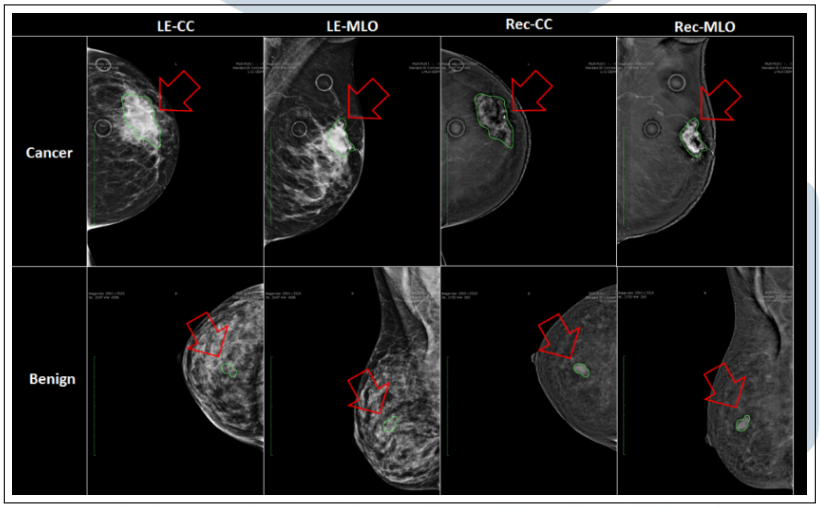
# BAB 2
# LANDASAN TEORI

## 2.1 Klasifikasi Kanker Payudara

Kanker payudara merupakan salah satu jenis kanker dengan prevalensi tertinggi pada perempuan sehingga deteksi dini menjadi faktor penting dalam meningkatkan efektivitas pengobatan. Metode tradisional diagnosis masih sangat bergantung pada keterlibatan tenaga medis yang berpotensi menimbulkan keterlambatan penanganan dan subjektivitas hasil. Pendekatan berbasis kecerdasan buatan mulai diterapkan dengan memanfaatkan dataset gambar kanker payudara dalam sistem klasifikasi otomatis untuk meningkatkan akurasi dan efisiensi. Pada tahap awal, penelitian berfokus pada rekayasa fitur secara manual, sedangkan perkembangan terbaru menunjukkan peran pembelajaran mendalam seperti *Convolutional Neural Networks* (CNN) dalam mempercepat proses deteksi kanker [10].



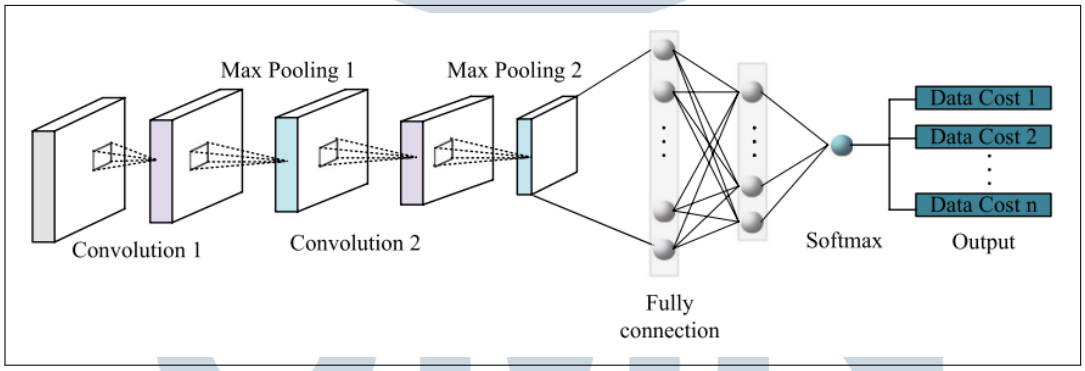

Gambar 2.1. Contoh gambar kanker payudara

Data medis bersifat sensitif dan dilindungi oleh regulasi ketat seperti *Health Insurance Portability and Accountability Act* (HIPAA) di Amerika Serikat, *General Data Protection Regulation* (GDPR) di Eropa, dan UU No. 17 Tahun 2023 tentang kesehatan di Indonesia sehingga tidak dapat dipertukarkan secara bebas antar institusi kesehatan. Ketersediaan data yang berkualitas tinggi dalam jumlah besar diperlukan untuk melatih model kecerdasan buatan yang efektif,

sedangkan keterbatasan akses data menimbulkan kontradiksi antara perlindungan privasi dan kebutuhan fusi data. Federated Learning hadir sebagai pendekatan yang memungkinkan pemanfaatan data tersebar tanpa memindahkan data mentah sehingga sesuai untuk pengembangan sistem diagnosis medis yang akurat dan juga menjaga privasi pasien [11].

Deteksi dini kanker payudara menjadi faktor penting untuk pengobatan dan keselamatan pasien. Pada tahap awal, gejala penyakit sering tidak tampak jelas sehingga banyak kelainan dapat terlewatkan [10]. Berbagai penelitian telah menerapkan pembelajaran mesin untuk meningkatkan deteksi dini, mengurangi risiko kematian, dan memperpanjang harapan hidup pasien. Pembagian data pasien masih jarang dilakukan karena pertimbangan privasi, teknis, dan hukum. Teknik keamanan dan privasi memungkinkan perlindungan data pasien yang lebih ketat sekaligus memungkinkan pemanfaatan data untuk penelitian dan keperluan klinis rutin [11].

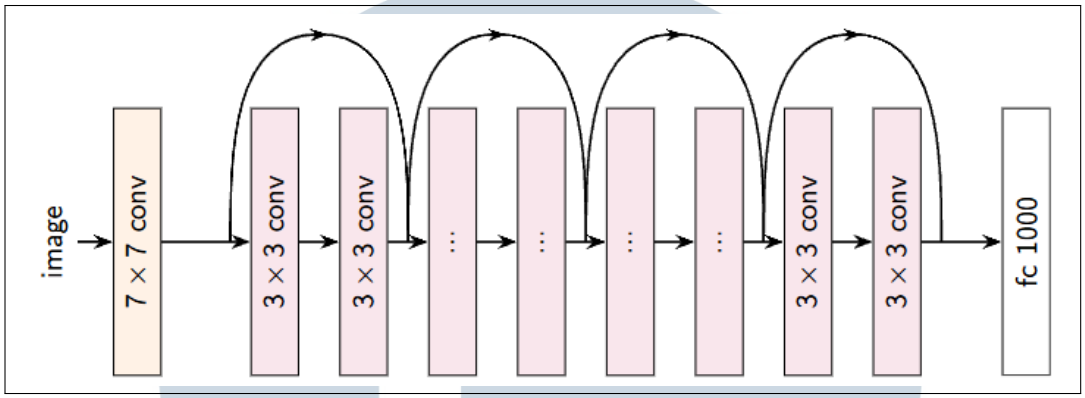
## 2.2 Resnet22 CNN




Gambar 2.2. Struktur Umum CNN untuk Klasifikasi Gambar

*Convolutional Neural Network* (CNN) merupakan salah satu arsitektur *deep learning* yang sangat populer dalam pengolahan citra karena kemampuannya melakukan ekstraksi fitur secara otomatis melalui lapisan konvolusi. Pada gambar 2.2, terlihat bahwa CNN terdiri dari beberapa tahapan seperti *convolution*, *pooling*, *activations*, dan *fully connected* yang bekerja secara berlapis-lapis untuk mengenali pola spasial pada gambar. Struktur bertingkat ini membuat CNN mampu mendeteksi fitur dasar seperti garis dan tepi pada lapisan awal, kemudian fitur yang lebih kompleks pada lapisan yang lebih dalam. Sejumlah penelitian menunjukkan CNN memiliki performa tinggi pada berbagai tugas computer vision dan aplikasi

medis, termasuk klasifikasi citra radiologi dan deteksi kanker, bahkan mampu menyamai kemampuan tenaga profesional kesehatan dalam beberapa kasus [12].




Gambar 2.3. Arsitektur Umum ResNet [13]

Penambahan lapisan pada CNN dapat meningkatkan akurasi, terlalu banyak lapisan sering menimbulkan masalah seperti *vanishing gradient*, *overfitting*, dan penurunan performa. Untuk mengatasinya, Residual Network (ResNet) pertama kali diperkenalkan oleh Kaiming He et al. dari Microsoft Research pada tahun 2015 sebagai solusi dan memperkenalkan konsep *residual learning* dengan *skip connection*, yang memungkinkan *input* suatu blok langsung diteruskan ke *output* tanpa melewati semua lapisan konvolusi. Dengan demikian, jaringan hanya mempelajari selisih (residual) antara *input* dan *output*, menjaga aliran gradient tetap stabil selama *training*. Pendekatan ini membuat jaringan dapat menjadi lebih dalam tanpa kehilangan akurasi. Pada gambar 2.3 ditunjukkan arsitektur umum dari ResNet yang terdiri atas beberapa blok residual. ResNet terbukti efektif sebagai backbone di berbagai penelitian *computer vision*, termasuk pada citra medis, karena mampu meningkatkan generalisasi tanpa menambah parameter secara berlebihan [14].

