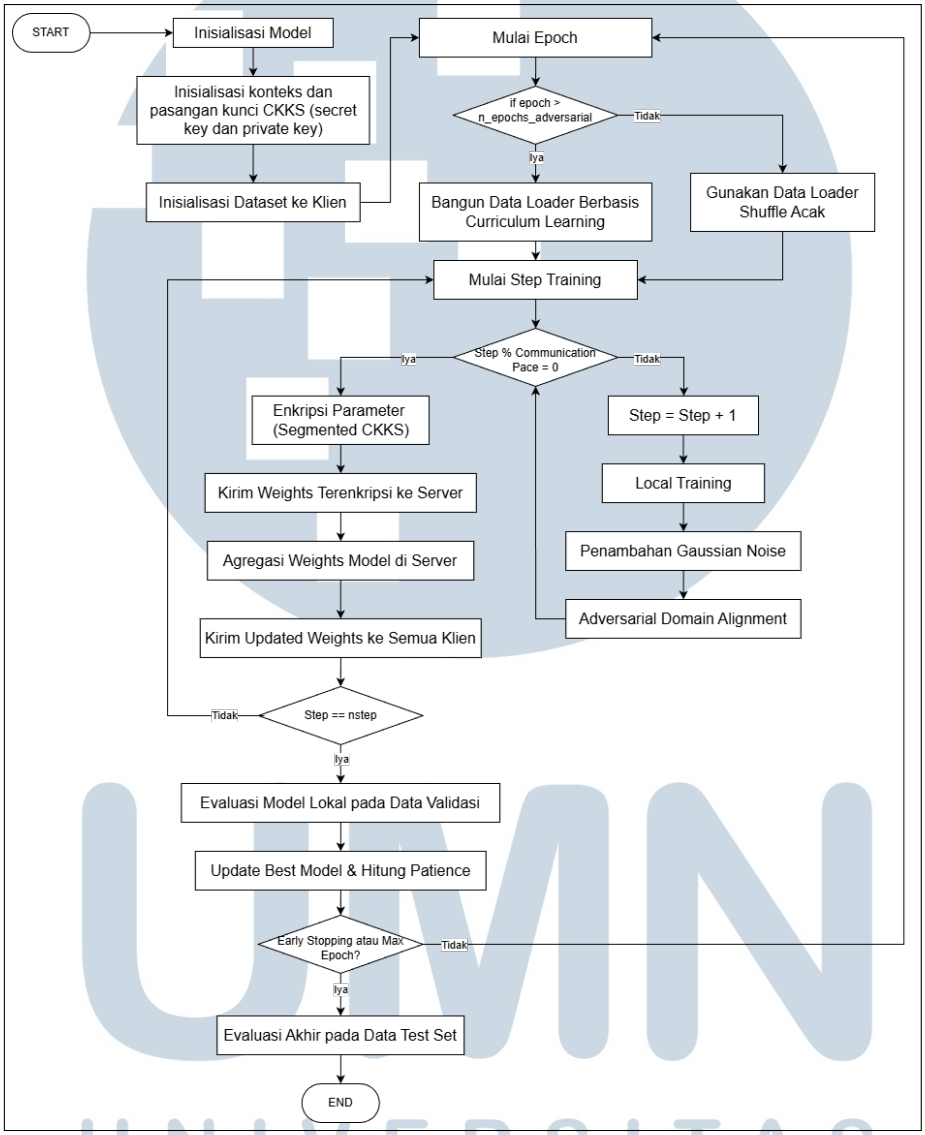
Gambar 3.5. Alur kerja data image preprocessing

Gambar 3.5 menunjukkan tahapan *image preprocessing* yang digunakan dalam penelitian ini. Proses diawali dengan *input image* yang kemudian diubah ukurannya (*resize*) agar sesuai dengan kebutuhan model. Selanjutnya, citra dengan format 16-bit dikonversi menjadi 8-bit dengan rentang nilai piksel 0–255 untuk menyederhanakan representasi data dan menyesuaikannya dengan proses pemrosesan selanjutnya.

Setelah konversi bit-depth, citra diberikan augmentasi berupa *ColorJitter* dan filter *sharpen* untuk meningkatkan variasi data serta mempertegas detail dan tepi penting pada struktur citra. Tahap akhir dilakukan dengan mengubah citra ke format *grayscale* guna mengurangi kompleksitas kanal warna sebelum citra digunakan sebagai masukan ke model.

3.4 Perancangan Model

3.4.1 Model Federated Learning dengan Segmented CKKS HE



Gambar 3.6. Diagram alur proses *Federated Learning* dengan *Segmented* CKKS

Gambar 3.6 memperlihatkan tahapan kerja sistem secara keseluruhan. Proses dimulai dengan inisialisasi model global oleh server, diikuti pembuatan kunci enkripsi oleh KMC, dan pelatihan model di setiap klien menggunakan data lokal. Hasil pelatihan terenkripsi dikirim ke server untuk agregasi *homomorphic*, kemudian hasilnya didistribusikan kembali ke klien untuk proses dekripsi dan pembaruan model. Seluruh proses ini diulang hingga model mencapai konvergensi

atau kondisi *early stopping*.

