BAB 3
METODOLOGI PENELITIAN

Metodologi penelitian disusun untuk mengevaluasi dampak penerapan *Packed CKKS Homomorphic Encryption* pada sistem *Federated Learning* untuk klasifikasi kanker payudara. Fokus evaluasi mencakup tiga aspek, yaitu performa model (AUC dan PR–AUC), efisiensi waktu komputasi (pelatihan lokal, agregasi, enkripsi, dan dekripsi), serta biaya komunikasi (*data transfer volume*).

Tahapan penelitian mencakup studi literatur, pengumpulan dataset CBIS-DDSM, pra-pemrosesan data, pembagian data berbasis `patient_id` ke dalam subset *train*, *validation*, dan *test*, serta distribusi data *train* ke tiga klien untuk mencegah kebocoran data antar subset maupun lintas klien. Selanjutnya, diterapkan *data transformation* yang meliputi perubahan ukuran citra, normalisasi intensitas, dan peningkatan kontras menggunakan CLAHE sebelum data dimuat ke dalam model. Pelatihan *Federated Learning* dilakukan menggunakan FedAvg dengan pengaturan komunikasi berbasis *pace*, diintegrasikan dengan CKKS pada proses komunikasi dan agregasi, serta *top-k sparsification* untuk mengurangi jumlah pembaruan model yang dikirim pada setiap ronde. Evaluasi dilakukan pada mode *plain* dan mode terenkripsi dengan variasi *top-k* untuk menganalisis *trade-off* antara privasi, performa, dan overhead sistem.

3.1 Studi Literatur

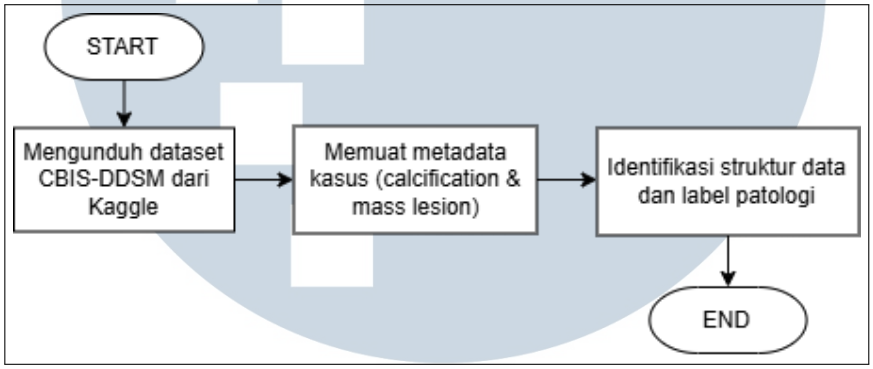
Studi literatur dilakukan untuk memperoleh pemahaman yang komprehensif mengenai konsep dan perkembangan terkini terkait *Federated Learning* (FL) dan *Homomorphic Encryption* (HE) yang menjadi dasar penelitian ini. Tahapan ini bertujuan untuk mengidentifikasi metode, algoritma, serta pendekatan yang relevan dalam mendukung proses analisis dan perancangan sistem.

Proses studi literatur dilakukan melalui penelusuran berbagai sumber ilmiah, seperti jurnal internasional, prosiding konferensi, serta artikel akademik yang diperoleh dari basis data Scopus, Google Scholar, IEEE Xplore, SpringerLink, dan ScienceDirect. Pencarian dilakukan menggunakan kata kunci seperti “Federated Learning”, “Homomorphic Encryption”, “Packed HE”, dan “Breast Cancer Classification”. Hasil penelusuran digunakan untuk memahami perkembangan penelitian terdahulu, menemukan kesenjangan penelitian, serta menentukan

pendekatan yang paling sesuai untuk diterapkan dalam penelitian ini.

Tahapan ini juga berperan dalam menentukan arah eksperimen yang dilakukan, termasuk pemilihan algoritma FedAvg sebagai metode agregasi model, penggunaan skema *Packed Homomorphic Encryption* (PHE) untuk perlindungan privasi, serta penerapan model pada studi kasus klasifikasi kanker payudara. Hasil kajian literatur menjadi dasar dalam perancangan sistem dan rancangan eksperimen yang dijelaskan pada bagian selanjutnya.

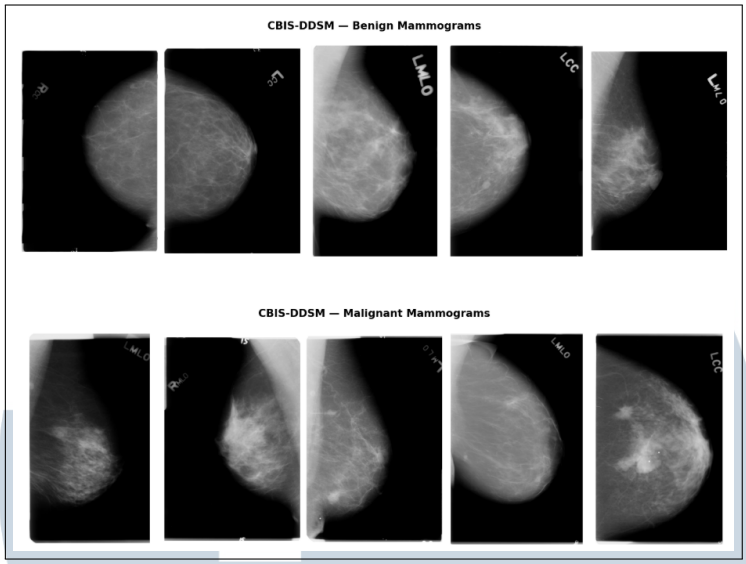
3.2 Pengumpulan Data



Gambar 3.1. Flowchart pengumpulan data

Alur pada Gambar 3.1 merangkum proses pengambilan dataset CBIS-DDSM dan pemuatan metadata awal yang digunakan dalam penelitian ini. Data yang digunakan dalam penelitian ini berasal dari *Curated Breast Imaging Subset of Digital Database for Screening Mammography* (CBIS-DDSM), yang merupakan hasil kurasi dari *Digital Database for Screening Mammography* (DDSM) dan disediakan secara publik di Kaggle. Dataset ini juga tersedia dalam versi terdistribusi di Kaggle dengan struktur berkas yang telah disesuaikan untuk kebutuhan analisis berbasis citra.

CBIS-DDSM berisi citra mamografi yang telah dianotasi oleh ahli radiologi dengan dua kategori temuan utama, yaitu *calcification* dan *mass lesion*. Label klasifikasi yang digunakan mencakup dua kelas, yaitu *benign* dan *malignant*. *Calcification* menggambarkan endapan kalsium kecil yang tampak sebagai bintik putih pada citra mamografi, sedangkan *mass lesion* mengacu pada massa jaringan atau benjolan yang dapat mengindikasikan adanya pertumbuhan abnormal pada area payudara. Contoh visual kategori *benign* dan *malignant* ditunjukkan pada Gambar 3.2.



Gambar 3.2. Contoh citra mamografi kategori *benign* dan *malignant*.

Koleksi resmi CBIS-DDSM pada TCIA terdiri atas sekitar 1.566 subjek dengan lebih dari 10.239 citra dalam format DICOM. Pada penelitian ini, data diakses dari versi Kaggle yang merupakan replikasi dari dataset TCIA tersebut. Dataset ini menjadi dasar untuk tahap pra-pemrosesan metadata, pembagian dataset, serta pelatihan model dalam skema *Federated Learning*.

3.2.1 Struktur Dataset

Dataset CBIS-DDSM menyediakan metadata kasus yang memuat informasi identitas pasien, label patologi, serta jalur berkas citra. Ringkasan struktur kolom yang digunakan pada penelitian ini disajikan pada Tabel 3.1.

Tabel 3.1. Struktur kolom pada berkas deskripsi dataset CBIS-DDSM

Kolom	**Keterangan**
`patient_id`	Nomor identifikasi unik pasien
`pathology`	Label patologi (`BENIGN`, `BENIGN_WITHOUT_CALLBACK`, `MALIGNANT`)
`image file path`	Jalur citra mamografi (format JPEG)
`cropped image path`	Jalur citra hasil pemotongan (*cropped*)
`ROI mask path`	Jalur berkas penanda area lesi (*Region of Interest*)

Seluruh label `pathology` diseragamkan menjadi dua kelas biner:

- *Benign*: gabungan antara `BENIGN` dan `BENIGN_WITHOUT_CALLBACK`.

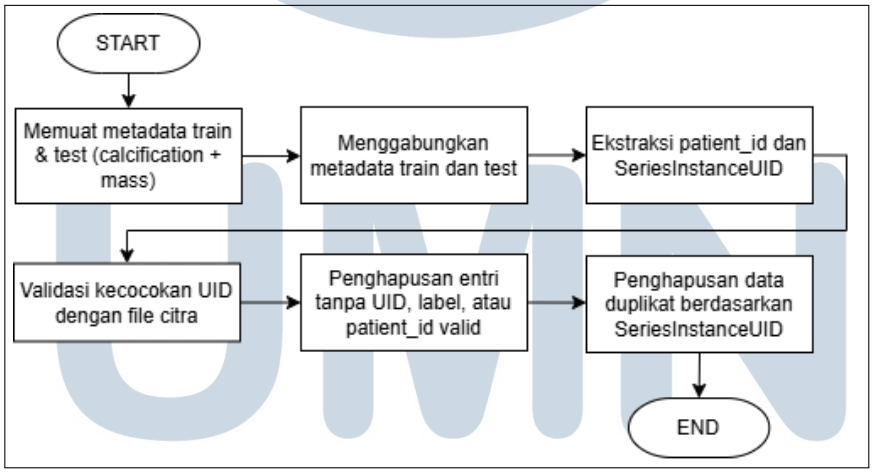
- *Malignant*: diambil dari label `MALIGNANT`.

Empat berkas CSV digunakan sebagai sumber metadata utama, masing-masing memuat deskripsi kasus untuk kategori *calcification* dan *mass lesion* pada subset pelatihan dan pengujian. Ringkasan jumlah entri untuk setiap berkas ditunjukkan pada Tabel 3.2.

Tabel 3.2. Ringkasan jumlah entri pada setiap berkas CSV CBIS-DDSM

Nama Berkas	Kategori	Jumlah Entri
`calc_case_description_train_set.csv`	*Calcification*	1.546
`calc_case_description_test_set.csv`	*Calcification*	326
`mass_case_description_train_set.csv`	*Mass lesion*	1.318
`mass_case_description_test_set.csv`	*Mass lesion*	378

3.3 Pra-pemrosesan Data



Gambar 3.3. Flowchart pra-pemrosesan data

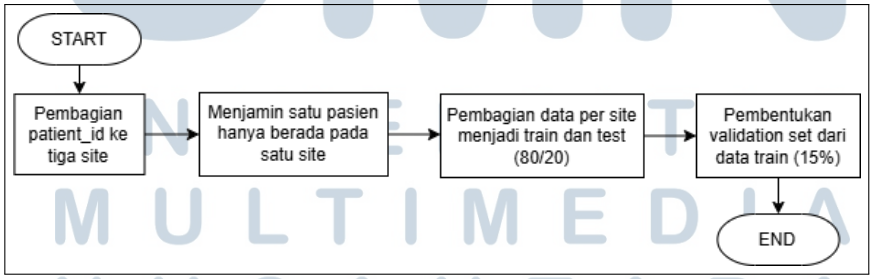
Tahapan pra-pemrosesan metadata ditunjukkan pada Gambar 3.3, mulai dari penggabungan berkas CSV hingga pembersihan entri tidak valid dan duplikat. Tahap pra-pemrosesan data pada penelitian ini berfokus pada pengolahan metadata untuk membentuk satu himpunan data yang konsisten sebelum dilakukan pembagian dataset dan pelatihan model. Pra-pemrosesan mencakup penggabungan berkas CSV, penyeragaman format kolom, validasi keterhubungan metadata dengan berkas citra, serta pembersihan entri yang tidak valid atau duplikat.