Tabel 2.1. Arsitektur ResNet22 CNN [5]

| Layer | Configuration |
|---|---|
| | **F: Feature Extractor** |
| 1 | Conv(1, 16, 3, 1, 1), MaxPool(3, 2, 0, 1) |
| 2.1 | Block(16, 16, 3, $s_1$=1, $s_2$=1, 1), Conv(16, 16, 1, 1) |

Lanjut pada halaman berikutnya

Tabel 2.1 Arsitektur ResNet22 CNN (lanjutan)

| Layer | Configuration |
|---|---|
| 2.2 | Block(16, 32, 3, $s_1$=2, $s_2$=1, 1) |
| 3.1 | Block(16, 32, 3, $s_1$=2, $s_2$=1, 1), Conv(16, 32, 1, 2) |
| 3.2 | Block(16, 32, 3, $s_1$=2, $s_2$=1, 1) |
| 4.1 | Block(32, 64, 3, $s_1$=2, $s_2$=1, 1), Conv(32, 64, 1, 2) |
| 4.2 | Block(32, 64, 3, $s_1$=2, $s_2$=1, 1) |
| 5.1 | Block(64, 128, 3, $s_1$=2, $s_2$=1, 1), Conv(64, 128, 1, 2) |
| 5.2 | Block(64, 128, 3, $s_1$=2, $s_2$=1, 1) |
| 6.1 | Block(128, 256, 3, $s_1$=2, $s_2$=1, 1), Conv(128, 256, 1, 2) |
| 6.2 | Block(128, 256, 3, $s_1$=2, $s_2$=1, 1) |
| **Cls: Classifier** | |
| 1 | FC(256, 128), BN, ReLU, Dropout() |
| 2 | FC(128, 64), BN, ReLU, Dropout() |
| 3 | FC(64, 2), Sigmoid |
| **D: Domain Discriminator** | |
| 1 | FC(256, 4), ReLU |
| 2 | FC(4, 2), Sigmoid |

Selain varian populer seperti ResNet-18, ResNet-50, dan ResNet-101, terdapat beberapa arsitektur turunan yang dirancang khusus untuk kebutuhan tugas tertentu, salah satunya ResNet-22. Pada penelitian [5], ResNet-22 digunakan sebagai *feature extractor* untuk klasifikasi kanker payudara. Arsitektur ini diawali dengan lapisan *convolution* sebagai pengolah fitur awal yang diikuti oleh *max pooling* untuk menurunkan dimensi representasi. Setelah itu, model menyusun beberapa blok residual (Block) pada setiap tingkat kedalaman, dimana setiap blok terdiri atas *batch normalization*, fungsi aktivasi ReLU, dan dua lapisan *convolution*. Konfigurasi ini memungkinkan jaringan mengekstraksi fitur citra secara bertahap mulai dari fitur sederhana hingga fitur spasial yang lebih kompleks pada level kedalaman yang lebih tinggi.

Setelah melalui bagian ekstraksi fitur, ResNet-22 terhubung dengan bagian *classifier* yang terdiri dari beberapa *fully connected* (FC) *layer*. Lapisan ini dilengkapi *batch normalization*, fungsi aktivasi ReLU, serta *dropout* untuk mengurangi *risiko overfitting*. Lapisan terakhir menggunakan fungsi aktivasi

sigmoid yang berperan menghasilkan probabilitas kelas, terutama ketika tugas klasifikasi bersifat biner. Selain itu, arsitektur tersebut juga dilengkapi dengan domain discriminator yang digunakan dalam skema federated learning untuk menangani distribusi data yang tidak seragam antar klien. *Domain discriminator* terdiri dari *fully connected layer* dengan aktivasi ReLU pada lapisan tersembunyi dan *sigmoid* pada lapisan keluaran untuk mendeteksi bias domain antar sumber data.

```
def resnet22(input_channels, activation):
    return ViewResNetV2(
        input_channels=input_channels,
        activation=activation,
        num_filters=16,
        first_layer_kernel_size=7,
        first_layer_conv_stride=2,
        blocks_per_layer_list=[2, 2, 2, 2, 2],
        block_strides_list=[1, 2, 2, 2, 2],
        block_fn=layers.BasicBlockV2,
        first_layer_padding=0,
        first_pool_size=3,
        first_pool_stride=2,
        first_pool_padding=0,
        growth_factor=2,
    )
```

Kode 2.1: Kode ResNet22

### 2.2.1 Encoder ResNet22

Encoder merupakan komponen yang berfungsi untuk mengekstraksi fitur dari citra masukan dan mengubahnya menjadi representasi vektor berdimensi tetap yang dapat digunakan pada tahap klasifikasi. Dalam konteks klasifikasi citra medis, encoder umumnya dirancang untuk menangkap pola visual penting seperti tekstur, tepi, dan struktur anatomis yang relevan [5].

Penelitian sebelumnya oleh Jiménez-Sánchez et al. [5] memanfaatkan arsitektur ResNet22 sebagai encoder dalam kerangka *Federated Learning* untuk klasifikasi kanker payudara. Arsitektur ini didasarkan pada prinsip *residual learning*, yang memungkinkan pelatihan jaringan dalam dengan lebih stabil melalui penggunaan *skip connection*. Pendekatan ini efektif dalam mengatasi permasalahan *vanishing gradient* yang sering muncul pada jaringan konvolusional yang dalam [5].

Secara umum, proses ekstraksi fitur diawali dengan lapisan konvolusi awal berukuran kernel besar untuk menangkap fitur tingkat rendah, diikuti oleh operasi *pooling* untuk mereduksi dimensi spasial. Selanjutnya, fitur diproses melalui beberapa *residual layer* yang tersusun atas blok-blok residual, dengan jumlah filter yang meningkat secara bertahap. Setiap blok residual biasanya dilengkapi dengan *Batch Normalization* dan fungsi aktivasi ReLU untuk menjaga kestabilan distribusi aktivasi selama proses pelatihan [5].

Pada tahap akhir, operasi *average pooling* digunakan untuk mengagregasi informasi spasial dan menghasilkan representasi fitur berdimensi tetap. Representasi ini berfungsi sebagai ringkasan karakteristik visual citra dan menjadi masukan bagi modul klasifikasi. Dalam arsitektur yang digunakan oleh Jiménez-Sánchez et al., encoder menghasilkan vektor fitur berdimensi 256 yang digunakan pada tahap pengambilan keputusan selanjutnya [5].

Alur proses *encoder* dapat dijelaskan sebagai berikut:

1. **Lapisan awal:** *Image* masuk melalui lapisan konvolusi berukuran kernel $7\times7$ diikuti *max pooling* $3\times3$ untuk melakukan ekstraksi fitur awal.
2. **Lima residual layer:** Setiap layer terdiri atas dua blok residual dengan jumlah filter meningkat secara bertahap ($16 \rightarrow 32 \rightarrow 64 \rightarrow 128 \rightarrow 256$). Setiap blok residual menggunakan *Batch Normalization* dan *ReLU activation* untuk menjaga kestabilan pelatihan.
3. **Output:** Setelah melewati seluruh residual layer, diperoleh representasi fitur akhir berdimensi 256 yang digunakan sebagai masukan pada tahap klasifikasi menggunakan *classifier*.

```
class Encoder(nn.Module):
    def __init__(self, input_channels=1, activation='relu'):
        super(Encoder, self).__init__()

        self.view_resnet = resnet22(input_channels, activation)
        self.all_views_avg_pool = layers.AllViewsAvgPool()
        self.all_views_gaussian_noise_layer = layers.
    AllViewsGaussianNoise(0.01)

    def forward(self, x):
        h = self.all_views_gaussian_noise_layer.
    single_add_gaussian_noise(x)
        result = self.view_resnet(h)
```

```
        h = self.all_views_avg_pool.single_avg_pool(result)
        return h

    def load_state_from_shared_weights(self, state_dict, view):
        view_angle = view.lower().split("-")[-1]
        view_key = view.lower().replace("-", "")
        self.view_resnet.load_state_dict(
            filter_strip_prefix(state_dict, "four_view_resnet.{}."
    .format(view_angle))
```

Kode 2.2: Kode Encoder ResNet22

### 2.2.2 Classifier

Classifier merupakan komponen yang bertugas memetakan representasi fitur yang dihasilkan oleh encoder ke dalam ruang kelas target. Pada arsitektur jaringan saraf konvolusional, classifier umumnya terdiri atas beberapa lapisan *fully connected* yang berfungsi mengintegrasikan fitur-fitur tingkat tinggi menjadi prediksi kelas [5].