Secara umum, alur kerja sistem mengikuti skema federated learning berbasis iterasi bertahap (*pace-based aggregation*) di dalam setiap epoch. Proses pelatihan tidak hanya dilakukan satu kali agregasi per epoch, melainkan beberapa kali sesuai dengan parameter *pace*. Tahapan utama sistem dapat dirangkum sebagai berikut:

1. **Inisialisasi Model Global dan Lingkungan Pelatihan**

   Server memulai proses dengan membangun model global menggunakan arsitektur `Classifier`. Selanjutnya, model lokal, *discriminator*, serta optimizer untuk setiap klien diinisialisasi. Dataset lokal masing-masing klien dimuat dan disiapkan dalam bentuk *train*, *validation*, dan *test loader*. Ukuran *mini-batch* pada pelatihan lokal ditentukan secara dinamis berdasarkan jumlah langkah pelatihan (*nsteps*) dalam setiap *epoch*, sehingga satu *epoch* terdiri dari sejumlah iterasi tetap pada setiap klien.

2. **Inisialisasi CKKS**

   Sistem menghasilkan konteks dan pasangan kunci CKKS (*secret key* dan *private key*) dengan parameter keamanan tertentu. Objek enkripsi ini digunakan oleh klien untuk mengenkripsi pembaruan parameter model sebelum dikirim ke server, sementara proses agregasi dilakukan sepenuhnya dalam domain terenkripsi.

3. **Prediksi Awal dan Curriculum Learning**

   Pada awal setiap *epoch*, setiap klien melakukan inferensi terhadap seluruh data pelatihan lokal menggunakan model terkini. Hasil prediksi ini digunakan untuk melacak perubahan performa model terhadap setiap sampel antar *epoch*. Berdasarkan informasi tersebut, mekanisme *curriculum learning* diterapkan dengan menyesuaikan bobot *sampling* data, sehingga sampel yang dianggap lebih sulit atau terlupakan akan lebih sering muncul pada proses pelatihan selanjutnya.

4. **Pelatihan Lokal dan Adversarial Alignment**

   Setiap *epoch* terdiri dari sejumlah iterasi pelatihan lokal (*inner loop*) yang ditentukan oleh parameter *nsteps*. Pada setiap iterasi, klien melakukan pembaruan parameter model menggunakan fungsi kerugian klasifikasi. Selain itu, mekanisme *adversarial alignment* dijalankan dengan

mengekstraksi representasi fitur dan menambahkan gangguan acak berbasis distribusi Gaussian untuk menyelaraskan distribusi fitur antar klien.

5. **Agregasi Federated Berbasis Pace**

   Alih-alih melakukan agregasi satu kali per *epoch*, sistem menerapkan *federated aggregation* secara bertahap berdasarkan parameter *pace*. Setiap sejumlah iterasi tertentu, klien menghitung selisih parameter model terhadap model global terakhir. Pembaruan ini kemudian dikirim ke server untuk dilakukan agregasi *federated*. Pembaruan parameter dienkripsi terlebih dahulu menggunakan skema *Segmented* CKKS sebelum dikirim ke server. Server melakukan operasi penjumlahan dan penskalaan parameter secara homomorfik tanpa melakukan dekripsi.

6. **Dekripsi dan Sinkronisasi Model**

   Hasil agregasi global kemudian didekripsi oleh klien yang memiliki kunci rahasia, dan parameter model global yang telah diperbarui didistribusikan kembali ke seluruh klien. Model lokal kemudian disinkronkan dengan model global terbaru sebelum melanjutkan iterasi pelatihan berikutnya.

7. **Evaluasi, Early Stopping, dan Pengujian Akhir**

   Pada akhir setiap *epoch*, model global dievaluasi menggunakan data validasi dari seluruh klien. Metrik performa dan waktu komputasi dicatat untuk keperluan analisis. Proses pelatihan dihentikan lebih awal apabila kriteria *early stopping* terpenuhi. Setelah pelatihan selesai, model global terbaik dievaluasi kembali pada data tes untuk memperoleh nilai akhir AUC, PR-AUC, dan akurasi.

3.4.2 Perancangan Arsitektur Model

Pada penelitian ini, perancangan model difokuskan pada integrasi mekanisme pembelajaran terdistribusi, penyelarasan fitur antar klien, serta perlindungan privasi parameter model dalam skema *federated learning*. Arsitektur model yang digunakan mengadopsi pendekatan dari Jiménez-Sánchez et al. untuk *adversarial domain alignment* dan *curriculum learning*, serta skema enkripsi homomorfik berbasis *Segmented CKKS* dari Pan et al..