Empat berkas CSV pada CBIS-DDSM terdiri atas dua subset (pelatihan dan pengujian) dan dua kategori temuan (*calcification* dan *mass lesion*). Untuk mempermudah pengelolaan, dua berkas pelatihan (`calc_...train...` dan `mass_...train...`) digabungkan menjadi satu metadata *train*, sedangkan dua berkas pengujian (`calc_...test...` dan `mass_...test...`) digabungkan menjadi satu metadata *test*. Selanjutnya, kedua metadata tersebut diseragamkan struktur kolomnya agar konsisten dan dapat diproses dengan pipeline yang sama.

Validasi dilakukan dengan memeriksa keberadaan `patient_id`, label `pathology`, serta jalur citra yang dapat diakses. Entri yang tidak memiliki `patient_id` atau label patologi yang valid dihapus. Duplikasi entri juga dihapus untuk memastikan setiap citra/seri hanya dihitung satu kali pada tahap pembagian dataset. Selain itu, bila metadata memuat pengenal seri (misalnya *SeriesInstanceUID*/UID), validasi dilakukan dengan mencocokkan UID yang diekstraksi dari jalur berkas dengan struktur direktori citra yang tersedia, sehingga metadata benar-benar mereferensikan citra yang ada.

Hasil pra-pemrosesan metadata menghasilkan himpunan data yang konsisten untuk tahap berikutnya, yaitu pembagian dataset dan pelatihan model. Secara ringkas, output utama dari tahap ini adalah: (1) metadata gabungan untuk subset *train* dan *test*, (2) label patologi yang telah diseragamkan menjadi dua kelas biner (*benign*/*malignant*), serta (3) data yang telah dibersihkan dari entri tidak valid dan duplikasi sehingga meminimalkan risiko kebocoran atau ketidakkonsistenan saat pembagian data berbasis `patient_id`.

3.4 Pembagian Dataset



Gambar 3.4. Flowchart pembagian dataset

Proses pembagian dataset berbasis `patient_id` untuk membentuk tiga *site* serta subset *train*, *validation*, dan *test* diilustrasikan pada Gambar 3.4. Pembagian

dataset dirancang untuk mensimulasikan skenario *Federated Learning* sekaligus mencegah kebocoran informasi lintas himpunan. Pembagian dilakukan berbasis `patient_id` sehingga seluruh citra milik pasien yang sama tidak tersebar ke himpunan yang berbeda. Strategi ini penting untuk mencegah *patient-level leakage*, yaitu kondisi ketika citra dari pasien yang sama muncul pada *train* dan *test* atau pada klien yang berbeda.

Pada penelitian ini, setiap `patient_id` dipetakan secara deterministik ke tiga klien fiktif (`site0`, `site1`, dan `site2`) menggunakan fungsi hash MD5(`patient_id`) mod 3. Pemetaan deterministik memastikan replikasi eksperimen, sekaligus menjaga bahwa setiap pasien hanya muncul pada satu *site*.

Setelah data dipetakan ke masing-masing *site*, dilakukan pembagian *train* dan *test* dengan rasio 80%/20% pada setiap *site*. Selanjutnya, himpunan *validation* dibentuk dengan mengambil 15% dari data *train*. Dengan demikian, proporsi akhir pada setiap *site* menjadi sekitar 68% untuk *train*, 12% untuk *validation*, dan 20% untuk *test*. Pemisahan dilakukan secara *stratified* terhadap label untuk menjaga stabilitas proporsi *benign* dan *malignant* pada setiap *split*. Seluruh citra kemudian diorganisasi ke dalam struktur direktori:

`cbisdataset/site{0-2}/{train,val,test}/{benign,malignant}/.`

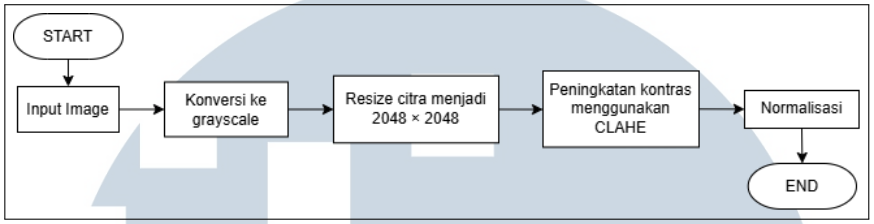
Tabel 3.3 merangkum jumlah entri pada setiap kombinasi *site/phase/label*. Dengan skema pembagian tersebut, distribusi data antarklien dapat dibuat relatif seimbang dan seluruh pembagian tetap *patient-disjoint* antar *site*.

Tabel 3.3. Rekap jumlah data per *site* dan *phase* (entri per label).

Site	**Train**			**Val**			**Test**			**Total**
	Benign	Malignant	Subtotal	Benign	Malignant	Subtotal	Benign	Malignant	Subtotal	
site0	390	306	696	69	54	123	118	88	206	1.025
site1	383	338	721	68	60	128	136	84	220	1.069
site2	377	294	671	67	52	119	131	88	219	1.009
Total	1.150	938	2.088	204	166	370	385	260	645	3.103

Berdasarkan pembagian pada Tabel 3.3, setiap *site* memiliki subset *train*, *validation*, dan *test* dengan distribusi label yang relatif seimbang. Selain itu, pemetaan berbasis `patient_id` memastikan tidak terjadi irisan pasien antar *site* maupun antar subset, sehingga evaluasi model pada data *test* lebih merepresentasikan kemampuan generalisasi pada pasien yang benar-benar baru.

3.5 Data Transformation

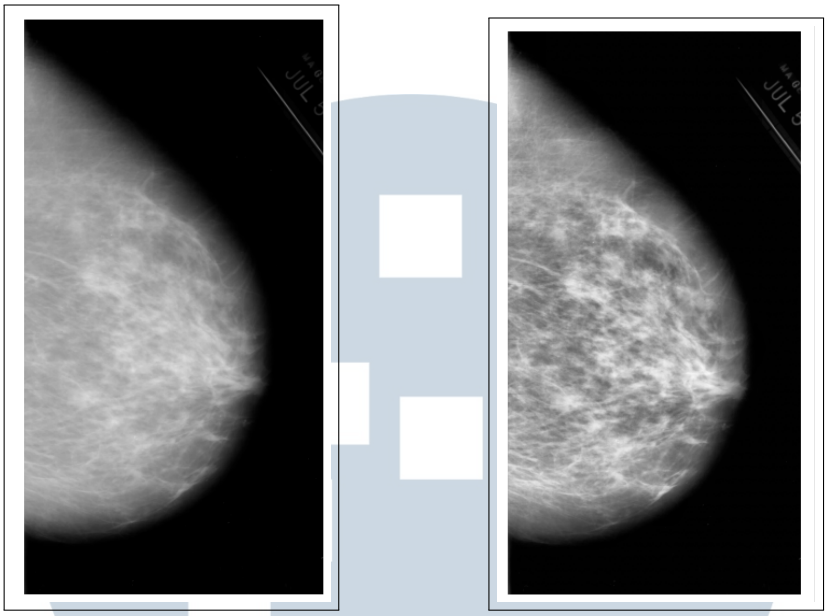


Gambar 3.5. Flowchart *data transformation*

Gambar 3.5 mengilustrasikan transformasi citra pada tahap pemuatan data sebelum pelatihan lokal. *Data transformation* diterapkan untuk menyesuaikan representasi citra mamografi agar sesuai dengan masukan model CNN. Tahap ini dilakukan secara lokal pada masing-masing klien dan bersifat *on-the-fly*, yaitu dijalankan saat pemuatan data sebelum citra dimasukkan ke model pada proses pelatihan lokal. Dengan pendekatan ini, data yang tersimpan pada direktori *site* tetap mempertahankan struktur pembagian *train*/*validation*/*test*, sedangkan transformasi citra dilakukan dinamis untuk setiap *mini-batch*.

Transformasi yang digunakan meliputi konversi citra menjadi *grayscale* satu kanal untuk menyesuaikan karakteristik citra mamografi, perubahan ukuran citra menjadi 2048×2048 piksel, peningkatan kontras lokal menggunakan metode *Contrast-Limited Adaptive Histogram Equalization* (CLAHE), konversi ke tensor numerik, serta normalisasi intensitas dengan `mean=[0.5]` dan `std=[0.5]`. Kombinasi transformasi ini bertujuan meningkatkan keterbacaan struktur jaringan pada citra kontras rendah serta menstabilkan proses pelatihan.

Contoh hasil transformasi CLAHE ditunjukkan pada Gambar 3.6.



(a) Citra asli (b) Setelah CLAHE

Gambar 3.6. Perbandingan citra mamografi sebelum dan sesudah peningkatan kontras menggunakan *Contrast-Limited Adaptive Histogram Equalization* (CLAHE).

Implementasi transformasi citra dilakukan secara modular. Kelas `CLAHEPreprocess` digunakan untuk melakukan CLAHE dan penyaringan halus (*Gaussian blur*) guna mereduksi *noise* lokal.

```
import cv2
import numpy as np
from PIL import Image

class CLAHEPreprocess:
    def __init__(self, clip_limit=2.0, tile_grid_size=(8,8)):
        self.clip_limit = clip_limit
        self.tile_grid_size = tile_grid_size

    def __call__(self, img):
        img_np = np.array(img)
        if len(img_np.shape) == 3:
            img_np = cv2.cvtColor(img_np, cv2.COLOR_RGB2GRAY)

        clahe = cv2.createCLAHE(clipLimit=self.clip_limit,
                                tileGridSize=self.tile_grid_size)
        img_clahe = clahe.apply(img_np)
        img_blur = cv2.GaussianBlur(img_clahe, (3, 3), 0)
        return Image.fromarray(img_blur)
```

Kode 3.1: Implementasi transformasi CLAHE

Pipeline transformasi kemudian dibangun menggunakan `torchvision.transforms` agar seluruh transformasi diterapkan berurutan secara konsisten.

```python
from torchvision import transforms

data_transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.Resize((2048, 2048)),
    CLAHEPreprocess(clip_limit=2.0, tile_grid_size=(8,8)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])
])
```

Kode 3.2: Konfigurasi pipeline *data transformation*

3.6 Arsitektur Sistem

Bagian ini menjelaskan rancangan sistem *Federated Learning* yang mengintegrasikan *Federated Averaging (FedAvg)* dengan *Packed CKKS Homomorphic Encryption*. Sistem memastikan proses pelatihan kolaboratif berlangsung tanpa pertukaran data mentah antarklien, karena pembaruan model dikomunikasikan dalam bentuk *ciphertext*. Arsitektur terdiri dari tiga komponen utama: server global, klien lokal, dan modul enkripsi CKKS.

1. Server Global

   Server menginisialisasi model global dan mendistribusikannya kepada seluruh klien sebagai inisialisasi pelatihan lokal. Sinkronisasi parameter dilakukan secara periodik berdasarkan parameter *pace*, yaitu ketika jumlah langkah pelatihan lokal mencapai kelipatan tertentu atau ketika langkah terakhir dalam satu *epoch* tercapai.