Dalam arsitektur yang diadopsi oleh Jiménez-Sánchez et al. [5], classifier dibangun di atas representasi fitur berdimensi 256 yang dihasilkan oleh encoder ResNet22. Struktur classifier tersebut terdiri atas beberapa lapisan *fully connected* yang disusun secara bertahap untuk mereduksi dimensi fitur sebelum menghasilkan keluaran akhir. Setiap lapisan umumnya diikuti oleh *Batch Normalization* dan fungsi aktivasi ReLU guna menjaga stabilitas pelatihan serta meningkatkan kemampuan generalisasi model. Selain itu, mekanisme *Dropout* diterapkan sebagai bentuk regularisasi untuk mengurangi risiko *overfitting*.

Pada lapisan keluaran, fungsi aktivasi *Softmax* digunakan untuk mengubah nilai *logits* menjadi probabilitas antar kelas. Pendekatan ini memungkinkan model menghasilkan prediksi secara probabilistik, yang umum digunakan pada tugas klasifikasi biner maupun multikelas [15]. Dalam konteks klasifikasi kanker payudara, keluaran classifier merepresentasikan probabilitas citra termasuk ke dalam kelas *benign* atau *malignant*.

Secara berurutan, arsitektur *classifier* terdiri dari:

1. **Lapisan FC1:** Mengubah dimensi fitur dari 256 menjadi 128 neuron. Lapisan ini diikuti dengan aktivasi ReLU, *Batch Normalization*, dan *Dropout*.

2. **Lapisan FC2:** Mengubah dimensi dari 128 menjadi 64 neuron, juga dilengkapi dengan aktivasi ReLU, *Batch Normalization*, dan *Dropout*.

3. **Lapisan FC3:** Lapisan akhir dengan 2 neuron yang merepresentasikan dua kelas target, yaitu *benign* dan *malignant*. Fungsi aktivasi *Softmax* digunakan pada lapisan ini untuk mengubah nilai logit menjadi probabilitas masing-masing kelas.

```
class Classifier(nn.Module):
def init(self, num_classes=2):
    super(Classifier, self).init()
    self.encoder = Encoder()
    self.fc1 = nn.Linear(256, 128)
    self.fc2 = nn.Linear(128, 64)
    self.fc3 = nn.Linear(64, num_classes)
    self.bn1 = nn.BatchNorm1d(128)
    self.bn2 = nn.BatchNorm1d(64)
    self.drop1 = nn.Dropout()
    self.drop2 = nn.Dropout()

def forward(self, input):
    logits = self.encoder(input)
    logits = F.relu(self.bn1(self.fc1(logits)))
    logits = self.drop1(logits)
    logits = F.relu(self.bn2(self.fc2(logits)))
    logits = self.drop2(logits)
    logits = self.fc3(logits)
    probs = torch.nn.functional.softmax(logits, dim=1)
    return probs, logits
```

Kode 2.3: Kelas Classifier pada arsitektur ResNet22

## 2.3 Federated Learning

*Federated Learning* (FL) adalah kerangka kerja pembelajaran terdistribusi yang aman, di mana model global dibangun dengan melibatkan banyak klien tanpa membagikan data mentah secara langsung. Pendekatan ini menjadi solusi ketika data sulit atau tidak memungkinkan untuk dipusatkan, baik karena keterbatasan teknis maupun alasan privasi. Dalam banyak kasus, pengumpulan data yang besar dan berkualitas tinggi merupakan syarat penting untuk membangun model *machine learning* atau *deep learning* yang andal, seperti untuk deteksi anomali, klasifikasi

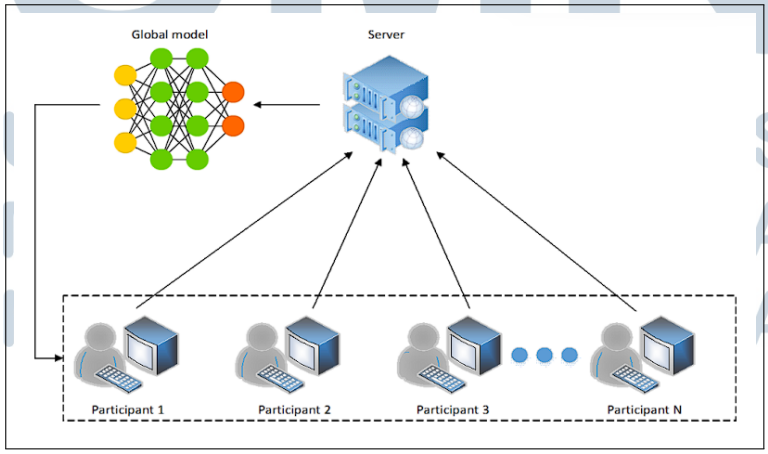
gambar, atau pemrosesan bahasa alami. Namun, jika data tersebar di berbagai lokasi dengan pembatasan akses, pelatihan terpusat (*centralized training*) tidak selalu bisa dilakukan [16].

Konsep FL pertama kali diperkenalkan oleh McMahan pada tahun 2017, dan mengusung metode pelatihan kolaboratif serta terdesentralisasi. Dalam FL, setiap klien memiliki data lokal yang digunakan untuk melatih model awal yang diberikan oleh server pusat. Setelah pelatihan selesai, klien tidak mengirimkan data mentah, melainkan parameter atau bobot (*weights*) hasil pelatihan lokal. Server pusat kemudian menggabungkan parameter yang diterima, biasanya dengan fungsi agregasi sederhana seperti rata-rata, untuk memperbarui model global. Model yang telah diperbarui dibagikan kembali ke klien, lalu proses ini diulang sampai jumlah ronde tercapai atau model sudah konvergen [17].

Secara umum, tahapan FL adalah sebagai berikut:

1. Server pusat membagikan model awal ke seluruh klien.
2. Setiap klien melatih model dengan data lokal.
3. Parameter atau bobot hasil pelatihan dikirim ke server melalui saluran yang aman.
4. Server menggabungkan parameter dari semua klien dan memperbarui model global.

Proses ini diulang sampai model global terfederasi memiliki performa yang mendekati model pelatihan terpusat. FL dapat mengatasi keterbatasan data terpusat, mengurangi latensi, dan menjaga privasi pengguna.




Gambar 2.4. Gambar Konsep Federated Learning [16]

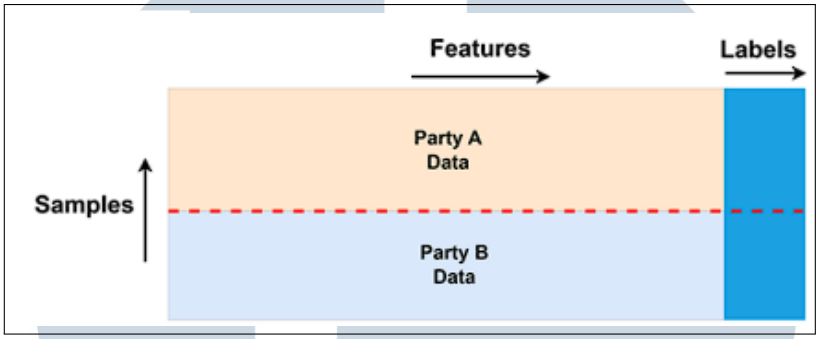
Federated Learning (FL) menjadi salah satu pendekatan penting dalam melatih model pembelajaran mesin tanpa harus mengakses langsung data lokal pengguna. Hal ini memberikan keuntungan besar dari sisi privasi, karena data sensitif tidak perlu dipindahkan atau dibagikan ke server pusat. Privasi menjadi isu krusial terutama di bidang kesehatan, misalnya dalam deteksi penyakit kanker payudara yang membutuhkan data medis pasien. Oleh karena itu, FL dapat menjadi solusi untuk menghasilkan model deteksi yang akurat dan efektif sekaligus menjaga kerahasiaan data. Sejumlah penelitian sebelumnya juga telah menerapkan FL untuk klasifikasi kanker payudara, sebagaimana dirangkum dalam Tabel 2.2.