Model dilatih secara terdistribusi pada setiap klien menggunakan data lokal tanpa membagikan data mentah ke server. Fokus utama penelitian ini bukan

pada pengembangan arsitektur jaringan saraf baru, melainkan pada pengaturan alur pelatihan, mekanisme *sampling* data, dan pengamanan parameter model yang dikirimkan ke server.

3.4.3 Encoder dan Classifier

Proses ekstraksi fitur *image mammography* dilakukan menggunakan sebuah *encoder* berbasis arsitektur *ResNet22 (ViewResNetV2)* yang diadopsi dari Sánchez et al. Encoder ini dirancang untuk memproses *image grayscale mammography* beresolusi tinggi dan menghasilkan representasi fitur yang bersifat informatif serta stabil untuk proses klasifikasi.

Setiap *image mammogprahpy* terdiri dari empat *view*, yaitu L-CC, L-MLO, R-CC, dan R-MLO. Keempat *view* tersebut diproses menggunakan *encoder* dengan bobot yang sama (*shared weights*). Representasi fitur yang dihasilkan dari masing-masing *view* kemudian digabungkan menggunakan mekanisme *average pooling* untuk membentuk satu vektor fitur global berdimensi 256.

Encoder ResNet22 terdiri dari lima lapisan residual dengan konfigurasi jumlah blok [2, 2, 2, 2, 2], serta jumlah filter yang meningkat secara bertahap dari 16 hingga 256. Setiap blok residual dilengkapi dengan *Batch Normalization* dan fungsi aktivasi ReLU untuk menjaga stabilitas pelatihan dan mempercepat konvergensi model.

Vektor fitur hasil encoder selanjutnya diberikan sebagai masukan ke *classifier* berbasis *Multi-Layer Perceptron (MLP)*. *Classifier* terdiri dari beberapa lapisan *fully connected* yang diikuti oleh aktivasi ReLU, *Batch Normalization*, dan *Dropout* dengan rasio 0,2 untuk mengurangi risiko *overfitting*. *Classifier* berfungsi untuk menghasilkan prediksi kelas akhir pada setiap klien selama proses pelatihan lokal.

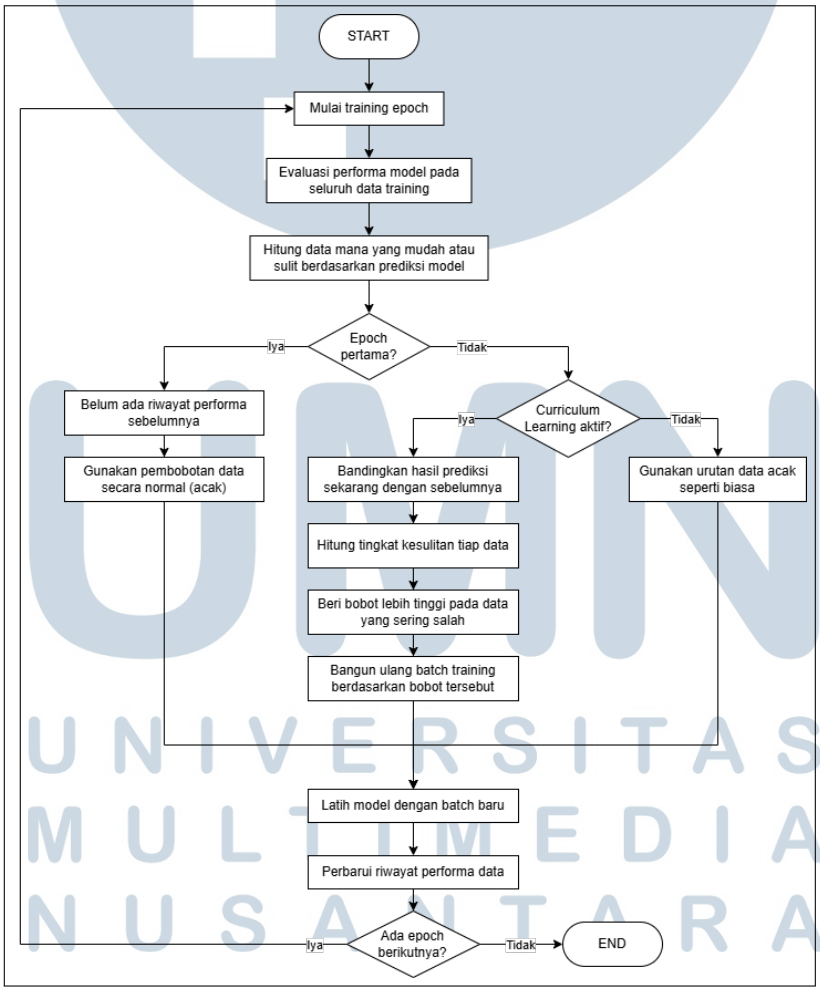
3.4.4 Curriculum Learning Berbasis Kesulitan Sampel

Penelitian ini menerapkan mekanisme *curriculum learning* untuk mengatur urutan dan frekuensi kemunculan sampel data selama proses pelatihan lokal di setiap klien. Pendekatan ini bertujuan untuk membantu model mempelajari data secara bertahap, dimulai dari sampel yang lebih mudah hingga sampel yang memiliki tingkat kesulitan lebih tinggi.