   Pada setiap ronde agregasi, server menerima pembaruan dari klien dalam bentuk *ciphertext* CKKS beserta *mask* sparsifikasi. Agregasi dilakukan secara langsung pada domain homomorfik menggunakan operasi linear CKKS berupa penjumlahan dan perkalian skalar untuk membentuk *ciphertext* global. Bobot agregasi klien ditetapkan secara seragam $\left(w_i = \frac{1}{K}\right)$ atau proporsional terhadap jumlah data latih lokal $\left(w_i = \frac{n_i}{\sum_j n_j}\right)$ sesuai konfigurasi sistem.

   Server menghitung *sum mask* sebagai faktor normalisasi yang merepresentasikan total kontribusi efektif pada setiap batch terenkripsi. Hasil agregasi berupa pasangan *(ciphertext global, sum mask)* kemudian dibroadcast kembali ke seluruh klien untuk digunakan pada ronde berikutnya. Parameter global per ronde, metrik evaluasi, waktu komputasi, serta biaya komunikasi dicatat untuk kebutuhan analisis.

2. Klien Lokal

   Klien menjalankan pelatihan secara mandiri menggunakan data lokal tanpa mengirimkan data mentah ke server. Setiap klien menyimpan model klasifikasi dan diskriminator domain. Model klasifikasi mempelajari prediksi label pada distribusi data masing-masing *site*, sedangkan diskriminator domain mendukung penyelarasan representasi fitur lintas klien.

   Pelatihan lokal dilakukan selama *nsteps* pada setiap *epoch*. Pada setiap langkah, model klasifikasi diperbarui dengan meminimalkan *cross-entropy loss*. Mekanisme *feature alignment* diterapkan melalui pembelajaran *adversarial*, di mana representasi fitur *encoder* diberi gangguan *noise* kecil, diskriminator domain dilatih untuk membedakan asal domain, dan *encoder* dioptimasi agar menghasilkan representasi yang lebih *invariant* terhadap domain, sehingga pergeseran distribusi antarklien dapat ditekan.

   Strategi *curriculum learning* diterapkan setelah fase *adversarial* tertentu. Prediksi benar dan salah pada data latih dihitung pada awal setiap *epoch* untuk membentuk bobot *sampling*, sehingga sampel yang relatif sulit atau terlupakan memperoleh probabilitas pemilihan yang lebih besar pada *epoch* berikutnya.

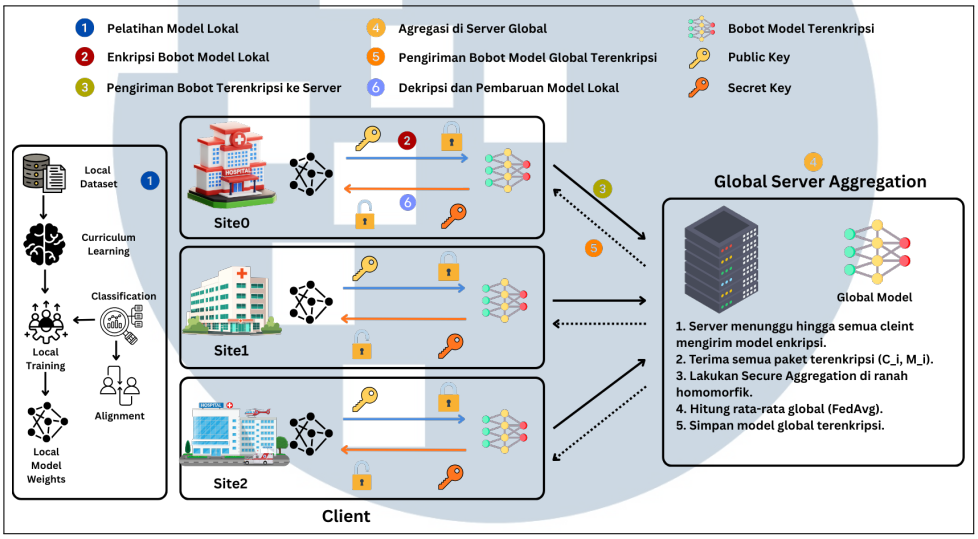
   Pada momen agregasi berbasis *pace*, parameter model diubah ke dalam bentuk vektor satu dimensi melalui proses *flattening* dan dipartisi menjadi *batch* berukuran tetap. Sparsifikasi *top-k* diterapkan pada tingkat *batch* dengan memilih *batch* yang memiliki rata-rata magnitudo absolut tertinggi. *Batch* terpilih dienkripsi menggunakan CKKS dan dikirim ke server bersama *mask* sparsifikasi. Setelah menerima pembaruan global, klien mendekripsi *ciphertext*, melakukan normalisasi menggunakan *sum mask*, merekonstruksi parameter model, dan melanjutkan pelatihan pada ronde berikutnya.

3. *Packed CKKS Homomorphic Encryption*

   Modul enkripsi menerapkan skema CKKS melalui pustaka *TenSEAL*. Konteks CKKS dibangkitkan satu kali pada awal pelatihan dan dibagikan sebagai konteks publik agar seluruh klien menggunakan konfigurasi enkripsi yang identik, sedangkan kunci privat dipertahankan pada sisi klien untuk keperluan dekripsi.

   Parameter model dipadatkan (*packing*) ke dalam slot-slot CKKS sehingga satu *ciphertext* dapat memuat banyak elemen parameter secara simultan.

Untuk menekan biaya komunikasi, sparsifikasi *top-k* diterapkan pada tingkat *batch* hasil *packing*. Server melakukan agregasi langsung pada *ciphertext* terpilih tanpa mengakses parameter individual klien. Nilai *sum mask* digunakan pada tahap dekripsi untuk menormalkan hasil agregasi sesuai kontribusi efektif tiap *batch*, sehingga kerahasiaan pembaruan model dan efisiensi komunikasi tetap terjaga.



Gambar 3.7. Arsitektur sistem *Federated Learning* dengan integrasi *Packed CKKS Homomorphic Encryption*

Seluruh proses pelatihan berlangsung secara iteratif sepanjang beberapa *epoch*, dengan agregasi federasi yang dijalankan secara periodik mengikuti parameter *pace*. Urutan prosesnya adalah sebagai berikut:

1. Server membentuk direktori komunikasi bersama dan menginisialisasi model global. Model ini kemudian didistribusikan ke seluruh klien sebagai inisialisasi parameter pelatihan lokal. Pada tahap ini ditetapkan seluruh parameter federasi, meliputi jumlah *epoch*, jumlah langkah lokal per *epoch* (*nsteps*), nilai *pace*, ukuran *batch* enkripsi, serta rasio sparsifikasi *top-k*.
2. Konteks *Packed CKKS Homomorphic Encryption* diinisialisasi satu kali pada awal pelatihan. Konteks ini memuat parameter skema CKKS dan kunci yang diperlukan untuk operasi homomorfik. Konteks publik disimpan pada direktori komunikasi agar seluruh klien menggunakan konfigurasi enkripsi yang identik, sementara kunci privat tetap berada di sisi klien untuk keperluan dekripsi.

3. Pada setiap *epoch*, masing-masing klien menjalankan pelatihan lokal selama *nsteps* menggunakan data lokal. Pembaruan model klasifikasi dilakukan dengan meminimalkan *cross-entropy loss*. Secara paralel, mekanisme *feature alignment* diterapkan melalui pembelajaran adversarial lintas-*site*, di mana representasi fitur diberi gangguan *noise* kecil dan diskriminator domain dilatih untuk membedakan asal domain, sementara *encoder* dioptimasi agar menghasilkan representasi yang *invariant* terhadap domain.

4. Setelah fase *adversarial* tertentu, strategi *curriculum learning* diterapkan. Pada awal setiap *epoch*, klien menghitung prediksi benar/salah pada data latih untuk membentuk bobot *sampling*. Bobot ini digunakan dalam *weighted sampler* sehingga sampel yang lebih sulit memperoleh probabilitas pemilihan yang lebih besar pada *epoch* berikutnya.

5. Ketika jumlah langkah lokal mencapai kelipatan *pace*, atau ketika langkah lokal terakhir dalam satu *epoch* tercapai, proses komunikasi federasi dijalankan. Setiap klien menyiapkan pembaruan parameter model untuk dikirim ke server secara aman.

6. Parameter model lokal diubah ke dalam bentuk vektor satu dimensi melalui proses *flattening* dan dipartisi ke dalam *batch* sesuai ukuran slot enkripsi CKKS. Sparsifikasi *top-k* diterapkan pada tingkat *batch* dengan memilih *batch* yang memiliki rata-rata magnitudo absolut tertinggi. Hanya *batch* terpilih yang dienkripsi menjadi *ciphertext* dan dikirim ke server bersama *mask* sparsifikasi.

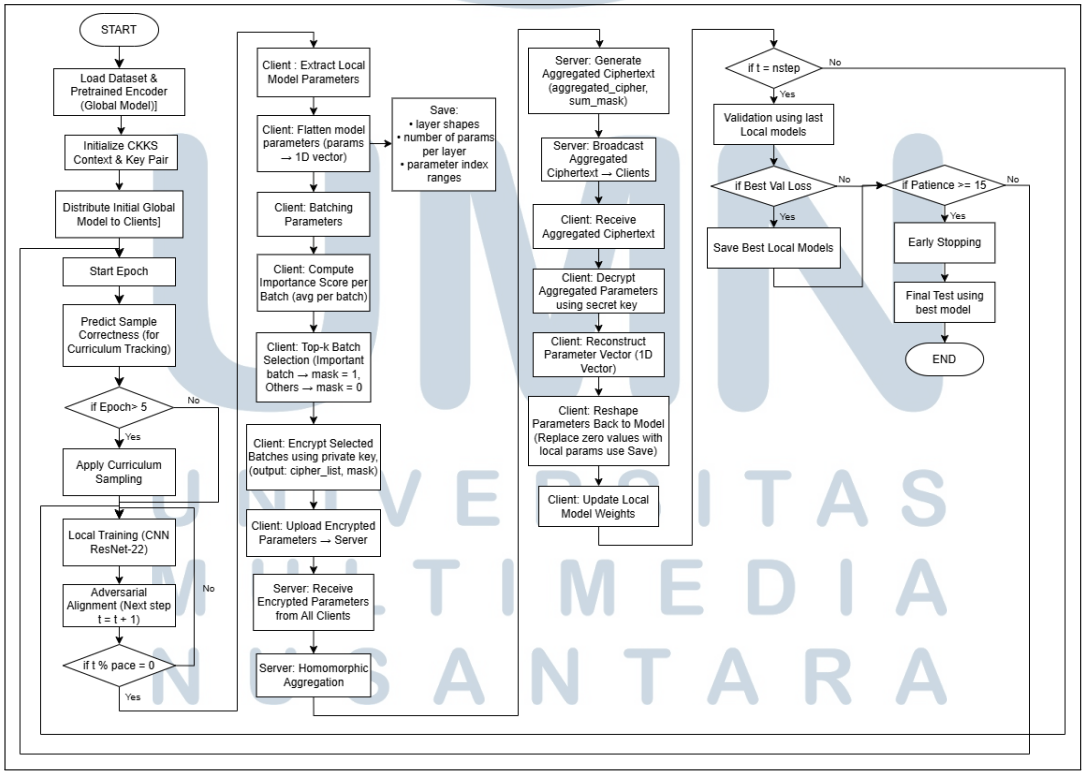
7. Server menerima *ciphertext* dan *mask* dari seluruh klien, kemudian melakukan agregasi homomorfik secara langsung pada *ciphertext* aktif menggunakan operasi linear CKKS. Agregasi dapat dilakukan secara seragam atau berbobot sesuai konfigurasi, sehingga menghasilkan *ciphertext global* dan *sum mask* yang merepresentasikan kontribusi efektif pada setiap *batch*. Seluruh proses agregasi dilakukan tanpa mendekripsi pembaruan klien.