Tabel 2.2. Penelitian Federated Learning dalam Bidang Kesehatan Sebelumnya

| **Penulis** | **Judul** | **ACC (%)** | **AUC (%)** | **PR-AUC (%)** | **Referensi** |
|---|---|---|---|---|---|
| Y. Zhang et al. (2021) | *Federated deep learning for mammogram image analysis* | - | 89 | 84 | [18] |
| L. Li, et al. (2022) | *A Federated Learning Framework for Breast Cancer Histopathological Image Classification* | 82.65-86.03 | – | – | [10] |
| I. Adjei-Mensah, et al. (2024) | *Cov-Fed: Federated learning-based framework for COVID-19 diagnosis using chest X-ray scans* | 87.65 | 92.46 | – | [19] |
| H. Chai, et al. (2024) | *A decentralized federated learning-based cancer survival prediction method with privacy protection* | 58.9 | 57.8 | – | [20] |
| A. Jiménez-Sánchez, et al. (2020) | *Memory-aware curriculum federated learning for breast cancer classification* | – | 79 | 82 | [5] |
| N. S. Joynab, et al. (2024) | *A federated learning aided system for classifying cervical cancer using PAP-SMEAR images* | 82.18 | – | – | [21] |

### 2.3.1 Federated Learning Berdasarkan Partisi Data

#### A Horizontal Federated Learning (HFL)




Gambar 2.5. Horizontal Federated Learning

*Horizontal Federated Learning (HFL)*, juga dikenal sebagai *sample-based FL* atau *homogeneous FL*, terjadi ketika himpunan data dari berbagai sumber memiliki ruang fitur yang sama, namun berbeda pada sampel yang dimiliki [22, 23] . Kondisi ini umum ditemui ketika organisasi atau perangkat memiliki data dengan jenis yang sama, tetapi berasal dari pengguna yang berbeda. Misalnya, sejumlah rumah sakit dapat bekerja sama melatih model menggunakan rekam medis pasien, di mana setiap rumah sakit memiliki kumpulan pasien yang berbeda namun dengan set fitur medis yang identik [22].

HFL sering disebut sebagai arsitektur paling umum dalam federated learning karena memungkinkan penggunaan model yang sama pada semua klien, sehingga mempermudah proses agregasi di server pusat. Keuntungan utama dari pendekatan ini adalah mendukung pembelajaran yang independen di tiap klien sekaligus meningkatkan aspek keamanan. Agar konvergensi model lebih baik, beberapa algoritme seperti MIME mengusulkan pengiriman tambahan berupa gradien lokal maupun statistik lain dari klien ke server [23].

Secara konsep, HFL juga dikenal sebagai *sample-partitioned* atau *example-partitioned federated learning*, di mana data dipartisi secara horizontal seperti dalam tabel basis data yang baris mewakili sampel dan setiap baris memiliki seluruh fitur. Sebagai ilustrasi, dua bank regional mungkin memiliki kelompok nasabah yang berbeda, dengan tumpang tindih pengguna yang kecil, tetapi tetap bisa berkolaborasi melatih model karena struktur data dan model bisnis mereka serupa. Dengan demikian, HFL menjadi kerangka penting yang memungkinkan

kolaborasi antar organisasi atau perangkat dengan data homogen namun berbeda entitas [24].

### 2.3.2 Federated Average

FedAvg adalah metode agregasi terpusat yang didasarkan pada algoritma Stochastic Gradient Descent (SGD), yaitu algoritma optimisasi yang digunakan untuk memperkirakan parameter model agar sesuai sebaik mungkin dengan perbedaan antara keluaran prediksi dan keluaran sebenarnya. Pada dasarnya, FedAvg bekerja dengan menghitung rata-rata dari pembaruan parameter model yang dikirimkan oleh semua pihak (klien) yang berpartisipasi dalam federasi [25]. Berikut adalah pseudocode dari algoritma Federated Averaging (FedAvg) yang digunakan untuk menggabungkan pembaruan model dari setiap klien ke model global:

**Algorithm 1 Federated Averaging** (Global Model Aggregation) [5]

**Server executes:**
Initialize global model parameters $w^{(0)}$
**for** each communication round $q = 1, 2, \ldots, Q$ **do**
  **for** each client $i = 1, \ldots, N$ **in parallel do**
    Receive updated local model $w_i^{(q)}$
  **end for**
  **if** $q \% \tau = 0$ **then**
    Aggregate global model:

$$w^{(q)} \leftarrow \frac{1}{N} \sum_{i=1}^{N} w_i^{(q)}$$

    **for** each client $i = 1, \ldots, N$ **do**
      Deploy global model to client:

$$w_i^{(q)} \leftarrow w^{(q)}$$

    **end for**
  **end if**
**end for**
**return** global model $w^{(Q)}$

- $N$ jumlah klien yang berpartisipasi dalam sistem *Federated Learning*.

- $Q$ menyatakan jumlah total ronde komunikasi global.
- $q$ menyatakan indeks ronde komunikasi, dengan $q = 1, 2, \ldots, Q$.
- $w^{(q)}$ menyatakan parameter model global pada ronde ke-$q$.
- $w_i^{(q)}$ menyatakan parameter model lokal milik klien ke-$i$ pada ronde ke-$q$.
- $i$ menyatakan indeks klien, dengan $i = 1, 2, \ldots, N$.
- $\tau$ menyatakan interval agregasi model global.
- $\frac{1}{N}\sum_{i=1}^{N} w_i^{(q)}$ menyatakan proses *Federated Averaging* untuk membentuk model global dari parameter model lokal.

Algoritma *Federated Averaging* digunakan untuk melakukan agregasi parameter model global pada skema *Federated Learning*. Pada setiap *communication round*, masing-masing klien melakukan pelatihan lokal menggunakan data privat yang dimilikinya dan menghasilkan parameter model lokal. Parameter-parameter lokal tersebut kemudian dikirimkan ke server untuk diagregasi menggunakan rata-rata aritmatika guna memperoleh model global yang diperbarui. Model global selanjutnya didistribusikan kembali ke seluruh klien untuk digunakan pada proses pelatihan di ronde berikutnya. Pendekatan ini memungkinkan pembelajaran kolaboratif tanpa perlu memindahkan data mentah dari klien ke server [5].

### 2.3.3 Curriculum Learning

*Curriculum Learning* (CL) merupakan pendekatan pembelajaran yang dirancang untuk memperkuat kemampuan generalisasi model dengan menyajikan data pelatihan secara bertahap, dimulai dari yang mudah hingga yang lebih kompleks. Metode ini telah terbukti efektif dalam meningkatkan performa model pada beragam permasalahan pembelajaran mesin. [26, 27]. CL memiliki sebuah fungsi penilaian (*scoring function*) didefinisikan untuk menentukan prioritas tiap sampel pelatihan, fungsi ini dapat mengukur tingkat kesulitan. Berdasarkan nilai skor tersebut, sampel pelatihan akan diberikan bobot atau dihadirkan ke optimizer dalam urutan tertentu. Dengan menyusun data pelatihan dalam urutan atau bobot yang tepat, metode CL dapat mempengaruhi jenis minimum lokal yang dicapai oleh optimizer, yang pada akhirnya dapat meningkatkan akurasi klasifikasi [5].

Metode ini memanfaatkan mekanisme perhatian atau ketidakpastian prediksi model sebagai dasar penentuan tingkat kesulitan sampel. Berbeda dengan penelitian sebelumnya yang banyak menggabungkan CL dengan teknik *data augmentation* untuk meningkatkan kemampuan transfer antar domain, penerapan CL dalam *Federated Learning* (FL) difokuskan pada penjadwalan data di dalam setiap klien secara lokal. Melalui konsep *temporal ensembling* dan *consistency training*, performa model dilacak dari satu epoch ke epoch berikutnya untuk mengidentifikasi sampel yang terlupakan oleh model lokal. Sampel-sampel tersebut kemudian diberi bobot lebih tinggi agar lebih sering digunakan selama pelatihan berikutnya. Dengan demikian, CL dalam FL berfungsi sebagai mekanisme memory-aware data scheduling yang menjaga konsistensi antara model lokal dan global, sekaligus meningkatkan kemampuan adaptasi terhadap heterogenitas data antar-klien [5].

Bobot kurikulum ditentukan berdasarkan perubahan performa model terhadap setiap sampel pelatihan dari satu epoch ke epoch berikutnya. Prediksi model pada epoch sebelumnya dibandingkan dengan prediksi pada epoch saat ini untuk mengidentifikasi dinamika pembelajaran. Apabila performa model terhadap suatu sampel mengalami penurunan, yang mengindikasikan bahwa sampel tersebut semakin sulit atau cenderung terlupakan oleh model, maka sampel tersebut diberikan bobot yang lebih tinggi. Sebaliknya, sampel dengan performa yang stabil atau meningkat diberikan bobot yang lebih rendah.

Setelah melewati tahap awal pelatihan, mekanisme pembobotan ini diterapkan secara lokal pada masing-masing klien dalam pembentukan ulang data pelatihan. Sampel dengan bobot lebih tinggi akan memiliki peluang lebih besar untuk dipilih pada proses pelatihan berikutnya, sehingga model diarahkan untuk lebih fokus pada sampel yang sulit. Pendekatan ini memungkinkan setiap klien menyesuaikan strategi pembelajaran terhadap karakteristik data lokalnya secara adaptif, menjaga konsistensi antara model lokal dan global, serta mendukung prinsip privasi dengan tetap memproses data secara lokal tanpa pertukaran data mentah.