Tingkat kesulitan setiap sampel ditentukan berdasarkan konsistensi prediksi

model pada beberapa iterasi pelatihan sebelumnya. Sampel yang sering mengalami kesalahan prediksi akan diberikan bobot yang lebih besar, sehingga memiliki probabilitas lebih tinggi untuk dipilih kembali pada iterasi pelatihan selanjutnya. Sebaliknya, sampel yang telah dipelajari dengan baik akan tetap disertakan, namun dengan frekuensi yang lebih rendah.

Bobot sampel ini dihitung menggunakan fungsi pembobotan khusus dan diimplementasikan melalui *WeightedRandomSampler*. Dengan mekanisme ini, seluruh data pelatihan tetap digunakan, namun distribusi kemunculan sampel disesuaikan dengan tingkat kesulitannya. Mekanisme curriculum learning ini mulai diterapkan setelah fase awal pelatihan untuk memastikan bahwa model telah memiliki representasi fitur yang cukup stabil sebelum diberikan penekanan pada sampel yang sulit. Alur kerja curriculum learning ditunjukkan pada Gambar 3.7.



Gambar 3.7. Alur kerja Curriculum Learning

3.4.5 Adversarial Domain Alignment

Penelitian ini mengadopsi mekanisme *adversarial domain alignment* untuk mengatasi perbedaan distribusi data antar klien (*domain shift*). Pada setiap klien, digunakan sebuah *discriminator* yang bertugas membedakan representasi fitur berdasarkan asal domain klien. Pada mekanisme ini, *encoder* berperan sebagai *generator* yang berusaha menghasilkan representasi fitur yang tidak dapat dibedakan antar domain, sedangkan discriminator berusaha mengklasifikasikan domain asal fitur tersebut.

Proses ini membentuk skema pelatihan *adversarial* yang mendorong *encoder* untuk menghasilkan representasi fitur yang bersifat *domain-invariant*. *Gaussian noise* ditambahkan untuk meningkatkan stabilitas pelatihan *adversarial*. Pendekatan ini bertujuan untuk meningkatkan kemampuan generalisasi model global terhadap data yang berasal dari klien dengan distribusi yang berbeda.

3.4.6 Federated Learning Setup

Konfigurasi *federated learning* pada penelitian ini dirancang untuk mengatur mekanisme pelatihan terdistribusi, frekuensi komunikasi antar klien dan server, serta strategi pengendalian konvergensi model. Seluruh proses pelatihan dilakukan tanpa pertukaran data mentah antar klien, sehingga privasi data tetap terjaga. Parameter yang digunakan dalam skema *federated learning* ditunjukkan pada Tabel 3.2.

Tabel 3.2. Konfigurasi parameter *federated learning*

Kategori	**Parameter**
Federated Learning	Jumlah klien: 3
	Jumlah langkah komunikasi (*nsteps*): 120
	Frekuensi sinkronisasi model (*pace*): 40
	Aktivasi *adversarial alignment*: setelah 10 *epoch*
	Penggunaan *curriculum learning*: `True`
	Patience untuk *early stopping*: 15

Pelatihan *federated learning* pada penelitian ini menggunakan skema sinkronisasi berbasis *pace*. Pada setiap klien, model dilatih secara lokal selama sejumlah langkah pelatihan tertentu sebelum dilakukan komunikasi dengan server.

Nilai *pace* ditetapkan sebesar 40, yang berarti setiap klien mengirimkan pembaruan parameter model ke server setelah menyelesaikan 40 langkah pelatihan lokal.

Jumlah total langkah komunikasi global (*nsteps*) pada penelitian ini adalah 120. Konfigurasi tersebut menghasilkan tiga kali proses sinkronisasi model global dalam satu *epoch*. Skema ini memungkinkan proses agregasi dilakukan secara periodik tanpa menunggu seluruh *epoch* selesai, sehingga efisiensi komunikasi antara klien dan server dapat ditingkatkan.

Selama proses pelatihan lokal, setiap klien menerapkan mekanisme *curriculum learning* untuk mengatur distribusi kemunculan sampel pelatihan. Sampel dengan tingkat kesulitan yang lebih tinggi diberikan bobot lebih besar sehingga lebih sering muncul dalam proses pelatihan. Pendekatan ini membantu model belajar secara bertahap dan menjaga stabilitas proses optimisasi pada lingkungan *federated learning*.

Mekanisme *adversarial domain alignment* diaktifkan setelah model melalui fase pelatihan awal selama 10 *epoch*. Penundaan aktivasi ini bertujuan untuk memastikan bahwa *encoder* telah menghasilkan representasi fitur yang cukup stabil. Setelah diaktifkan, proses *adversarial alignment* digunakan untuk menyelaraskan distribusi fitur antar klien agar representasi yang dihasilkan menjadi lebih konsisten.