8. *Ciphertext global* dan *sum mask* dibroadcast kembali ke seluruh klien. Setiap klien mendekripsi *ciphertext* menggunakan kunci privat dan melakukan normalisasi berdasarkan *sum mask*. Vektor hasil dekripsi kemudian direkonstruksi menjadi parameter model. Elemen bernilai nol akibat sparsifikasi dipertahankan menggunakan parameter lokal yang bersesuaian,

lalu vektor dipetakan kembali ke struktur *tensor* model untuk melanjutkan pelatihan pada ronde berikutnya. Model global direpresentasikan sebagai rata-rata dari model lokal hasil rekonstruksi tersebut.

9. Evaluasi dilakukan pada akhir setiap *epoch* menggunakan *validation set* di masing-masing klien. Metrik yang dihitung meliputi *validation loss*, akurasi, AUC, dan PR–AUC. Model terbaik disimpan berdasarkan penurunan *validation loss*. Pelatihan dilanjutkan ke *epoch* berikutnya apabila nilai terbaik masih mengalami perbaikan; penghitung *patience* dinaikkan ketika tidak terjadi perbaikan hingga mencapai ambang tertentu, lalu proses dihentikan melalui *early stopping*. Pengujian akhir dilakukan menggunakan model terbaik yang tersimpan.

Gambar 3.8 menggambarkan tahapan komputasi utama dalam sistem *Federated Learning* yang diusulkan. Alur proses meliputi inisialisasi model global, pembentukan konteks CKKS, pelatihan dan enkripsi parameter lokal, agregasi *ciphertext* di server, hingga dekripsi dan evaluasi model global pada setiap ronde federasi.



Gambar 3.8. Alur proses *Federated Learning* dengan mekanisme *Packed CKKS Homomorphic Encryption*

Seluruh proses eksperimen dijalankan pada lingkungan komputasi berbasis *Ubuntu* dengan spesifikasi perangkat keras dan perangkat lunak sebagaimana ditunjukkan berikut:

- Sistem Operasi: Ubuntu 20.04.6 LTS (64-bit)
- Prosesor: Intel® Core™ i7-7700 CPU @ 3.60 GHz × 8
- Memori (RAM): 32 GB
- Kartu Grafis: NV132 (NVIDIA)
- Kapasitas Penyimpanan: 4.5 TB
- Sistem Jendela: X11 (GNOME 3.36.8)

3.6.1 Threat Model

Threat model pada penelitian ini mendefinisikan ruang lingkup keamanan sistem *Federated Learning* yang dianalisis, mencakup aset yang dilindungi, aktor ancaman, serta asumsi keamanan yang digunakan dalam penerapan *Homomorphic Encryption*.

Sistem *Federated Learning* terdiri atas sejumlah klien yang melakukan pelatihan model secara lokal menggunakan data citra medis masing-masing dan sebuah *Parameter Server* (PS) yang bertugas mengoordinasikan proses agregasi parameter model global. PS tidak memiliki akses terhadap data mentah klien dan hanya memproses pembaruan model dalam bentuk terenkripsi.

Aset utama yang dilindungi meliputi data citra medis lokal milik klien serta informasi sensitif yang dapat tersirat dalam parameter atau pembaruan model selama proses pelatihan federasi. Kebocoran informasi dari parameter model berpotensi mengungkap karakteristik data medis klien dan menimbulkan pelanggaran privasi.

Model ancaman yang digunakan mengikuti skenario *honest-but-curious*, di mana PS dan klien diasumsikan menjalankan protokol *Federated Learning* sesuai spesifikasi, tetapi tetap diperlakukan sebagai pihak yang tidak sepenuhnya dipercaya karena berpotensi mencoba memperoleh informasi tambahan dari data yang diterima. Skenario serangan aktif seperti manipulasi pembaruan model, pengiriman parameter palsu, maupun kolusi antara PS dan klien tidak termasuk dalam cakupan penelitian ini.

Skema *Homomorphic Encryption* yang digunakan mengasumsikan satu pasangan kunci kriptografi yang dibagikan (*shared key pair*) di antara seluruh klien. Kunci publik digunakan oleh klien untuk mengenkripsi pembaruan model, sedangkan kunci privat digunakan oleh klien untuk mendekripsi hasil agregasi global. PS tidak memiliki akses ke kunci privat dan hanya melakukan operasi aritmetika secara homomorfik pada *ciphertext*. Distribusi kunci diasumsikan dilakukan melalui kanal yang aman, serta tidak terjadi kolusi antara PS dan klien.

Komunikasi antara klien dan PS diasumsikan dapat diobservasi oleh pihak eksternal tanpa adanya modifikasi aktif selama transmisi. Perlindungan terhadap ancaman inferensi dan kebocoran informasi dilakukan dengan memastikan bahwa parameter model tetap berada dalam bentuk terenkripsi selama proses transmisi dan agregasi melalui skema *Packed CKKS Homomorphic Encryption*. Pendekatan ini digunakan sebagai simulasi untuk mengevaluasi dampak penggunaan *Homomorphic Encryption* terhadap performa model dan overhead komputasi, dan tidak dimaksudkan sebagai representasi langsung dari sistem *Federated Learning* pada lingkungan produksi.

3.7 Implementasi Model

Implementasi model pada penelitian ini berfokus pada arsitektur *Convolutional Neural Network (CNN)* yang digunakan sebagai model klasifikasi pada setiap klien. Penelitian ini tidak mengusulkan arsitektur CNN baru, melainkan mengadopsi arsitektur dari penelitian terdahulu dan menerapkannya secara identik pada seluruh klien untuk menjaga konsistensi struktur model selama pelatihan federatif. Model yang digunakan terdiri atas dua komponen, yaitu *encoder* sebagai ekstraktor fitur dan *classifier* sebagai pemetaan fitur ke ruang keluaran kelas, sehingga pembaruan parameter hasil pelatihan lokal dapat diagregasi secara kompatibel pada tahap agregasi global sesuai mekanisme yang telah dijelaskan pada bagian sebelumnya.

Ekstraksi fitur citra mamografi menggunakan *encoder* berbasis ResNet-22 (*ViewResNetV2*) yang diadopsi dari Jimenez-Sánchez et al. *Encoder* dirancang untuk memproses citra mamografi *grayscale* beresolusi tinggi dan menghasilkan representasi fitur yang stabil. Setiap sampel memuat empat *view* (L-CC, L-MLO, R-CC, dan R-MLO) yang diproses menggunakan *shared weights* untuk mempertahankan konsistensi representasi antar-*view*. Representasi fitur dari seluruh *view* kemudian digabungkan melalui *average pooling* sehingga diperoleh satu

vektor fitur global berdimensi 256. *Encoder* ResNet-22 tersusun atas lima kelompok blok residual dengan konfigurasi $[2, 2, 2, 2, 2]$ dan jumlah filter yang meningkat bertahap dari 16 hingga 256, dengan *Batch Normalization* dan aktivasi ReLU pada tiap blok untuk mendukung stabilitas pelatihan.

Vektor fitur keluaran *encoder* menjadi masukan bagi *classifier* berbasis *Multi-Layer Perceptron (MLP)* untuk menghasilkan probabilitas kelas. *Classifier* terdiri dari beberapa lapisan *fully connected* yang dikombinasikan dengan ReLU dan *Batch Normalization*, serta *Dropout* (0,2) sebagai regularisasi guna mengurangi risiko *overfitting*. Pembaruan bobot *encoder* dan *classifier* menjadi parameter yang dikomunikasikan dalam proses *federated learning*, baik pada mode komunikasi *plain* maupun mode terenkripsi, sesuai konfigurasi sistem.

3.8 Implementasi Federated Learning

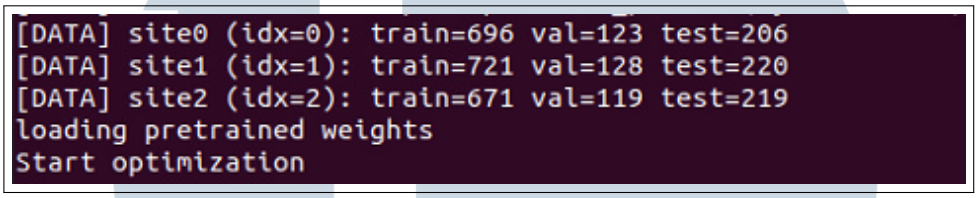
Implementasi *federated learning* pada penelitian ini dirancang untuk mendukung pelatihan model klasifikasi secara terdistribusi dengan tetap menjaga privasi data lokal di setiap klien. Sistem terdiri atas tiga klien, yaitu `site0`, `site1`, dan `site2`, serta satu server global yang berperan sebagai koordinator agregasi model. Seluruh proses pelatihan dilakukan tanpa pertukaran data mentah antarklien. Setiap klien hanya mengirimkan pembaruan parameter model hasil pelatihan lokal ke server, sehingga data sensitif tetap tersimpan secara lokal di masing-masing *site*.

Kerangka *federated learning* yang digunakan mengadopsi pendekatan yang diperkenalkan oleh Jimenez-Sánchez et al. dengan mengintegrasikan mekanisme *adversarial domain alignment* dan *curriculum learning* pada tahap pelatihan lokal. Agregasi parameter global dilakukan menggunakan skema *Federated Averaging (FedAvg)* yang dijalankan secara periodik mengikuti parameter *pace*. Selain itu, penelitian ini menambahkan mekanisme *early stopping* di sisi server untuk mengendalikan konvergensi model dan mencegah pelatihan berlebih. Dengan kombinasi tersebut, sistem *federated learning* tidak hanya berfokus pada kolaborasi pelatihan antar klien, tetapi juga pada stabilitas proses optimisasi dan generalisasi model global.

3.8.1 Inisialisasi Klien dan Pembagian Dataset

Implementasi *federated learning* pada penelitian ini mengikuti pipeline pelatihan yang diperkenalkan oleh Jimenez-Sánchez et al., termasuk tahap

inisialisasi model menggunakan bobot *pretrained*. Bobot *pretrained* tersebut dimuat pada awal eksekusi sebagai inisialisasi parameter sebelum proses optimisasi federasi dimulai. Untuk memastikan proses inisialisasi berjalan sesuai implementasi referensi, sistem mencetak status pemuatan bobot *pretrained* serta ringkasan pembagian jumlah sampel (*train/validation/test*) pada tiap klien ditunjukkan pada Gambar 3.9.



Gambar 3.9. Output runtime pemuatan bobot *pretrained* dan ringkasan pembagian data (*train/validation/test*) pada masing-masing klien.