### 2.3.4 Adversarial Alignment

Metode *Federated Adversarial Alignment* digunakan untuk mengatasi perbedaan distribusi data atau *domain shift* yang timbul akibat perbedaan perangkat pencitraan maupun protokol medis antar institusi. Pendekatan ini bertujuan menyelaraskan ruang fitur antar domain tanpa membagikan data mentah, sehingga

tetap menjaga privasi pasien. Setiap klien dalam sistem memiliki dua komponen utama, yaitu *feature extractor* dan *discriminator*. *Feature extractor* bertugas mengekstraksi representasi laten dari citra medis, sedangkan *discriminator* berperan membedakan apakah suatu fitur berasal dari domain lokal atau domain lain [5].

Selama proses pelatihan, representasi fitur dari masing-masing klien ditambahkan dengan *Gaussian noise* untuk menjaga privasi, lalu digunakan dalam proses penyelarasan fitur. *Discriminator* dilatih untuk membedakan domain asal representasi, sementara *feature extractor* berusaha menghasilkan fitur yang membuat *discriminator* gagal mengenali domainnya. Dengan mekanisme ini, model belajar menghasilkan distribusi fitur yang konsisten di seluruh domain. Proses pelatihan dilakukan secara adversarial dalam dua tahap: pertama, melatih *discriminator* agar dapat membedakan domain; kedua, melatih *feature extractor* agar menipu *discriminator* dengan membuat representasi antar domain semakin seragam. Pendekatan ini memungkinkan model global beradaptasi lebih baik terhadap variasi data dari berbagai sumber sekaligus mempertahankan kerahasiaan data medis [5].

```
nn = []
noises = []
for i in range(n_sites):
    nn = tdist.Normal(torch.tensor([0.0]), 0.001 * torch.std(fs[i
    ].detach().cpu()))
    noises.append(nn.sample(fs[i].size()).squeeze().to(device))

for i in range(n_sites):
    for j in range(n_sites):
        if i != j:
            optimizerDs[i].zero_grad()
            optimizerGs[i].zero_grad()
            optimizerGs[j].zero_grad()


            d1 = discriminators[i](fs[i].detach() + noises[i])
            d2 = discriminators[i](fs[j].detach() + noises[j])
            num_dataG[i] += d1.size(0)
            num_dataD[i] += d1.size(0)
            lossD = advDloss(d1, d2)
            lossG = advGloss(d1, d2)

            lossD_all[i] += lossD.item() * d1.size(0)
            lossG_all[i] += lossG.item() * d1.size(0)
```

```
            lossG_all[j] += lossG.item() * d2.size(0)
            lossD = 0.1 * lossD

        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 2.4: Proses *adversarial alignment* pada pelatihan federated learning

Pada mekanisme *adversarial alignment*, setiap klien memiliki komponen *discriminator* yang bertugas membedakan apakah suatu representasi fitur berasal dari domain lokal atau dari domain klien lain. Pendekatan ini mengikuti prinsip *adversarial learning*, di mana model pengekstrak fitur berperan menghasilkan representasi yang sulit dibedakan antar domain, sementara *discriminator* berusaha mengidentifikasi asal domain dari representasi tersebut [5].

Proses pelatihan diawali dengan penambahan gangguan acak (*Gaussian noise*) pada representasi fitur untuk meningkatkan robustitas model serta mengurangi risiko *overfitting*. Selanjutnya, representasi fitur dari klien sendiri dan klien lain digunakan secara bersamaan dalam proses pembelajaran *adversarial*. *Discriminator* dilatih untuk membedakan perbedaan distribusi fitur antar domain, sedangkan pengekstrak fitur dilatih untuk menghasilkan representasi yang semakin seragam sehingga menyulitkan proses pembedaan domain [5].

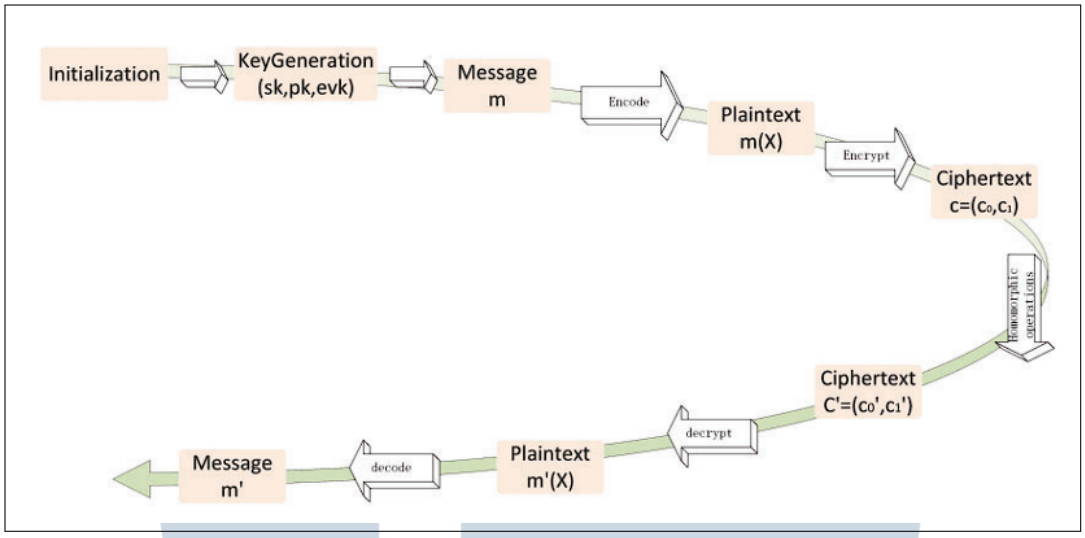
Mekanisme pembaruan parameter pada *adversarial alignment* dilakukan setelah tahap awal pelatihan, ketika model telah mencapai tingkat stabilitas tertentu. Dengan skema ini, proses penyelarasan tidak mengganggu fase pembelajaran awal, melainkan secara bertahap mendorong terbentuknya representasi fitur yang bersifat *domain-invariant*. Akibatnya, model global yang dihasilkan memiliki kemampuan generalisasi yang lebih baik terhadap variasi data antar institusi, sekaligus tetap menjaga prinsip privasi karena seluruh proses dilakukan tanpa pertukaran data mentah [5].

## 2.4 Segmented CKKS Homomorphic Encryption

*Homomorphic Encryption* (HE) adalah metode kriptografi yang memungkinkan perhitungan dilakukan langsung pada data terenkripsi tanpa

perlu melakukan dekripsi terlebih dahulu. Hasil dari perhitungan tersebut juga tetap dalam bentuk terenkripsi dan hanya dapat diakses oleh pihak yang memiliki *secret key* asli. Dengan kemampuannya menjaga privasi selama proses komputasi, HE menjadi komponen penting dalam berbagai aplikasi yang membutuhkan keamanan tinggi seperti layanan cloud, sistem pemungutan suara elektronik, dan kolaborasi data lintas institusi [28]. Berdasarkan kompleksitas operasinya, HE dapat dibagi menjadi tiga kategori utama: *Partial Homomorphic Encryption* (PHE) yang mendukung satu jenis operasi (penjumlahan atau perkalian), *Leveled Homomorphic Encryption* (LHE) yang memungkinkan operasi terbatas sesuai tingkat kedalaman perkalian, dan *Fully Homomorphic Encryption* (FHE) yang mendukung operasi penjumlahan dan perkalian tanpa batasan [29].

Salah satu skema *Fully Homomorphic Encryption* (FHE) yang paling efisien untuk komputasi numerik adalah *CKKS (Cheon–Kim–Kim–Song)*. CKKS diperkenalkan oleh Cheon *et al.* untuk mendukung operasi aritmetika *approximate* pada bilangan real dan kompleks. Berbasis pada konsep *Ring Learning With Errors (RLWE)*, CKKS memetakan vektor bilangan kompleks menjadi representasi polinomial melalui proses *encoding*, sehingga memungkinkan operasi matematika dilakukan langsung di dalam ruang polinomial terenkripsi. Setiap hasil operasi, baik penjumlahan maupun perkalian, tetap dalam bentuk terenkripsi hingga tahap dekripsi dilakukan oleh pemegang kunci rahasia [30]. Karena kemampuannya ini, CKKS banyak diterapkan pada skenario seperti *Federated Learning*, di mana beberapa institusi dapat melatih model secara kolaboratif tanpa perlu berbagi data asli [29].




Gambar 2.6. Alur kerja umum skema CKKS, meliputi proses inisialisasi hingga *decoding*. [30]

Secara umum, alur kerja CKKS ditunjukkan pada Gambar 2.6. Skema ini terdiri atas beberapa tahap utama, yaitu inisialisasi, *key generation*, *encoding*, enkripsi, operasi homomorfik, *rescaling*, dekripsi, dan *decoding*. Pada tahap inisialisasi, sistem menentukan parameter keamanan, dimensi cincin, serta modulus yang akan digunakan. Selanjutnya, proses *key generation* menghasilkan pasangan *public key* dan *secret key*, beserta kunci tambahan untuk mendukung operasi perkalian pada data terenkripsi. Tahap *encoding* mengubah data kompleks menjadi representasi polinomial yang sesuai untuk proses enkripsi, sedangkan tahap enkripsi menghasilkan *ciphertext* melalui kombinasi antara *public key* dan derau acak.