Mekanisme *early stopping* diterapkan di sisi server untuk mengendalikan konvergensi model global. Evaluasi performa model dilakukan menggunakan data validasi pada setiap akhir *epoch*. Proses pelatihan akan dihentikan apabila nilai *validation loss* tidak mengalami perbaikan selama sejumlah *epoch* berturut-turut. Nilai *patience* pada mekanisme *early stopping* ditetapkan sebesar 15 *epoch*. Apabila tidak terjadi penurunan *validation loss* selama 15 *epoch* berturut-turut, proses pelatihan dihentikan secara otomatis. Model global dengan performa validasi terbaik kemudian disimpan dan digunakan pada tahap evaluasi data uji.

3.4.7 Parameter Homomorphic Encryption (HE)

Penelitian ini mengadopsi skema *Homomorphic Encryption* CKKS dengan pendekatan *segmented encryption* sebagaimana diperkenalkan oleh Pan et al. pada metode FedSHE. Skema ini digunakan untuk melindungi parameter model yang dikirimkan dari klien ke server selama proses federated learning. Implementasi CKKS difokuskan pada pengamanan bobot model tanpa melakukan operasi pelatihan langsung pada data terenkripsi.

Pada sisi klien, parameter model hasil pelatihan lokal terlebih dahulu

diratakan menjadi vektor satu dimensi, kemudian dibagi ke dalam beberapa segmen agar sesuai dengan kapasitas slot pada skema CKKS. Setiap segmen dienkripsi menggunakan kunci publik sebelum dikirimkan ke server. Server hanya melakukan agregasi terhadap parameter yang telah terenkripsi tanpa memiliki akses terhadap nilai asli parameter tersebut.

Parameter CKKS yang digunakan dalam penelitian ini disesuaikan dengan kebutuhan komputasi *federated learning* dan batasan kedalaman operasi aritmatika yang diperlukan. Pemilihan parameter mengikuti konfigurasi yang direkomendasikan dalam FedSHE untuk menjaga keseimbangan antara tingkat keamanan, ketelitian numerik, dan efisiensi komputasi. Daftar parameter CKKS yang digunakan ditampilkan pada Tabel 3.3.

Tabel 3.3. Parameter CKKS yang Digunakan dalam Model

Parameter	**Deskripsi**	**Contoh Nilai**
`poly_modulus_degree`	Menentukan jumlah slot atau kapasitas enkripsi per *ciphertext*. Semakin besar nilai ini, semakin tinggi tingkat keamanan namun juga meningkatkan biaya komputasi.	`8192`, `16384`, `32768`
`scale_factor`	Faktor skala untuk merepresentasikan bilangan real. Semakin besar nilainya, ketelitian meningkat namun noise bertambah lebih cepat.	2^{20}
`security_level`	Level keamanan terhadap serangan kriptografi (128-bit atau 192-bit).	`128` dan `192`
`multiplicative_depth`	Jumlah maksimum operasi perkalian homomorfik sebelum *noise* melebihi batas dekripsi.	`3`

Lanjutan ke halaman berikutnya

Tabel 3.3 Parameter CKKS yang Digunakan dalam Model (lanjutan)

Parameter	Deskripsi	Contoh Nilai
`context`	Konteks HE yang menyimpan seluruh parameter di atas dan dibagikan antara server serta klien.	Disimpan pada `ckks_context.con`
`public_key` / `secret_key`	Kunci publik digunakan untuk enkripsi (dibagikan), sedangkan kunci rahasia hanya disimpan di sisi klien.	`ckks_pub.key`, `ckks_secret.key`

Pada penelitian ini, pasangan kunci CKKS (*public key* dan *secret key*) dihasilkan satu kali pada awal pelatihan dan dibagikan secara identik kepada seluruh klien. *Public key* digunakan untuk proses enkripsi pembaruan parameter sebelum dikirim ke server, sedangkan *secret key* digunakan oleh klien untuk melakukan dekripsi hasil agregasi global. Server tidak memiliki akses terhadap *secret key* dan hanya melakukan operasi agregasi pada ciphertext.

3.5 Implementasi Model

3.5.1 Inisialisasi Model dan CKKS

Lingkungan pelatihan dibangun dalam skenario *Federated Learning* dengan skema *Homomorphic Encryption CKKS*. Pada tahap awal, sistem melakukan inisialisasi konteks dan pasangan kunci (*secret key* dan *public key*) CKKS yang digunakan sepanjang proses komunikasi. Konteks enkripsi ini dikelola oleh komponen manajemen kunci dan direferensikan oleh seluruh klien selama proses pelatihan. Proses inisialisasi konteks dan kunci CKKS tercatat pada log konsol seperti ditunjukkan pada Gambar 3.8.

```
[CKKS] Generating CKKS keys/context (depth-adaptive, FedSHE-style)...
ckks_params:  {'scheme': 'CKKS', 'n': 8192, 'scale': 1048576, 'qi_sizes': [30, 20, 20, 20, 30], 'sec': 128}
success: valid
[CKKS] Key generation done. KMC holds HE (context+keys).
```

Gambar 3.8. Log inisialisasi konteks dan kunci *CKKS* pada awal pelatihan.