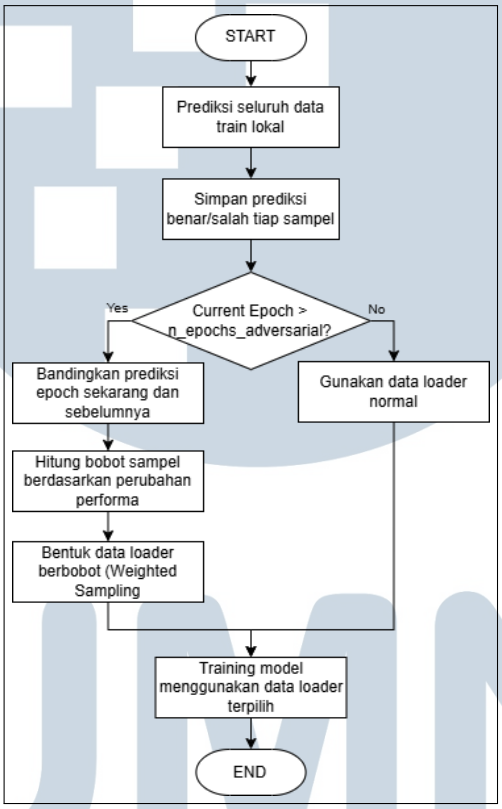
3.8.2 Skema Pelatihan Federated Learning

Pelatihan *federated learning* dijalankan secara sinkron dengan pembaruan model global yang dilakukan secara periodik. Pada setiap klien, model dilatih secara lokal selama sejumlah langkah pelatihan (*nsteps*) dalam satu *epoch*. Parameter *pace* digunakan untuk mengatur frekuensi komunikasi antara klien dan server, sehingga agregasi global dapat dilakukan beberapa kali dalam satu *epoch* tanpa harus menunggu seluruh proses pelatihan lokal selesai. Pada penelitian ini, konfigurasi *pace* dan *nsteps* diatur sedemikian rupa sehingga dalam satu *epoch* dapat terjadi beberapa ronde agregasi global.

Pada setiap ronde agregasi, server menerima pembaruan parameter dari seluruh klien dan membentuk model global baru. Model hasil agregasi tersebut kemudian didistribusikan kembali ke masing-masing klien sebagai inisialisasi pelatihan lokal selanjutnya. Skema ini memungkinkan pertukaran informasi antar klien terjadi secara bertahap dan berulang, sehingga proses pembelajaran kolaboratif dapat berlangsung secara lebih stabil meskipun distribusi data pada setiap klien tidak identik. Frekuensi agregasi global yang dipicu oleh parameter *pace* ditunjukkan melalui *output runtime* pada Gambar 3.10.

Gambar 3.10. Contoh *output runtime* pemicu agregasi FedAvg pada langkah ke-40, ke-80, dan ke-120 dari total *nsteps*=120 dalam satu *epoch*, sesuai pengaturan parameter *pace*=40.

3.8.3 Strategi *Curriculum Learning*



Gambar 3.11. Alur strategi *curriculum learning* pada pelatihan lokal.

Alur pada Gambar 3.11 merangkum mekanisme *curriculum learning* yang digunakan untuk menyesuaikan prioritas sampel selama pelatihan lokal. Pada awal *epoch*, model melakukan prediksi pada seluruh data *train* lokal dan menyimpan status prediksi benar/salah untuk setiap sampel. Jika *epoch* saat ini telah melewati fase *adversarial* (misalnya `epoch` > `n_epochs_adversarial`), prediksi pada *epoch* berjalan dibandingkan dengan *epoch* sebelumnya untuk mengukur perubahan performa per sampel. Perubahan ini kemudian digunakan untuk menghitung bobot sampel, sehingga sampel yang mengalami penurunan prediksi atau lebih sulit dipelajari memperoleh bobot lebih tinggi. Selanjutnya, bobot tersebut digunakan

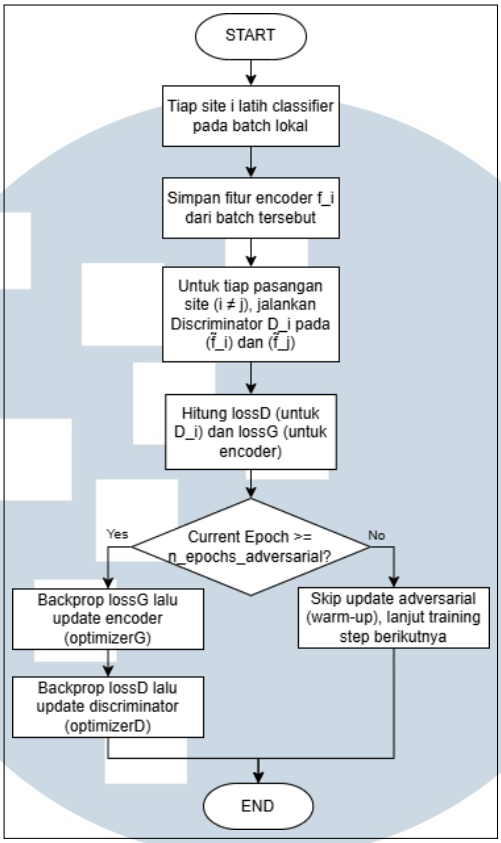
untuk membentuk *data loader* berbobot melalui *Weighted Sampling* agar sampel prioritas lebih sering muncul pada iterasi pelatihan berikutnya. Apabila fase *adversarial* belum selesai, pelatihan dilakukan menggunakan *data loader* normal tanpa pembobotan.

Strategi *curriculum learning* diterapkan untuk mengatur distribusi kemunculan sampel pelatihan secara adaptif selama proses pelatihan lokal. Pendekatan ini juga mengacu pada metode yang diperkenalkan oleh Jimenez-Sánchez et al., di mana tingkat kesulitan sampel ditentukan berdasarkan dinamika performa prediksi model antar *epoch*. Sampel yang cenderung lebih sulit dipelajari atau mengalami penurunan performa prediksi diberikan bobot yang lebih besar, sehingga memiliki peluang lebih tinggi untuk dipilih pada proses pelatihan berikutnya.

Pendekatan *curriculum learning* ini mendorong model untuk belajar secara bertahap, dimulai dari sampel yang relatif lebih mudah hingga sampel yang lebih kompleks. Dalam konteks *federated learning*, strategi ini berperan penting untuk menjaga stabilitas optimisasi ketika model harus beradaptasi dengan distribusi data yang berbeda pada setiap klien dan mengalami pembaruan global secara periodik.

3.8.4 Mekanisme *Adversarial Domain Alignment*

Perbedaan distribusi data antarklien ditangani melalui mekanisme *adversarial domain alignment* yang mengacu pada metode Jiménez-Sánchez *et al.* Pendekatan ini bertujuan menyelaraskan representasi fitur yang dihasilkan *encoder* agar bersifat *domain-invariant*, sehingga model global menjadi lebih *robust* terhadap variasi data antar *site*. Setiap klien mempertahankan sebuah *domain discriminator* yang dilatih untuk membedakan asal domain dari representasi fitur, sementara *encoder* dioptimasi secara *adversarial* untuk menghasilkan fitur yang sulit dibedakan oleh *discriminator* tersebut.

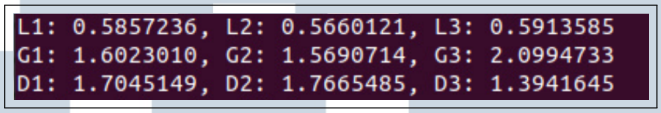


Gambar 3.12. Alur mekanisme *adversarial domain alignment* pada pelatihan lokal.

Alur pada Gambar 3.12 merangkum proses penyelarasan fitur lintas domain pada pelatihan lokal. Setiap *site* terlebih dahulu melatih *classifier* pada *mini-batch* lokal dan menyimpan representasi fitur f_i dari *encoder*. Selanjutnya, untuk setiap pasangan *site* (i, j), *discriminator* D_i membedakan fitur dari domain asal (f_i) dan domain lain (f_j) untuk menghitung *loss discriminator* $(\mathscr{L}_D)$ dan *loss generator* $(\mathscr{L}_G)$. Pembaruan komponen *adversarial* diaktifkan setelah melewati fase *warm-up*, yaitu ketika *epoch* memenuhi kondisi `epoch` $\geq$ `n_epochs_adversarial`; sebelum kondisi tersebut terpenuhi, langkah *adversarial* dilewati agar representasi dasar untuk klasifikasi terbentuk lebih stabil.

Aktivasi mekanisme *adversarial* tidak dilakukan sejak awal pelatihan. Proses penyelarasan fitur baru diaktifkan setelah model melewati fase pelatihan awal selama sejumlah *epoch* tertentu. Penundaan ini bertujuan memastikan bahwa *encoder* telah mempelajari representasi fitur dasar yang cukup stabil untuk tugas klasifikasi sebelum dilakukan penyelarasan lintas domain. Dengan strategi ini, proses *adversarial domain alignment* dapat berjalan lebih terkendali dan tidak mengganggu konvergensi awal model.

Keluaran pelatihan *adversarial* dipantau melalui pencetakan nilai *loss* komponen klasifikasi dan komponen *adversarial* selama pelatihan lokal berlangsung. Nilai tersebut dicetak secara periodik mengikuti siklus pelatihan yang dipicu oleh parameter *pace*, sehingga dalam satu *epoch* dapat terjadi beberapa kali pembaruan model yang melibatkan langkah klasifikasi dan langkah *adversarial*. Contoh keluaran *debug* nilai *loss* komponen tersebut ditunjukkan pada Gambar 3.13.



Gambar 3.13. Contoh output *runtime* yang menampilkan nilai *loss* komponen pelatihan, termasuk komponen terkait mekanisme *adversarial domain alignment* (mis. *loss generator* dan *loss discriminator*) yang dipantau selama pelatihan lokal.

3.8.5 Validasi, *Early Stopping*, dan Pengujian

Evaluasi performa model dilakukan pada akhir setiap *epoch* menggunakan *validation set* di masing-masing klien. Metrik yang dihitung pada tahap validasi dibatasi pada *validation loss* dan akurasi. Nilai *validation loss* digunakan sebagai indikator utama untuk memantau konvergensi model dan menentukan model terbaik selama proses pelatihan. Model dengan nilai *validation loss* terendah disimpan sebagai kandidat model terbaik untuk setiap klien.

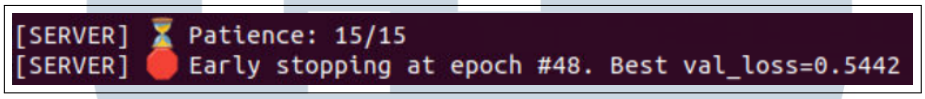
Mekanisme *early stopping* digunakan untuk menghentikan pelatihan ketika model tidak lagi menunjukkan peningkatan performa pada data validasi. Mekanisme ini dikendalikan oleh parameter *patience*, yang menyatakan jumlah maksimum *epoch* berturut-turut tanpa perbaikan nilai *validation loss*. Proses pelatihan dihentikan secara otomatis apabila nilai *validation loss* global tidak mengalami penurunan selama jumlah *epoch* tersebut. Implementasi *early stopping* dijalankan setelah proses validasi pada setiap *epoch*. Logika pemantauan nilai *validation loss* dan penghentian pelatihan ditunjukkan pada Kode 3.3.

```
if avg_loss < best_loss:
    best_loss = avg_loss
    patience_counter = 0
    torch.save({"state_dict": local_models[i].state_dict()},
        os.path.join(COMM_PATH, "agg", f"best_site{i}.pt"))
else:
    patience_counter += 1
```

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

Kode 3.3: Implementasi mekanisme *early stopping* (main.py)

Verifikasi *runtime* dilakukan melalui keluaran program yang menunjukkan penghentian pelatihan sebelum mencapai jumlah *epoch* maksimum ketika tidak terjadi perbaikan nilai *validation loss*. Contoh keluaran *runtime early stopping* ditunjukkan pada Gambar 3.14.



