BAB 2
LANDASAN TEORI

2.1 Federated Learning

Federated learning adalah paradigma *machine learning* terdistribusi yang memungkinkan pelatihan model kolaboratif tanpa mengumpulkan data ke lokasi terpusat. Dalam *federated learning*, setiap perangkat klien mempertahankan data lokalnya dan hanya mengirimkan pembaruan model ke server pusat, yang kemudian mengagregasi pembaruan dari semua klien untuk memperoleh model global [1]. Pendekatan ini berbeda dengan *machine learning* terpusat konvensional, di mana seluruh data dikumpulkan ke server tunggal untuk pelatihan [9].

Motivasi utama *federated learning* mencakup tiga aspek kritis, yaitu privasi data, efisiensi komunikasi, dan kepatuhan terhadap peraturan. Dari sisi privasi, pendekatan ini memungkinkan data sensitif seperti riwayat medis dan informasi finansial tetap berada di perangkat lokal tanpa perlu dikirim ke server pusat [10]. Selain itu, *federated learning* meningkatkan efisiensi komunikasi dengan hanya mengirimkan pembaruan model, bukan keseluruhan dataset sehingga mengurangi beban jaringan [1]. Di sisi regulasi, pendekatan ini juga mendukung kepatuhan terhadap aturan perlindungan data seperti General Data Protection Regulation (GDPR) yang membatasi transfer data lintas wilayah [9].

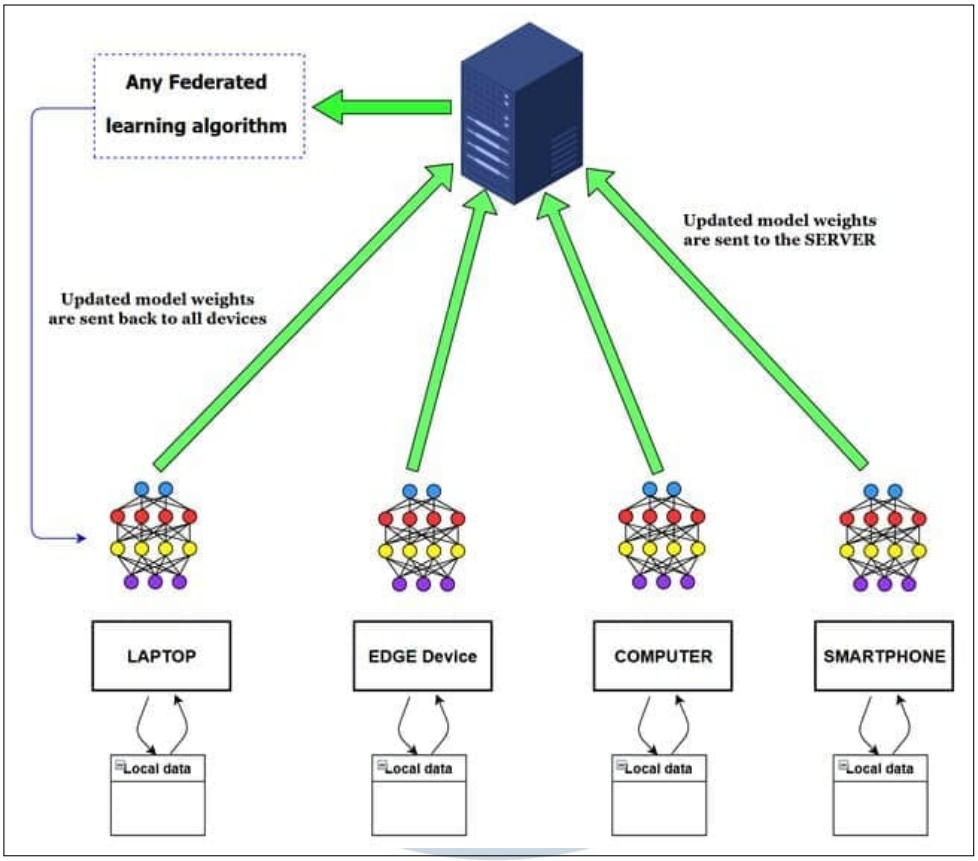
Dalam beberapa tahun terakhir, *federated learning* telah menarik perhatian luas dari komunitas penelitian maupun industri karena kemampuannya menjawab tantangan utama dalam pengelolaan data terdistribusi [11]. Seiring meningkatnya kekhawatiran terhadap privasi data dan keterbatasan dalam berbagi data lintas institusi, pendekatan ini menjadi solusi yang menjanjikan untuk berbagai aplikasi seperti kesehatan, keuangan, dan sistem cerdas berbasis perangkat *mobile* [12]. Selain itu, perkembangan perangkat komputasi di sisi klien serta kebutuhan akan pembelajaran yang adaptif dan skalabel, turut mendorong pesatnya eksplorasi metode, arsitektur, dan mekanisme keamanan dalam *federated learning*. Hal ini tercermin dari meningkatnya jumlah publikasi ilmiah yang membahas peningkatan performa, efisiensi komunikasi, dan perlindungan privasi dalam skenario *federated learning* [13]. Contoh penelitian terkait *federated learning* ditunjukkan pada Tabel 2.1.