Setelah data terenkripsi, operasi aritmetika dapat dilakukan langsung pada *ciphertext*. Operasi penjumlahan dilakukan dengan menjumlahkan komponen polinomial dari dua *ciphertext*, sedangkan operasi perkalian memerlukan proses tambahan berupa *rescaling* untuk mengontrol pertumbuhan derau agar hasil dekripsi tetap akurat. Akhirnya, tahap dekripsi dan *decoding* mengembalikan hasil komputasi ke bentuk semula, membentuk siklus tertutup *encrypted computation–decrypted restoration* yang mendukung komputasi aman dan efisien pada data terenkripsi.

Penggunaan CKKS pada model berskala besar masih menghadapi kendala efisiensi karena keterbatasan jumlah elemen maksimum yang dapat dienkripsi dalam satu vektor, yaitu 16.384 elemen. Untuk mengatasi hal tersebut, Pan *et al.* (2024) memperkenalkan metode *Segmented CKKS Homomorphic Encryption* [6]. Pendekatan ini membagi vektor panjang menjadi beberapa segmen lebih kecil

sebelum proses enkripsi dilakukan, sehingga setiap segmen dapat diproses secara paralel tanpa melampaui batas kapasitas enkripsi. Dengan cara ini, Segmented CKKS memungkinkan penerapan enkripsi homomorfik pada model dengan jumlah parameter yang jauh lebih besar. Metode *Segmented CKKS Homomorphic Encryption* terdiri atas empat tahap utama, yaitu *depth-adaptive key generation*, *segmented encryption*, *segmented decryption*, dan *remove padding*. Setiap tahap dirancang untuk mengatasi keterbatasan ukuran vektor pada skema CKKS konvensional sekaligus menjaga efisiensi dan keamanan proses enkripsi. Berikut merupakan daftar dan penjelasan singkat dari setiap variabel yang digunakan pada tiga algoritma, yaitu *CKKS Key Generation*, *Segmented Encryption*, dan *Segmented Decryption*.

- $d$: merupakan *multiplication depth*, yaitu kedalaman maksimum operasi perkalian homomorfik yang dapat dilakukan sebelum tingkat presisi ciphertext menurun secara signifikan.
- $s$: menyatakan tingkat keamanan (*security level*) dalam satuan bit, misalnya 128-bit atau 192-bit, yang menentukan kompleksitas komputasi untuk memecahkan kunci privat.
- $ModDict$: merupakan *modulus chain dictionary* yang berisi kombinasi parameter CKKS (nilai modulus, derajat polinomial, serta panjang rantai modulus) untuk setiap pasangan $(s,d)$.
- $N$: *polynomial modulus degree*, menentukan jumlah slot yang dapat dienkripsi dalam satu ciphertext, di mana jumlah slot adalah $N/2$.
- $MC$: *modulus chain*, yaitu urutan nilai modulus (contohnya [60, 40, 40, 40, 60]) yang mengatur tingkat presisi dan kedalaman multiplikasi.
- $P_k$ dan $S_k$: masing-masing merupakan *public key* dan *secret key* hasil proses *key generation* menggunakan parameter $(N,MC)$.
- $V$: vektor data asli (*plain vector*) yang akan dienkripsi, misalnya representasi bobot model atau gradien lokal.
- $ARR_{seg}$: array hasil segmentasi ciphertext yang menyimpan potongan-potongan hasil enkripsi dari $V$.

- $l_v$: panjang dari vektor $V$, digunakan untuk menentukan jumlah segmen yang diperlukan.
- $l_g$: jumlah segmen yang dibentuk, dihitung sebagai $l_g = \lceil l_v/(N/2) \rceil$.
- $V_g$: potongan vektor $V$ pada indeks tertentu dengan panjang maksimum $N/2$, yang akan dienkripsi menjadi satu ciphertext.
- $ARR_{se}$: array ciphertext hasil proses enkripsi tersegmentasi yang akan digunakan untuk dekripsi.
- *Sk*: kunci privat (*secret key*) yang digunakan untuk mendekripsi setiap segmen dalam $ARR_{se}$.
- $l$: panjang asli dari vektor $V$, digunakan untuk memangkas hasil konkatenasi agar sesuai dengan ukuran data awal.

### 2.4.1 CKKS Key Generation

Tahap awal pada mekanisme *Segmented CKKS* atau *Depth-adaptive CKKS* adalah proses (*key generation*) yang menyesuaikan parameter enkripsi berdasarkan kedalaman perkalian (*multiplication depth*) dan tingkat keamanan (*security level*). Pada tahap ini, sistem menentukan derajat polinomial (*polynomial modulus degree*) yang paling sesuai agar kedua parameter tersebut dapat terpenuhi secara optimal. Untuk mendukung proses ini, dibentuk struktur data bernama *Modulus Chain Dictionary* (*ModDict*), yaitu kamus tiga tingkat yang menyimpan kombinasi parameter enkripsi. Kunci tingkat pertama menunjukkan tingkat keamanan, tingkat kedua menunjukkan kedalaman perkalian, dan tingkat ketiga menunjukkan derajat polinomial, dengan nilai berupa daftar rantai modulus (*modulus chains*) yang sesuai [6].

Berdasarkan struktur ini, sistem secara otomatis memilih konfigurasi terbaik dan menghasilkan pasangan kunci publik ($P_k$) dan kunci privat ($S_k$) yang efisien serta memenuhi tingkat keamanan yang ditentukan. Algoritma 2 menunjukkan proses pembangkitan kunci adaptif (*Depth-adaptive CKKS Key Generation*), di mana sistem secara otomatis memilih kombinasi parameter optimal dari struktur *ModDict* dan kemudian menghasilkan pasangan kunci publik ($P_k$) dan kunci privat ($S_k$). Pendekatan adaptif ini memastikan bahwa proses enkripsi tetap efisien tanpa mengorbankan tingkat keamanan.

**Algorithm 2** Depth-adaptive CKKS Key Generation Algorithm *(AutoKeyGen(d, s, ModDict))* [6]

**Require:** multiplication depth $d$, security level $s$, module chain dictionary $ModDict$

**Ensure:** public key $P_k$, private key $S_k$

$D_{arr} \rightarrow$ keys of $ModDict[s]$
$Max_d \rightarrow \max(D_{arr})$
**if** $d > Max_d$ **or** $d < 0$ **then**
    **break**
**end if**
$N_{arr} \rightarrow$ keys of $ModDict[s][d]$
$N \rightarrow \min(N_{arr})$
Modulus Chain $MC \rightarrow ModDict[s][d][N]$
$(P_k, S_k) \rightarrow HE.KeyGen(N, MC)$
**return** $N, P_k, S_k$

Tahapan pada algoritma tersebut dimulai dengan membaca struktur *ModDict* untuk memperoleh daftar tingkat kedalaman yang tersedia sesuai tingkat keamanan yang diminta. Jika kedalaman perkalian yang dimasukkan tidak valid, maka proses dibatalkan. Apabila valid, sistem kemudian menyeleksi derajat polinomial terkecil ($N$) yang sesuai, membentuk rantai modulus (*modulus chain*) yang sesuai dari *ModDict*, dan menjalankan fungsi `HE.KeyGen()` untuk menghasilkan pasangan kunci. Hasil akhir berupa pasangan kunci publik dan privat yang dapat digunakan dalam proses enkripsi CKKS selanjutnya. Implementasi dari algoritma ini diintegrasikan ke dalam sistem FedSHE melalui pemanggilan fungsi `generate_ckks_key()` ketika mode operasi yang digunakan adalah CKKS [6].