Gambar 3.14. Contoh output runtime yang menunjukkan aktivasi mekanisme *early stopping* ketika nilai *validation loss* tidak mengalami perbaikan selama sejumlah *epoch* berturut-turut.

Pengujian akhir (*final test*) dijalankan setelah pelatihan dihentikan menggunakan model terbaik yang tersimpan. Tahap ini mengevaluasi performa model pada *test set* di masing-masing klien menggunakan metrik akurasi, AUC, dan PR–AUC untuk menilai kemampuan generalisasi model terhadap data yang tidak digunakan selama pelatihan.

3.8.6 Parameter dan Konfigurasi Sistem Federasi

Konfigurasi sistem Federated Learning yang digunakan dalam implementasi ini dirangkum pada Tabel 3.4.

Tabel 3.4. Konfigurasi dan parameter utama Federated Learning

Parameter	Nilai / Keterangan
Framework	PyTorch dan TenSEAL
Jumlah klien	3 (`site0`, `site1`, `site2`)
Jumlah epoch federasi	100
Langkah lokal per epoch (`nsteps`)	120
Interval agregasi (`pace`)	40
	Lanjut pada halaman berikutnya

Tabel 3.4 Konfigurasi dan parameter utama Federated Learning (lanjutan)

Parameter	Nilai / Keterangan
Jumlah agregasi per epoch	3 kali
Optimizer	Adam
Learning rate	1×10^{-5}
Weight decay	1×10^{-4}
Batch size validasi & test	4
Ukuran citra masukan	2048×2048 (grayscale)
Jumlah kelas	2 (benign, malignant)
Curriculum learning	Aktif setelah epoch 5
Early stopping (`patience`)	15 epoch tanpa perbaikan

3.9 Implementasi Packed CKKS Homomorphic Encryption

Bagian ini menjelaskan penerapan *Packed CKKS Homomorphic Encryption* pada mode ckks. Berbeda dari mode *plain*, pembaruan parameter klien tidak dikirim dalam bentuk *plaintext*, melainkan dienkripsi dan diagregasi langsung dalam domain *ciphertext*. Seluruh alur CKKS dijalankan di dalam `main.py` pada setiap momen agregasi berbasis *pace*. Seluruh tahapan implementasi divalidasi melalui *runtime debugging* yang secara eksplisit mencetak struktur data, ukuran parameter, rentang indeks (*flat range*), batch terenkripsi, serta hasil dekripsi. *Output debug* ini digunakan sebagai dasar dokumentasi implementasi, di mana setiap potongan kode yang disajikan pada subbagian berikut memiliki korespondensi langsung dengan hasil *debug* yang ditampilkan selama eksekusi sistem.

3.9.1 Inisialisasi Konteks CKKS

Konteks CKKS dibangkitkan sekali di awal pelatihan ketika mode *ckks* dipilih. Konteks diserialisasi lengkap beserta *secret key* dan disimpan di `COMM_PATH/context_params`. Klien kemudian memuat konteks ini setiap kali menjalankan enkripsi maupun dekripsi parameter.

```
ckks_ctx = ts.context(
    ts.SCHEME_TYPE.CKKS,
    poly_modulus_degree=8192,
```

```
    coeff_mod_bit_sizes=[60, 40, 40, 60]
)
ckks_ctx.global_scale = 2**40
ckks_ctx.generate_galois_keys()
params = ckks_ctx.serialize(save_secret_key=True)

with open(os.path.join(COMM_PATH, "context_params"), "wb") as f:
    f.write(params)
```

Kode 3.4: Inisialisasi dan penyimpanan konteks CKKS

Keberhasilan penyimpanan konteks dan kelanjutan proses pelatihan dapat dilihat pada Gambar 3.15.

```
[CKKS] dir(ctx) filtered: ['__module__', 'auto_mod_switch', 'auto_rescale', 'galois_keys',
'generate_galois_keys', 'generate_relin_keys', 'global_scale', 'has_galois_keys', 'has_publ
ic_key', 'has_relin_keys', 'has_secret_key', 'public_key', 'relin_keys', 'secret_key']
loaded OK
has_secret_key: True
has_galois_keys: True
global_scale: 1099511627776.0
[CKKS] Context saved → ./comm/context_params (bytes=704882)
```

Gambar 3.15. *Output debug* saat konteks CKKS berhasil diserialisasi dan disimpan ke `COMM_PATH/context_params`.

3.9.2 Serialisasi Parameter, Packing Batch, dan Enkripsi Top-k

Alur *Packed* CKKS pada sisi klien mencakup serialisasi parameter model, pembagian parameter ke dalam *batch*, serta enkripsi *batch* terpilih menggunakan strategi sparsifikasi *top-k*. Seluruh tahapan dijalankan pada setiap momen agregasi federasi yang dipicu oleh parameter *pace*. Proses ini menyiapkan pembaruan parameter dalam bentuk *ciphertext* agar dapat diagregasi secara homomorfik di sisi server.

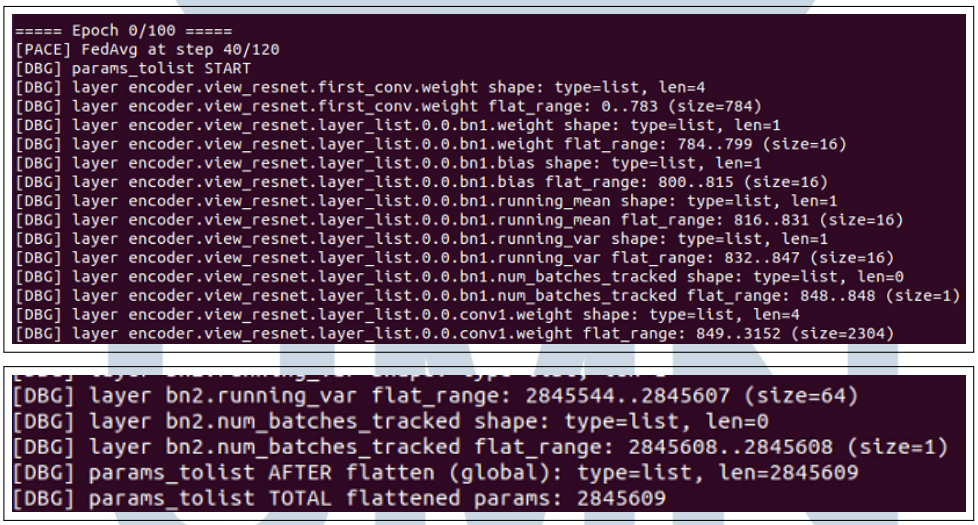
Ekstraksi parameter model lokal dilakukan menggunakan fungsi `params_tolist`. Parameter model dikonversi dari struktur *state dictionary* menjadi vektor satu dimensi (*flattened list*). Fungsi ini juga menghasilkan metadata berupa jumlah elemen tiap lapisan (`params_num`) serta bentuk asli parameter (`layer_shape`). Metadata tersebut digunakan untuk melacak rentang indeks (*flat range*) setiap lapisan dan memungkinkan rekonstruksi parameter ke bentuk semula setelah proses agregasi global.

```
params_lists  = {}
params_nums   = {}
layer_shapes  = {}
```

```
4
5  for i in range(n_sites):
6      params_list, params_num, layer_shape = params_tolist(
       local_models[i])
7      params_lists[i] = params_list
8      params_nums[i]  = params_num
9      layer_shapes[i] = layer_shape
10
11 total_sum = sum(params_num.values())
12 batch_num = int(np.ceil(total_sum / args.enc_batch_size))
```

Kode 3.5: Serialisasi parameter per klien dan perhitungan jumlah batch

Verifikasi *runtime* dilakukan dengan mencetak struktur lapisan, rentang indeks parameter, serta jumlah total parameter setelah proses *flattening*. Contoh *output debug* pada awal dan akhir proses `params_tolist` ditunjukkan pada Gambar 3.16.



Gambar 3.16. *Output debug* awal dan akhir proses `params_tolist` yang menunjukkan pemetaan parameter model ke vektor satu dimensi beserta rentang indeks tiap lapisan.

Pembagian *batch* dilakukan secara berurutan berdasarkan indeks pada vektor parameter global dengan ukuran tetap sesuai parameter `enc_batch_size`. Sparsifikasi kemudian diterapkan pada tingkat *batch* menggunakan strategi *top-k*. Nilai rata-rata magnitudo absolut dihitung pada setiap *batch* untuk menentukan *batch* terpilih. *Batch* terpilih direpresentasikan sebagai *mask* biner, sementara *batch* lainnya diabaikan pada ronde komunikasi tersebut.

Enkripsi *Packed* CKKS dijalankan hanya pada *batch* terpilih. Setiap *batch* dienkripsi menjadi satu *ciphertext* yang memuat ribuan parameter sekaligus.

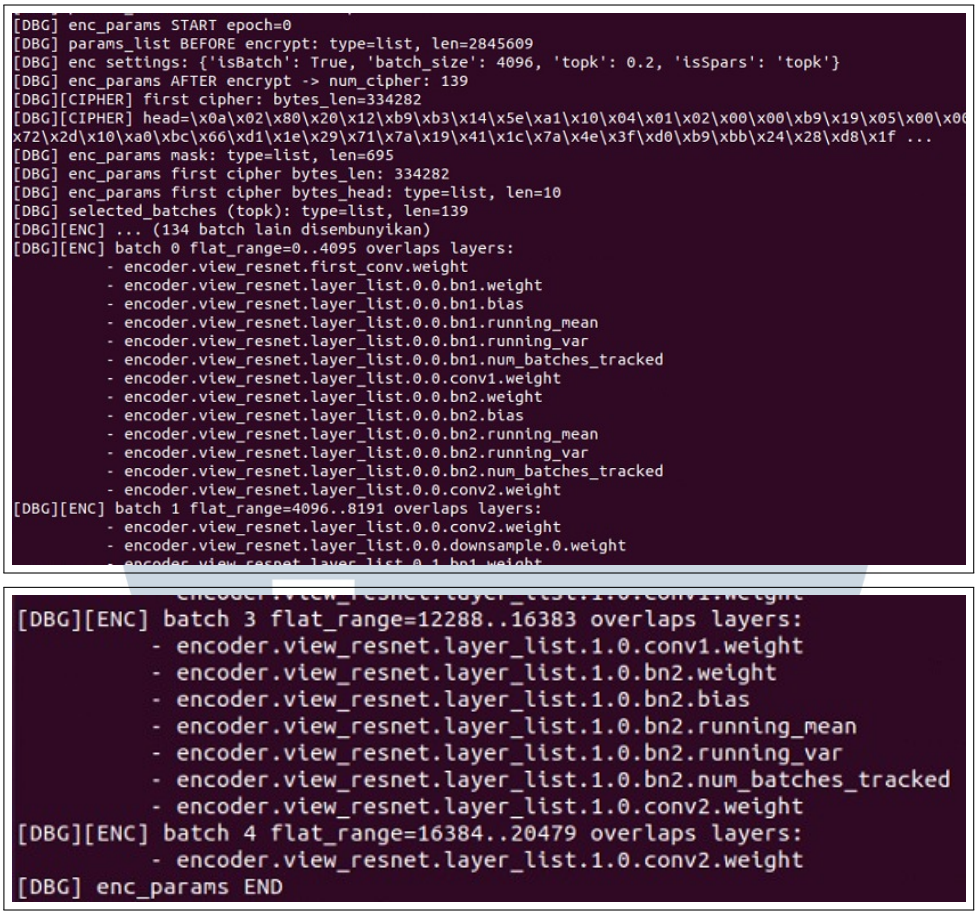
Informasi *mask* digunakan untuk menandai posisi *ciphertext* terhadap indeks *batch* pada vektor global.

```
cipher_lists = {}
masks        = {}

for i in range(n_sites):
    cipher, mask = enc_params(
        params_lists[i],
        {},
        args,
        epoch)
    cipher_lists[i] = cipher
    masks[i] = mask
```

Kode 3.6: Sparsifikasi *top-k* dan enkripsi *batch* per klien dengan *Packed* CKKS

Verifikasi *runtime* pada tahap enkripsi dilakukan dengan mencetak jumlah *batch* terpilih, jumlah ciphertext yang dihasilkan, serta keterkaitan *batch* terenkripsi terhadap lapisan model. Contoh *output debug* pada awal dan akhir proses `enc_params` ditunjukkan pada Gambar 3.17.