Dataset kemudian dimuat dalam bentuk terdistribusi ke beberapa klien menggunakan fungsi `build_sites()` yang dapat dilihat pada *snippet* kodingan 3.1. Fungsi tersebut menghasilkan *train set*, *validation loader*, dan *test loader* untuk

setiap klien. Pembagian dataset ini memungkinkan proses pelatihan lokal dilakukan secara independen pada masing-masing klien. Distribusi jumlah data pada setiap klien ditunjukkan pada Gambar 3.9.

```
trainsets, val_loaders, test_loaders, device = build_sites()
```

Kode 3.1: Snippet build data train, val, dan test client

```
[DATA] site0 (idx=0): train=696 val=123 test=206
[DATA] site1 (idx=1): train=721 val=128 test=220
[DATA] site2 (idx=2): train=671 val=119 test=219
loading pretrained weights
```

Gambar 3.9. Informasi jumlah data *train*, *validation*, dan *test* pada setiap klien.

Model global dikonstruksi menggunakan arsitektur *Classifier*. Setiap klien memperoleh model lokal dengan arsitektur yang identik, serta sebuah *Discriminator* yang digunakan dalam proses *adversarial alignment*. Seluruh model diinisialisasi pada perangkat pelatihan yang sama untuk menjaga konsistensi komputasi. Optimizer Adam digunakan untuk mengoptimasi parameter model klasifikasi, encoder, dan discriminator, sedangkan fungsi loss yang digunakan adalah *CrossEntropyLoss*.

Setiap klien kemudian dibungkus dalam objek *Client* yang menyimpan model lokal, discriminator, dataset pelatihan, serta referensi konteks CKKS. Dengan demikian, setiap klien mampu melakukan pelatihan lokal sekaligus proses enkripsi dan dekripsi parameter secara mandiri selama siklus *Federated Learning* berlangsung.

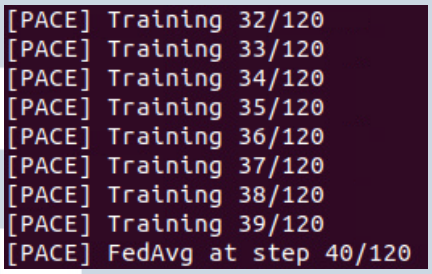
Potongan kode singkat yang merepresentasikan inisialisasi konteks CKKS, model global, dan klien ditunjukkan pada kodingan 3.2.

```
if getattr(params, "mode", "Plain") == "CKKS":
        print("[CKKS] Generating CKKS keys/context (depth-adaptive, FedSHE-style)...")
        HE = generate_ckks_key(params.ckks_sec_level, params.ckks_mul_depth, params.ckks_key_len)
        print("[CKKS] Key generation done. KMC holds HE (context+keys).")
```

Kode 3.2: Snippet inisialisasi CKKS, model global, dan klien.

3.5.2 Pelatihan Lokal pada Setiap Klien

Setiap klien menerima model global dan dataset lokal yang telah dibagi menjadi data latih, validasi, dan uji. Pelatihan dilakukan langkah demi langkah (*step*) dengan pembaruan parameter menggunakan *CrossEntropy Loss*. Sinkronisasi ke server mengikuti skema *pace*, yaitu setiap 40 langkah pelatihan lokal dilakukan agregasi *FedAvg* (Gambar 3.10).



Gambar 3.10. Log pelatihan lokal berbasis *pace* dan pemicu agregasi FedAvg.

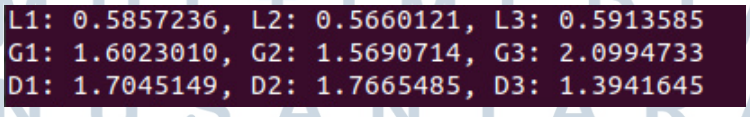
Mekanisme *adversarial domain alignment* aktif secara *forward* sejak awal, namun perhitungan *loss* dan pembaruan parameter baru dilakukan setelah epoch ke-10 (*n_epochs_adversarial*). Potongan kode implementasinya sebagai berikut:

```
if epoch >= params.n_epochs_adversarial:
        lossG.backward(retain_graph=True)
        optimizerGs[i].step()
        optimizerGs[j].step()

        lossD.backward(retain_graph=True)
        optimizerDs[i].step()
```

Kode 3.3: Implementasi mekanisme adversarial alignment

Setelah *adversarial* aktif, klien menerapkan *curriculum learning*, menyesuaikan distribusi kemunculan sampel berdasarkan tingkat kesulitan. Perubahan ini tercatat pada log pemuatan data selama pelatihan (Gambar 3.11).