```
if args.mode == "CKKS":
    HE = generate_ckks_key(args.ckks_sec_level,
                           args.ckks_mul_depth,
                           args.ckks_key_len)
```

Kode 2.5: Pemanggilan fungsi generate_ckks_key() pada mode CKKS di FedSHE

Ketika argumen mode yang diterima bernilai `"CKKS"`, sistem akan memanggil fungsi `generate_ckks_key()` dengan tiga parameter utama, yaitu tingkat keamanan, kedalaman perkalian, dan panjang kunci polinomial. Pemanggilan ini memicu proses pembentukan konteks CKKS dan pembangkitan pasangan kunci publik serta kunci privat secara otomatis. Potongan kode berikut memperlihatkan implementasi dari fungsi `generate_ckks_key()` yang digunakan dalam proses tersebut [6].

```
with open('ModDict.json', 'r') as fcc_file:
    schemeDict = json.load(fcc_file)

def generate_ckks_key(sec_level, mul_depth, poly_moduls_degree):
    HE = Pyfhel()
    ckks_params = schemeDict.get(sec_level, {}) \
                            .get(mul_depth, {}) \
                            .get(poly_moduls_degree, {})
    print("ckks_params:", ckks_params)
    status = HE.contextGen(**ckks_params)
    print(status)
    HE.keyGen()
    return HE
```

Kode 2.6: Implementasi fungsi generate_ckks_key() pada FedSHE

Fungsi tersebut memanfaatkan pustaka `Pyfhel` untuk membentuk konteks dan menghasilkan kunci enkripsi CKKS berdasarkan parameter yang diambil dari *file* `ModDict.json`. Struktur *ModDict* berperan sebagai kamus tiga tingkat yang menyimpan kombinasi parameter untuk setiap tingkat keamanan, kedalaman perkalian, dan derajat polinomial.

Tingkat keamanan (*security level*) menentukan ukuran modulus dan tingkat perlindungan terhadap serangan kriptografis, kedalaman perkalian (*multiplication depth*) menunjukkan batas maksimum operasi perkalian bertingkat sebelum derau (*noise*) melebihi ambang dekripsi, sedangkan derajat polinomial (*polynomial modulus degree*) menentukan kapasitas elemen kompleks dalam satu ciphertext. Dengan konfigurasi parameter yang dapat menyesuaikan kebutuhan komputasi dan keamanan, sistem FedSHE mampu mencapai efisiensi tinggi tanpa mengorbankan keamanan, sehingga pendekatan ini disebut *Segmented CKKS*.

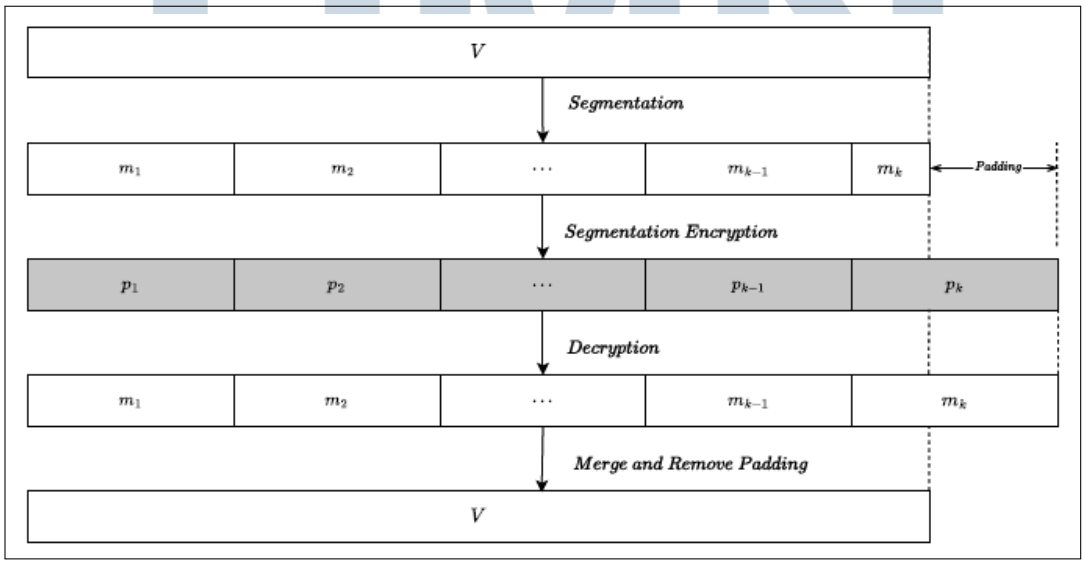
### 2.4.2 Segmented Encryption

**Algorithm 3** Segmented Encryption [6]

**Require:** plain vector: $V$, public key: $Pk$, polynomial modulus degree: $N$

**Ensure:** segmented encryption array: $ARR_{seg}$

init $ARR_{seg}$
$l_v \leftarrow$ length of $V$
**if** $l_v < N/2$ **then**
    $ARR_{seg}$.append($HE.Enc(V, Pk)$)
**else**
    $l_g \leftarrow \lceil l_v/(N/2) \rceil$
    **for** $i = 0$ to $(l_g - 1)$ **do**
        $start\_index \leftarrow (N/2) \times i$
        $end\_index \leftarrow (N/2) \times (i+1)$
        **if** $end\_index > l_v$ **then**
            $end\_index \leftarrow l_v$
        **end if**
        $V_g \leftarrow V[start\_index : end\_index]$
        $ARR_{seg}$.append($HE.Enc(V_g, Pk)$)
    **end for**
**end if**
**return** $ARR_{seg}$




Gambar 2.7. Proses Encryption Segmented CKKS

Algoritma 3 menggambarkan proses pembagian dan enkripsi vektor panjang agar sesuai dengan kapasitas enkripsi CKKS. Jika panjang vektor $V$ lebih kecil dari setengah derajat polinomial $(N/2)$, maka vektor dapat dienkripsi langsung tanpa pembagian. Sebaliknya, jika panjang $V$ melebihi batas tersebut, vektor akan dibagi menjadi beberapa segmen $m_i$ berukuran $N/2$, di mana segmen terakhir $m_k$ memiliki panjang $len(V)$ mod $(N/2)$. Sesuai dengan standar pengkodean CKKS, jika panjang suatu segmen kurang dari $N/2$, maka akan dilakukan proses *padding* dengan nilai nol hingga mencapai panjang $N/2$. Setelah proses segmentasi, setiap segmen dienkripsi secara terpisah menggunakan kunci publik $P_k$, menghasilkan himpunan *ciphertext* $[p_i]_{i=1...k}$. Pendekatan ini memungkinkan sistem untuk mengenkripsi data berdimensi besar tanpa melampaui kapasitas slot CKKS, sehingga mendukung efisiensi dan fleksibilitas dalam proses enkripsi terdistribusi [6].

```
elif self.args.mode == 'CKKS':  # CKKS Encryption
    print('CKKS encrypting...')
    enc_start = time.time()
    for k in w_new.keys():
        update_w[k] = w_new[k] - w_old[k]
        list_w = update_w[k].view(-1).cpu().tolist()
        list_w = np.array(list_w)
        vec_len = len(list_w)
        if len(list_w) <= self.HE.get_nSlots():
            plist_w = enc_vector(self.HE, list_w)
            update_w[k] = plist_w
        else:
            plist_w_arr = seg_enc_vector(self.HE, list_w, vec_len)
            update_w[k] = plist_w_arr
    enc_end = time.time()
    print('Encryption time:', colored((enc_end - enc_start), '
   green'))
```

Kode 2.7: Pemanggilan fungsi seg_enc_vector() pada mode CKKS di client.py

Implementasi algoritma 3 dilakukan pada sisi *client* untuk mengenkripsi parameter model sebelum dikirim ke server federasi. Potongan kode berikut memperlihatkan proses pemanggilan fungsi `seg_enc_vector()` pada mode CKKS di *file* `client.py`. Fungsi `enc_vector()` dan `seg_enc_vector()` yang digunakan dalam proses tersebut diimplementasikan pada *file* `SegCKKS.py`, seperti ditunjukkan pada potongan kode berikut [6].

```
def enc_vector(HE, arr_x):
```

```
    ptxt_x = HE.encodeFrac(arr_x)
    ctxt_x = HE.encryptPtxt(ptxt_x)
    return ctxt_x