Gambar 3.17. *output debug* awal dan akhir proses enkripsi *batch* terpilih menggunakan *Packed* CKKS, termasuk informasi *mask top-k*, jumlah *ciphertext*, dan keterkaitan *batch* dengan lapisan model.

3.9.3 Agregasi Homomorfik Ciphertext di Server

Agregasi model global dilakukan langsung pada domain *ciphertext* menggunakan operasi linear pada skema CKKS. Agregasi dijalankan pada tingkat *batch* yang aktif berdasarkan *mask* hasil sparsifikasi *top-k*. Setiap *ciphertext batch* dari klien dijumlahkan secara homomorfik dan dikalikan dengan bobot kontribusi klien. Implementasi ini mendukung dua skema pembobotan, yaitu pembobotan seragam dan pembobotan proporsional terhadap ukuran data pelatihan lokal.

Proses agregasi menghasilkan `agg_cipher` sebagai *ciphertext global* dan `sum_mask` sebagai faktor normalisasi untuk setiap indeks *batch*. Normalisasi diperlukan karena tidak seluruh klien selalu mengirim *batch* pada indeks yang sama akibat sparsifikasi. Nilai `sum_mask` digunakan pada tahap dekripsi untuk menyesuaikan hasil agregasi dengan jumlah kontribusi yang valid pada setiap *batch*.

```python
if args.weighted:
    total_train = sum(len(trainsets[i]) for i in range(n_sites))
    weights_client = {i: len(trainsets[i]) / total_train for i in
   range(n_sites)}
else:
    weights_client = {i: 1.0 / n_sites for i in range(n_sites)}

print("[CKKS] weights_client =", weights_client)

sum_mask, agg_cipher = ckks_aggregate(
    cipher_lists,
    masks,
    weights_client,
    args,
    batch_num
)
```

Kode 3.7: Pemilihan bobot klien dan agregasi homomorfik *ciphertext* CKKS

Verifikasi *runtime* dilakukan dengan mencetak nilai `weights_client` pada setiap ronde agregasi untuk memastikan skema pembobotan yang digunakan sesuai konfigurasi eksperimen. Contoh keluaran *debug* yang menampilkan bobot klien ditunjukkan pada Gambar 3.18.

```
[DBG] enc_params END
[CKKS] weights_client = {0: 0.3333333333333333, 1: 0.3453065134099617, 2: 0.321360153256705}
[DBG] dec_params START
```

Gambar 3.18. *output debug* yang menampilkan bobot kontribusi klien (`weights_client`) pada tahap agregasi homomorfik *ciphertext*.

3.9.4 Dekripsi dan Rekonstruksi Parameter Model

Tahap ini memulihkan parameter global hasil agregasi homomorfik ke dalam domain *plaintext* dan merekonstruksi parameter tersebut ke bentuk model yang dapat digunakan kembali pada pelatihan lokal. Setiap klien mendekripsi *ciphertext* agregat menggunakan konteks CKKS privat yang sama dengan yang digunakan pada tahap enkripsi. Proses ini menghasilkan vektor parameter global dalam bentuk *flattened list*.

Normalisasi hasil dekripsi dilakukan menggunakan `sum_mask` untuk menyesuaikan kontribusi *batch* yang valid pada setiap indeks. *Batch* yang tidak dikirim pada tahap sparsifikasi menghasilkan nilai nol pada *plaintext*. Nilai nol

tersebut dipertahankan pada tahap dekripsi dan akan digantikan dengan parameter lokal yang bersesuaian pada proses rekonstruksi model.

```
decrypted_global = {}

for i in range(n_sites):
    dec_list = dec_params(
        agg_cipher,      # aggregated ciphertext
        sum_mask,        # mask hasil agregasi
        {},              # enc_tools tidak digunakan
        args,
        []               # randk_list tidak digunakan
    )
    decrypted_global[i] = dec_list.tolist()
```

Kode 3.8: Dekripsi *ciphertext* agregat per klien

Verifikasi *runtime* dilakukan dengan mencetak status dekripsi serta cuplikan nilai hasil dekripsi untuk memastikan proses pemulihan *plaintext* berjalan sesuai konfigurasi CKKS. Contoh keluaran debug proses dekripsi ditunjukkan pada Gambar 3.19.

```
[CKKS] weights_client = {0: 0.3333333333333333, 1: 0.3453065134099617, 2: 0.321360153256705}
[DBG] dec_params START
[DBG] cipher_list BEFORE decrypt: num_cipher=695
[DBG][CIPHER] cipher[0]: bytes_len=131217
[DBG][CIPHER] head=\x0a\x02\x80\x20\x12\x80\x81\x08\x5e\xa1\x10\x04\x01\x02\x00\x00\x80\x00\x02\x00\x00\
xda\x3f\xf3\x2a\xb3\x76\x9d\xec\xd1\x8e\xa7\xe1\x4d\x7c\xba\x11\xe9\xde\x2a\xed\x1a\x5b\x05\xf9 ...
[DBG][CIPHER] cipher[1]: bytes_len=131217
[DBG][CIPHER] head=\x0a\x02\x80\x20\x12\x80\x81\x08\x5e\xa1\x10\x04\x01\x02\x00\x00\x80\x00\x02\x00\x00\
xda\x3f\xf3\x2a\xb3\x76\x9d\xec\xd1\x8e\xa7\xe1\x4d\x7c\xba\x11\xe9\xde\x2a\xed\x1a\x5b\x05\xf9 ...
[DBG][CIPHER] cipher[2]: bytes_len=131217
[DBG][CIPHER] head=\x0a\x02\x80\x20\x12\x80\x81\x08\x5e\xa1\x10\x04\x01\x02\x00\x00\x80\x00\x02\x00\x00\
xda\x3f\xf3\x2a\xb3\x76\x9d\xec\xd1\x8e\xa7\xe1\x4d\x7c\xba\x11\xe9\xde\x2a\xed\x1a\x5b\x05\xf9 ...
[DBG] ... 692 cipher lain disembunyikan
[DBG] sum_masks: type=list, len=695
[DBG] randk_list: type=list, len=0
[DBG] dec settings: {'isBatch': True, 'batch_size': 4096}
[DBG] dec_params first cipher bytes_len: 131217
[DBG] dec_params first cipher bytes_head: type=list, len=10
[DBG] dec_params AFTER decrypt (plain list/global): type=ndarray, len=2846720
[DBG] dec_params AFTER decrypt (plain head): type=ndarray, len=10
[DBG] dec_params END
```

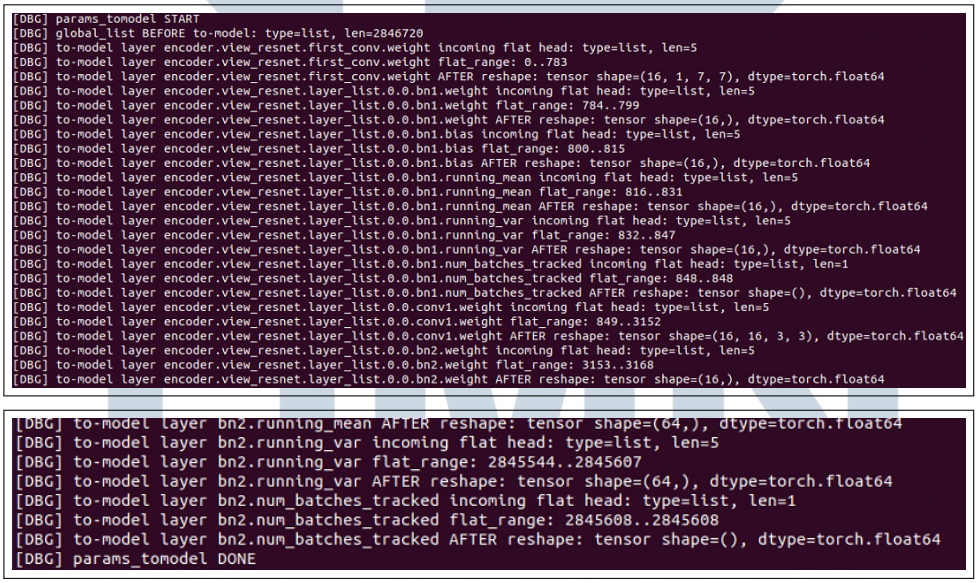
Gambar 3.19. Contoh *output debug* proses `dec_params` yang menunjukkan dekripsi *ciphertext* agregat CKKS menjadi parameter global dalam bentuk *flattened list*.

Rekonstruksi parameter global ke bentuk model dilakukan menggunakan `params_tomodel`. Proses diawali dengan pemetaan ulang parameter lokal menggunakan `params_tolist` untuk memperoleh metadata jumlah elemen dan bentuk asli setiap lapisan. Parameter global hasil dekripsi kemudian dimasukkan kembali ke struktur model. Elemen bernilai nol pada vektor global digantikan dengan parameter lokal yang bersesuaian sehingga lapisan yang tidak berkontribusi pada ronde tersebut tetap mempertahankan nilai sebelumnya.

```python
for i in range(n_sites):
    params_list, params_num, layer_shape = params_tolist(
    local_models[i])
    params_tomodel(
        local_models[i],
        decrypted_global[i],
        params_nums[i],
        layer_shapes[i],
        args,
        params_lists[i]
    )
```

Kode 3.9: Rekonstruksi parameter global ke model lokal

Keberhasilan proses rekonstruksi diverifikasi melalui *output runtime* yang mencetak status awal dan akhir pemetaan parameter kembali ke setiap lapisan model. Contoh *output debug* pada awal dan akhir proses `params_tomodel` ditunjukkan pada Gambar 3.20.