Gambar 3.11. Log nilai *generator loss* dan *discriminator loss* setelah aktivasi *adversarial* domain alignment pada epoch ke-10.

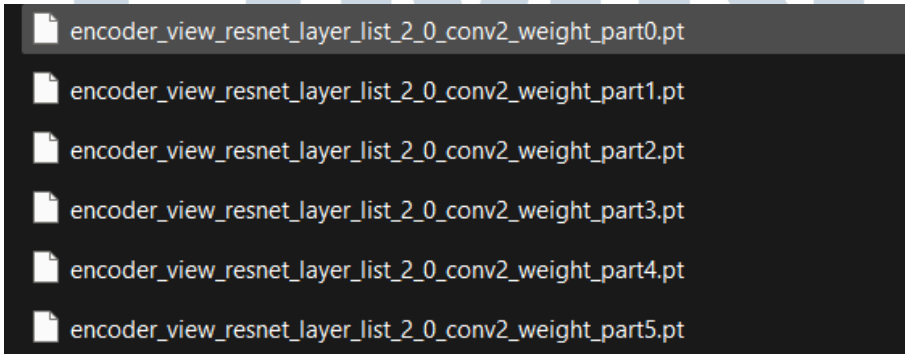
3.5.3 Enkripsi Parameter dengan Skema Segmented CKKS

Pada tahap ini, setiap klien mengenkripsi *update* parameter model menggunakan skema homomorfik CKKS secara *segmented*. Segmentasi diperlukan karena ukuran vektor parameter model *deep learning* bisa melebihi kapasitas slot CKKS dalam satu *ciphertext*. Prosesnya adalah sebagai berikut:

1. Hitung *update* parameter sebagai selisih antara bobot terbaru dan bobot sebelumnya.
2. Tentukan panjang vektor *update*.
    - Jika vektor muat dalam satu *ciphertext*, lakukan enkripsi langsung.
    - Jika vektor lebih besar dari kapasitas slot, pecah menjadi beberapa segmen dan enkripsi setiap segmen secara terpisah.
3. Hasil enkripsi berupa satu atau beberapa *ciphertext* per layer, siap dikirim ke server untuk agregasi.

```
Layer encoder.view_resnet.layer_list.2.0.conv1.weight | elements: 18432 | parts: 5
Layer encoder.view_resnet.layer_list.2.0.bn2.weight | elements: 64 | parts: 1
Layer encoder.view_resnet.layer_list.2.0.bn2.bias | elements: 64 | parts: 1
Layer encoder.view_resnet.layer_list.2.0.bn2.running_mean | elements: 64 | parts: 1
Layer encoder.view_resnet.layer_list.2.0.bn2.running_var | elements: 64 | parts: 1
Layer encoder.view_resnet.layer_list.2.0.bn2.num_batches_tracked | elements: 1 | parts: 1
Layer encoder.view_resnet.layer_list.2.0.conv2.weight | elements: 36864 | parts: 9
Layer encoder.view_resnet.layer_list.2.0.downsample.0.weight | elements: 2048 | parts: 1
Layer encoder.view_resnet.layer_list.2.1.bn1.weight | elements: 64 | parts: 1
```

Gambar 3.12. Contoh hasil *print* terminal segmentasi layer pada proses enkripsi CKKS



Gambar 3.13. Contoh hasil segmentasi layer pada proses enkripsi CKKS

Contoh output proses ini ditunjukkan pada Gambar 3.13 dan Gambar 3.12, di mana layer `encoder_view_resnet_layer_list_2_0_conv2_weight` dibagi menjadi beberapa bagian karena ukurannya melebihi kapasitas satu *ciphertext*.

3.5.4 Agregasi Parameter Terenkripsi di Server

Server menerima parameter terenkripsi (*update*) dari seluruh klien. Agregasi dilakukan langsung pada ciphertext menggunakan operasi penjumlahan homomorfik CKKS. Server menangani dua kasus:

- Parameter dalam satu *ciphertext* per layer
- Parameter yang disegmentasi menjadi beberapa bagian

Agregasi dilakukan per-layer dan per-segmen, sehingga struktur model tetap konsisten. Setelah agregasi selesai, server menyimpan model global terenkripsi ke file, siap dikirim kembali ke klien. Contoh pemanggilan fungsi agregasi (*safe snippet*):

```
update_w_avg, agg_time = aggregate_encrypted(
    clients_ciphertexts_list)
print("[CKKS] Server finished homomorphic aggregation.")
```

Kode 3.4: Pemanggilan fungsi agregasi homomorfik CKKS di server.