def seg_enc_vector(HE, vector, vecl):
    block_enc_arr = []
    block_len = HE.get_nSlots()
    block_arr_len = math.ceil(vecl / block_len)
    for i in range(block_arr_len):
        start_index = block_len * i
        end_index = block_len * (i+1)
        if end_index > vecl:
            end_index = vecl
        vector_block = vector[start_index:end_index]
        enc_vector_block = enc_vector(HE, vector_block)
        block_enc_arr.append(enc_vector_block)
    return block_enc_arr
```

Kode 2.8: Implementasi fungsi enc_vector() dan seg_enc_vector() di SegCKKS.py

Kedua fungsi tersebut memanfaatkan pustaka `Pyfhel` untuk melakukan proses *encoding* dan *encryption*. Fungsi `enc_vector()` mengenkripsi satu blok vektor, sedangkan `seg_enc_vector()` memecah vektor besar menjadi beberapa blok dan mengenkripsi masing-masing blok secara terpisah. Pendekatan ini memastikan seluruh parameter model dapat dienkripsi meskipun jumlahnya melampaui kapasitas satu ciphertext CKKS.

### 2.4.3 Segmented Decryption

**Algorithm 4** Segmented Decryption [6]

**Require:** encryption array: $ARR_{se}$, private key: $sk$, vector length: $l$

**Ensure:** plain vector: $V$

1: init array: $varr$
2: **for** encv in $ARR_{se}$ **do**
3: $varr$.append($HE.Dec(encv, sk)$)
4: **end for**
5: $V \leftarrow concatenate(varr)$
6: $V \leftarrow V[0:l]$
7: **return** $V$

Algoritma 4 menjelaskan proses dekripsi tersegmentasi (*segmented decryption*) untuk mengembalikan hasil enkripsi vektor panjang yang telah dibagi menjadi beberapa segmen. Setiap *ciphertext* dalam array terenkripsi ($ARR_{se}$) didekripsi secara terpisah menggunakan kunci privat (*sk*) dengan metode *HE.Dec*, menghasilkan kumpulan plaintext $m_i i = 1^k$. Seluruh hasil dekripsi kemudian digabungkan secara berurutan menggunakan operasi *concatenate* untuk membentuk kembali struktur vektor aslinya.

Karena pada proses enkripsi dilakukan *padding* untuk menyesuaikan ukuran segmen menjadi $N/2$, maka segmen terakhir dapat mengandung elemen tambahan bernilai nol. Oleh karena itu, setelah proses penggabungan, dilakukan pemangkasan pada hasil dekripsi menggunakan $V[0 : l]$ agar panjang vektor kembali sesuai dengan ukuran semula. Pendekatan ini memungkinkan sistem mengembalikan data terenkripsi secara efisien tanpa kehilangan informasi dan menjaga integritas bobot model setelah dekripsi [6].

```
elif self.args.mode == 'CKKS':
    update_w_avg = copy.deepcopy(w_glob)
    print('CKKS Decrypting...')
    dec_start = time.time()
    for k in update_w_avg.keys():
        origin_shape = list(self.model.state_dict()[k].size())
        enc_vec = update_w_avg[k]
        if isinstance(enc_vec, list):
            dec_vec = seg_dec_vector(self.HE, enc_vec)
        else:
            dec_vec = dec_vector(self.HE, enc_vec)
        vlen = np.prod(origin_shape)
        dec_vec = dec_vec[:vlen]  # remove padding
        update_w_avg[k] = dec_vec
        update_w_avg[k] = torch.FloatTensor(update_w_avg[k]).to(
    self.args.device).view(*origin_shape)
        update_w_avg[k] = torch.div(update_w_avg[k], self.args.
    num_users)
        self.model.state_dict()[k] += update_w_avg[k]
    dec_end = time.time()
    print('Decryption time:', colored(dec_end - dec_start, 'green'
    ))
```

Kode 2.9: Implementasi Segmented Decryption pada `client.py`

## 2.5 Evaluasi Performa dan Privasi

Evaluasi performa dan privasi merupakan aspek penting dalam penelitian yang melibatkan *Federated Learning* (FL) dengan penerapan skema *Homomorphic Encryption* (HE). Tujuan utama evaluasi ini adalah untuk mengukur sejauh mana sistem mampu mempertahankan akurasi model serta efisiensi pelatihan, sekaligus menjamin keamanan dan kerahasiaan data yang digunakan oleh masing-masing klien.

### 2.5.1 Evaluasi Accuracy, AUC, dan PR-AUC

Metrik *Accuracy* digunakan untuk mengukur proporsi prediksi yang diklasifikasikan dengan benar terhadap seluruh jumlah sampel. Nilai *Accuracy* dihitung berdasarkan jumlah prediksi benar, yaitu *True Positive* (TP) dan *True Negative* (TN), dibandingkan dengan total sampel yang terdiri dari TP, TN, *False Positive* (FP), dan *False Negative* (FN), sebagaimana ditunjukkan pada Persamaan 2.1.

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{2.1}$$

Evaluasi performa dilakukan untuk menilai kemampuan model dalam melakukan klasifikasi secara akurat dan efisien. Beberapa metrik yang digunakan antara lain *Area Under Curve* (AUC) dan *recision-Recall* AUC (PR-AUC), *communication cost* serta waktu komputasi. Metrik AUC yang diperoleh dari luas area di bawah kurva *Receiver Operating Characteristic (ROC)*. Kurva ROC memplot hubungan antara *True Positive Rate (TPR)* dan *False Positive Rate (FPR)* sebagaimana pada Persamaan 2.2 [31].

$$\text{TPR} = \frac{TP}{TP + FN}, \qquad \text{FPR} = \frac{FP}{FP + TN} \tag{2.2}$$

$$\text{AUC} = \int_0^1 \text{TPR}\, d(\text{FPR}) = 1 - \int_0^1 \text{FPR}\, d(\text{TPR}) \tag{2.3}$$

Semakin besar nilai AUC (mendekati 1), semakin baik kemampuan model dalam membedakan antara kelas positif dan negatif [31]. Selain AUC, metrik

Precision-Recall AUC (PR-AUC) juga digunakan untuk menilai performa model terutama pada data dengan distribusi label yang tidak seimbang [31]. Kurva ini memplot hubungan antara *Precision* dan *Recall*, yang didefinisikan sebagai berikut:

$$\text{Precision} = \frac{TP}{TP+FP}, \qquad \text{Recall} = \frac{TP}{TP+FN} \tag{2.4}$$

Nilai PR-AUC diperoleh dari integral antara *Precision* dan *Recall*:

$$\text{AUPRC} = \int_0^1 \text{Precision}\, d(\text{Recall}) \tag{2.5}$$

Nilai PR-AUC yang tinggi menunjukkan bahwa model memiliki kemampuan yang baik dalam mengidentifikasi kelas minoritas tanpa menghasilkan terlalu banyak kesalahan positif. Selain metrik klasifikasi, aspek efisiensi juga diukur melalui waktu pelatihan (training time). Waktu pelatihan dihitung berdasarkan total waktu komputasi pada sisi klien dan waktu sinkronisasi dengan server selama proses *federated aggregation*. Pengukuran ini penting untuk menganalisis sejauh mana penerapan federated learning dan enkripsi berdampak pada efisiensi pelatihan model secara keseluruhan [31].

### 2.5.2 Evaluasi Waktu Komputasi dan Communication Cost

Evaluasi privasi dalam *Segmented CKKS HE* berfokus pada dua aspek utama, yaitu waktu komputasi dan *communication cost*. Waktu komputasi mencakup waktu yang diperlukan untuk proses enkripsi dan dekripsi bobot model serta *training time local*, *transition time*, *aggregation time*. Sementara, biaya komunikasi (*communication cost*) yang timbul akibat penggunaan ciphertext [6]. Waktu total komputasi pada proses *Federated Learning* dengan penerapan *Homomorphic Encryption* dapat dihitung menggunakan Persamaan 2.6 [6].

$$T_{\text{total}} = T_{\text{local}} + T_{\text{enc}} + T_{\text{agg}} + T_{\text{dec}} \tag{2.6}$$

Persamaan tersebut merepresentasikan total waktu yang dibutuhkan selama satu *federated round*, di mana:

- $T_{\text{local}}$ adalah waktu pelatihan model pada sisi klien,

- $T_{\text{enc}}$ adalah waktu yang diperlukan untuk proses enkripsi model,
- $T_{\text{agg}}$ adalah waktu yang dibutuhkan untuk melakukan agregasi parameter terenkripsi di sisi server, dan
- $T_{\text{dec}}$ merupakan waktu dekripsi hasil agregasi sebelum pembaruan model dilakukan.

Dalam skema *Federated Learning* berbasis CKKS, terdapat redundansi pada *ciphertext*, sehingga ukuran data terenkripsi sangat bergantung pada pemilihan *polynomial modulus degree*. Misalkan $N$ merupakan *polynomial modulus degree*, $D$ merupakan *multiplication depth*, dan $L$ adalah jumlah parameter dari model CNN yang terdiri dari $n$ layer, dengan $L_i$ sebagai vektor parameter ter-flatten pada layer ke-$i$ [6].

Apabila *multiplication depth* bernilai 0, maka $D = 1$, dan meningkat seiring bertambahnya kedalaman operasi perkalian homomorfik. Oleh karena itu, biaya komunikasi setelah menerapkan skema CKKS dapat dihitung menggunakan Persamaan 2.7 berikut.

$$\text{COMM}_{\text{CKKS}} = \sum_{i=1}^{n} \left\lceil \frac{2L_i}{N} \right\rceil \times N \times 2 \times D \times 64 \text{ bit} \tag{2.7}$$

dengan:

- $L_i$ : jumlah parameter pada layer ke-$i$,
- $N$ : *polynomial modulus degree*,
- $D$ : *multiplication depth*.

Hasil dari perhitungan tersebut kemudian dikonversi ke dalam satuan megabyte (MB) untuk mengevaluasi efisiensi komunikasi yang terjadi pada setiap ronde federasi. Nilai ini memberikan gambaran mengenai overhead komunikasi yang ditimbulkan oleh penggunaan skema *Homomorphic Encryption* pada proses agregasi model [6].