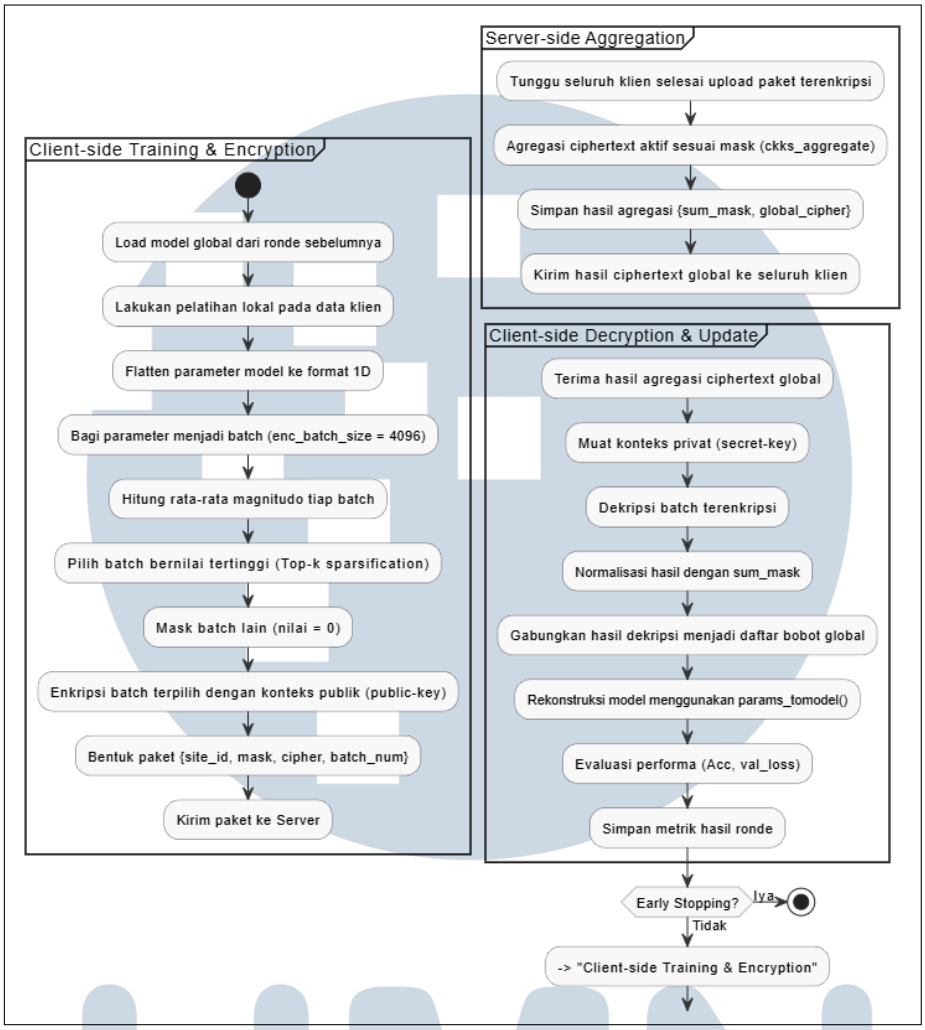
Gambar 3.20. *output debug* awal dan akhir proses rekonstruksi parameter global ke dalam struktur model lokal menggunakan `params_tomodel`.

Setelah seluruh klien selesai direkonstruksi, model hasil pembaruan digunakan sebagai parameter awal untuk ronde federasi berikutnya. Dengan mekanisme ini, proses pelatihan dapat berlanjut secara iteratif dengan memanfaatkan hasil agregasi homomorfik tanpa mengungkapkan parameter model individual pada sisi server. Parameter yang digunakan dalam eksperimen ini ditunjukkan pada Tabel 3.5.

Tabel 3.5. Parameter konfigurasi mekanisme *Packed CKKS Homomorphic Encryption* pada sistem federasi.

Parameter	**Nilai / Keterangan**
Skema	*Packed CKKS*
Library	TenSEAL
`poly_modulus_degree`	8192
`coeff_mod_bit_sizes`	[60, 40, 40, 60]
`global_scale`	2^{40}
`enc_batch_size`	4096
`isSpars`	`topk`
`topk`	0,2; 0,5; 0,8

Implementasi ini memungkinkan agregasi parameter dalam domain terenkripsi secara efisien. Pendekatan *batched packing* mengemas ribuan parameter dalam satu *ciphertext*, sementara sparsifikasi *top-k* menekan *data transfer volume* (MB) tanpa mengorbankan kualitas model secara signifikan. Penjagaan privasi dipertahankan karena server tidak pernah memegang *secret key* dan seluruh operasi agregasi dilakukan pada *ciphertext*. Alur lengkap integrasi mekanisme HE ke dalam FL dapat digambarkan dalam diagram pada Gambar 3.21.



Gambar 3.21. Alur Integrasi *Packed* CKKS HE.

3.9.5 Implikasi Rekonstruksi Model pada Skema *Top-k*

Penerapan *Packed CKKS Homomorphic Encryption* dengan mekanisme *top-k sparsification* menghasilkan pola pembaruan model yang berbeda dari *federated learning* konvensional. Pada setiap ronde federasi, hanya subset parameter dengan kontribusi terbesar yang dikirim ke server dan diagregasikan secara homomorfik, sedangkan parameter lain tetap dipertahankan secara lokal pada masing-masing klien. Skema ini membuat sinkronisasi model tidak lagi bersifat penuh, melainkan parsial dan bergantung pada parameter terpilih pada setiap ronde.

Rekonstruksi parameter dilakukan dengan memasukkan hasil agregasi pada indeks *batch* yang berkontribusi dan mempertahankan nilai parameter lokal sebelumnya pada *batch* yang tidak dikirim. Konsekuensinya, model hasil

rekonstruksi pada tiap klien tidak selalu identik secara parameter, meskipun seluruh klien menerima keluaran agregasi global yang sama. Perbedaan tersebut merupakan karakteristik yang inheren dari sparsifikasi *top-k*, bukan indikasi kegagalan agregasi maupun dekripsi.

Nilai *top-k* mengendalikan tingkat sinkronisasi global serta karakteristik pembaruan parameter dalam sistem federasi terenkripsi. Rasio *top-k* yang lebih besar mendorong perilaku sistem mendekati sinkronisasi penuh, sedangkan rasio yang lebih kecil memperbesar proporsi parameter yang tetap lokal dan memperkuat konsensus parsial antar klien. Karakteristik ini juga memunculkan efek regularisasi implisit karena sebagian parameter bertahan dari ronde ke ronde, sehingga pembaruan model menjadi lebih stabil terhadap variasi pembaruan lokal.

3.10 Evaluasi dan Pengukuran

Bagian ini menjelaskan metode evaluasi untuk mengukur kinerja sistem *Federated Learning* dengan agregasi terenkripsi berbasis CKKS. Evaluasi mencakup performa model, efisiensi komputasi, dan efisiensi komunikasi (*data transfer volume*). Seluruh eksperimen dijalankan melalui `main.py`. Pelaporan hasil mengacu pada ronde terbaik (*best round*), yaitu ronde ketika metrik evaluasi mencapai nilai maksimum selama pelatihan.

3.10.1 Evaluasi Performa Model

Evaluasi performa model dilakukan melalui dua tahap, yaitu validasi pada setiap *epoch* dan pengujian akhir (*final test*) setelah proses pelatihan selesai. Pada setiap *epoch*, validasi dijalankan di sisi klien menggunakan fungsi `val()` yang didefinisikan pada `client.py`. Model yang dievaluasi merupakan model lokal masing-masing klien, yang telah diperbarui melalui proses dekripsi dan rekonstruksi parameter hasil agregasi CKKS.

Pada tahap validasi, nilai *validation loss* dan akurasi dihitung untuk setiap klien. Nilai tersebut kemudian dirata-ratakan untuk memperoleh metrik global pada *epoch* bersangkutan. Rata-rata *validation loss* digunakan sebagai kriteria pemilihan model terbaik dan sebagai acuan mekanisme *early stopping*.

```
v_losses = []
v_accs   = []

for i in range(n_sites):
```

```
    vloss, vacc, _, _, _ = val(
        local_models[i], val_loaders[i], device
    )
    v_losses.append(vloss)
    v_accs.append(vacc)

avg_loss = float(np.mean(v_losses))
avg_acc  = float(np.mean(v_accs))
```

Kode 3.10: Validasi per-klien dan perhitungan metrik global (main.py)

Model terbaik pada setiap klien disimpan secara terpisah sebagai `best_site{i}.pt` ketika nilai *validation loss* global mencapai nilai minimum. Proses pelatihan dihentikan secara otomatis apabila tidak terjadi perbaikan nilai *validation loss* selama sejumlah *epoch* berturut-turut sesuai parameter *patience*.

Pengujian akhir (*final test*) dijalankan satu kali setelah proses pelatihan berhenti. Pada tahap ini, model terbaik masing-masing klien dimuat dan dievaluasi menggunakan `test set`. Metrik yang dihitung meliputi akurasi (ACC), AUC, dan PR–AUC.

```
best_path = os.path.join(COMM_PATH, "agg", f"best_site{i}.pt")
checkpoint = torch.load(best_path)

test_model = Classifier().to(device)
test_model.load_state_dict(checkpoint["state_dict"])

acc, roc_auc, pr_auc, cm, _ = final_test(
    test_model, test_loaders[i], device, test_loaders[i].dataset
)
```

Kode 3.11: Pengujian akhir model terbaik per klien (main.py)

3.10.2 Efisiensi Komputasi

Efisiensi komputasi diukur berdasarkan durasi pelatihan lokal dan durasi proses agregasi federasi selama pelatihan berlangsung. Pelatihan lokal dilakukan pada *inner loop* di masing-masing klien, sedangkan agregasi federasi dijalankan secara periodik sesuai parameter *pace*. Pada setiap event agregasi, sistem mencatat durasi proses agregasi *end-to-end*, yaitu sejak parameter lokal siap dipertukarkan hingga model hasil agregasi tersedia kembali untuk digunakan pada langkah berikutnya.

Total waktu agregasi federasi per *epoch* dihitung sebagai penjumlahan durasi seluruh event agregasi yang terjadi pada *epoch* tersebut. Waktu pelatihan lokal didefinisikan sebagai durasi *inner loop* dikurangi total waktu agregasi federasi, sehingga waktu pelatihan merepresentasikan komputasi murni untuk pembaruan model lokal (tanpa memasukkan waktu agregasi).

Pada mode CKKS, terdapat overhead tambahan akibat proses enkripsi dan dekripsi parameter. Waktu enkripsi (*encryption time*) dicatat ketika klien mengenkripsi parameter model (setelah *packing* dan sparsifikasi *top-k*) sebelum dikirim untuk agregasi. Waktu dekripsi (*decryption time*) dicatat ketika klien mendekripsi hasil agregasi terenkripsi dan merekonstruksi kembali parameter model. Nilai waktu enkripsi dan dekripsi dilaporkan sebagai rata-rata per *epoch* untuk menggambarkan *overhead* komputasi yang ditambahkan oleh mekanisme *homomorphic encryption*.

3.10.3 Efisiensi Data Transfer Volume (MB)

Efisiensi komunikasi diukur berdasarkan *data transfer volume*, yaitu total ukuran data yang dikirim (*upload*) dan diterima (*download*) selama proses agregasi federasi. Pengukuran dilakukan pada setiap event agregasi dan dinyatakan dalam satuan *megabyte* (MB), kemudian diakumulasikan untuk memperoleh total volume komunikasi per *epoch*.

Pada mode CKKS, ukuran data *upload* dihitung dari paket yang dikirim klien ke server setelah proses *packing* dan sparsifikasi *top-k*. Paket tersebut memuat *ciphertext batch* terpilih, *mask* biner yang menandai *batch* yang dienkripsi, serta metadata yang diperlukan untuk rekonstruksi, seperti jumlah *batch*. Ukuran paket ditentukan dari hasil serialisasi struktur data sebelum dikirimkan.

Ukuran data *download* dihitung dari *bundle global* yang di *broadcast* server ke seluruh klien setelah proses agregasi terenkripsi selesai. *Bundle* ini memuat *ciphertext* hasil agregasi global serta informasi normalisasi yang diperlukan agar klien dapat melakukan dekripsi dan merekonstruksi parameter model secara konsisten.

Nilai *upload* dan *download* dijumlahkan untuk memperoleh total *data transfer volume* pada setiap event agregasi. Seluruh volume komunikasi pada event agregasi dalam satu *epoch* kemudian diakumulasikan sebagai total *data transfer volume per epoch*.