Contoh hasil proses agregasi dari terminal ditunjukkan pada Gambar 3.14, termasuk waktu agregasi dan status siapnya tiap *site*.

```
[SERVER] Site 0 ready!
[SERVER] Site 1 ready!
[SERVER] Site 2 ready!
[CKKS][SERVER] Aggregation time: 0.6763s
[CKKS] Server finished homomorphic aggregation.
[CKKS][SERVER] Saved encrypted global model → ./comm/ckks/enc_global_model_step120.pt
```

Gambar 3.14. Contoh log terminal hasil agregasi homomorfik CKKS di server.

3.5.5 Dekripsi dan Rekonstruksi Parameter Global

Setelah server mengirimkan pembaruan model terenkripsi, setiap klien melakukan proses *decrypt-and-apply*. Tahap ini mencakup:

1. Mendekripsi parameter global, termasuk menangani parameter yang disegmentasi.
2. Mengembalikan bentuk *tensor* asli sesuai layer model.
3. Menggabungkan pembaruan global dengan parameter lokal.

Tahap dekripsi juga memperhatikan kondisi khusus: parameter berjenis bilangan bulat tidak dienkripsi, dan layer seperti `running_mean` atau `running_var` dipertahankan dari model lokal tanpa diganti. Contoh pemanggilan fungsi dekripsi pada klien ditunjukkan di bawah ini:

```
model_sd = clients[0].decrypt_global(HE, update_w_avg, n_sites)
```

Kode 3.5: Pemanggilan fungsi dekripsi dan rekonstruksi parameter global pada sisi klien.

Hasil proses dekripsi dapat dilihat pada log terminal, termasuk tipe data sebelum dan sesudah dekripsi, ukuran vektor, waktu dekripsi, serta lokasi *file* global model yang tersimpan (Gambar 3.15):

```
[CKKS] Before decrypt: <class 'Pyfhel.PyCtxt.PyCtxt'>
[CKKS] After decrypt : <class 'numpy.ndarray'> 64
[CKKS] Before decrypt: <class 'Pyfhel.PyCtxt.PyCtxt'>
[CKKS] After decrypt : <class 'numpy.ndarray'> 64
[CKKS][SITE0] Pure decryption time: 2.7690s
[CKKS][CLIENT0] Saved decrypted global model → ./comm/ckks/dec_global_model_step120.pt
```

Gambar 3.15. Contoh log terminal hasil dekripsi dan rekonstruksi parameter global di sisi klien.

3.5.6 Evaluasi Model dan Early Stopping

Evaluasi performa model dilakukan selama pelatihan dengan menggunakan *validation set* di setiap *epoch*. Pada setiap *epoch*, metrik seperti *loss* dan akurasi dihitung untuk memantau kemajuan pelatihan. Mekanisme *early stopping* diterapkan untuk menghentikan pelatihan secara otomatis ketika tidak terjadi peningkatan performa pada *validation loss* selama sejumlah *epoch* tertentu (*patience*).

Proses *early stopping* dihitung melalui *patience counter*, di mana jika nilai *loss* tidak membaik, *counter* bertambah. Ketika *counter* mencapai nilai *patience* yang ditetapkan, pelatihan dihentikan dan model terbaik berdasarkan *validation loss* disimpan sebagai model akhir. Hasil mekanisme *early stopping* dapat dilihat pada Gambar 3.16. Contoh pemanggilan mekanisme *early stopping* pada server:

```
if patience_counter >= params.patience:
    print(f"[SERVER]        Early stopping at epoch #{epoch}. Best
     val_loss={best_val_loss:.4f}")
    break
```

Kode 3.6: Pemantauan early stopping berdasarkan *validation loss* pada server.

Gambar 3.16. Contoh hasil *print* terminal mekanisme *early stopping*

Setelah pelatihan selesai, model global terbaik dievaluasi menggunakan *test set* pada masing-masing *site*. Proses ini menghasilkan metrik performa seperti akurasi, AUC, dan PR-AUC. Hasil pengukuran kemudian dirata-ratakan untuk memperoleh performa keseluruhan dari seluruh *site*. Contoh pemanggilan evaluasi akhir pada *site*:

```
acc, roc_auc, pr_auc, cm, filenames = final_test(global_model,
    test_loader, device, dataset_obj)
```

Kode 3.7: Evaluasi akhir model global menggunakan *test set* pada masing-masing site.

Rata-rata performa keseluruhan dihitung dengan mengekstrak metrik dari semua *site*:

```
avg_acc = float(np.mean(all_acc))
avg_roc = float(np.mean(all_roc))
avg_pr  = float(np.mean(all_pr))
print(f"[TEST] === AVERAGE OVER SITES === ACC={avg_acc:.4f}, AUC={
    avg_roc:.4f}, PR-AUC={avg_pr:.4f}")
```

Kode 3.8: Penghitungan rata-rata performa akhir dari seluruh site.

Sebagai bukti implementasi, dapat disertakan *screenshot* log terminal yang menunjukkan:

- Proses *early stopping* (*epoch* berhenti dan nilai *validation loss* terbaik)
- Evaluasi *test* set akhir per *site*
- Nilai rata-rata metrik performa keseluruhan