Tabel 2.1. Penelitian *federated learning*

Judul	Penulis	Metode	Hasil
MaskCrypt: Federated Learning with Selective Homomorphic Encryption	Hu dan Li (2025) [4]	FL + SHE	Mengurangi *overhead* komunikasi hingga 4,15×, melindungi data pribadi klien dari serangan inversi, dan mengurangi akurasi serangan inferensi keanggotaan hingga 49,2%
SenseCrypt: Sensitivity-guided Selective Homomorphic Encryption for Joint Federated Learning in Cross-Device Scenarios	Li et al. (2025) [14]	FL + SHE	Memastikan keamanan terhadap serangan inversi terkini, mencapai akurasi model normal seperti pada data IID, dan mengurangi waktu pelatihan sebesar 58,4%-88,7%
SHE-LoRA: Selective Homomorphic Encryption for Federated Tuning with Heterogeneous LoRA	Liu et al. (2026) [15]	FL + SHE + LoRA	Mencapai ketahanan yang kuat terhadap serangan terkini, serta mengurangi *overhead* komunikasi hingga 99,71% dan waktu enkripsi hingga 99,87%

2.1.1 Arsitektur

Arsitektur *federated learning* terdiri dari dua komponen utama, yakni sejumlah besar perangkat klien (*klien*) dan satu server pusat (*central server*) [16]. Perangkat klien adalah entitas yang menyimpan data lokal dan melakukan komputasi pelatihan lokal. Sementara itu, server pusat bertugas dalam mengoordinasikan proses pelatihan, menerima pembaruan model dari klien, dan menjalankan agregasi untuk menghasilkan model global yang diperbarui [7]. Setiap klien dalam sistem *federated learning* diasumsikan memiliki dataset lokal yang tidak diketahui oleh klien lain dan tidak diakses oleh server pusat [17]. Data lokal di setiap klien dapat bervariasi dalam ukuran, distribusi, dan karakteristik statistiknya, yang mencerminkan heterogenitas data dalam skenario praktis [16]. Server pusat tidak memiliki akses langsung ke data lokal klien dan hanya berinteraksi dengan informasi yang dikirimkan oleh klien dalam bentuk pembaruan model [7].



Gambar 2.1. Arsitektur *federated learning*
Sumber: [18]

2.1.2 Proses Pelatihan

Pelatihan dalam *federated learning* berlangsung melalui serangkaian ronde komunikasi antara server pusat dan klien. Setiap ronde terdiri dari empat tahapan utama yang terkoordinasi berikut [1].

1. Server menginisialisasi atau memperbarui model global dan mengirimkannya ke sejumlah klien yang dipilih secara acak
2. Setiap klien terpilih melakukan pelatihan lokal pada dataset pribadinya masing-masing menggunakan model global yang diterima
3. Setiap klien mengirimkan pembaruan model lokal kembali ke server
4. Server mengagregasi pembaruan dari semua klien untuk menghasilkan model global yang diperbarui untuk ronde berikutnya

2.1.3 Tantangan

Implementasi *federated learning* dalam praktik nyata menghadapi beberapa tantangan berikut.

1. Heterogenitas data (non-IID). Data lokal pada klien yang berbeda sering kali tidak mengikuti distribusi identik dan independen (non-IID), menciptakan heterogenitas statistik [7]. Heterogenitas data statistik ini menyebabkan objektif optimasi lokal klien menjadi tidak konsisten dengan objektif global, mengakibatkan model lokal konvergen ke arah yang berbeda-beda dan mencapai optimal lokal alih-alih optimal global sehingga merendahkan akurasi *federated learning* [6].

2. Kerentanan privasi residual. Model yang dikirim klien tetap rentan terhadap serangan inferensi. Penyerang dapat melakukan serangan inferensi keanggotaan untuk menentukan apakah sampel data tertentu ada dalam dataset pelatihan lokal klien, atau serangan inversi untuk merekonstruksi data lokal dari gradien yang diterima [19]. Serangan kebocoran gradien juga menunjukkan bahwa informasi sensitif dapat diekstrak langsung dari gradien dalam beberapa iterasi pelatihan [20].

2.2 Framework Plato

Plato merupakan sebuah *framework open-source* yang dirancang khusus untuk mendukung penelitian dalam bidang *federated learning* secara skalabel, fleksibel, dan mudah dikembangkan. *Framework* ini dikembangkan untuk menjawab berbagai keterbatasan pada penelitian *federated learning* sebelumnya, khususnya terkait kurangnya standar implementasi yang terbuka dan sulitnya mereproduksi hasil eksperimen. Dalam banyak penelitian terdahulu, implementasi sistem seringkali tidak dipublikasikan secara terbuka sehingga evaluasi performa antar metode menjadi tidak konsisten dan sulit dibandingkan secara adil [21].

Sebagai solusi atas permasalahan tersebut, Plato dikembangkan dengan tujuan utama untuk menyediakan lingkungan penelitian *federated learning* yang terstandarisasi dan dapat digunakan kembali (*reproducible*). *Framework* ini memungkinkan peneliti untuk mengimplementasikan, menguji, dan membandingkan berbagai algoritma *federated learning* dalam satu platform yang sama dengan metrik evaluasi yang konsisten. Dengan demikian, Plato

berperan penting dalam meningkatkan transparansi dan validitas hasil penelitian di bidang *federated learning* [21].

Pengembangan Plato didasarkan pada empat tujuan desain utama yang komprehensif. Keempat tujuan tersebut dirancang agar *framework* dapat mengatasi berbagai tantangan dalam penelitian *federated learning* modern [21].

1. Skalabilitas terhadap jumlah klien yang tak terbatas
   Jumlah klien yang dapat diemulasikan dengan memori GPU terbatas ditingkatkan melalui eksekusi berbasis *batch*. Setiap klien menjalankan proses pelatihan dalam proses terpisah sehingga memori GPU dilepaskan setelah satu ronde selesai. Selain itu, Plato mampu memanfaatkan banyak GPU secara paralel untuk meningkatkan tingkat konkurensi.

2. Ekstensibilitas untuk berbagai dataset, model, dan algoritma
   Plato menyediakan berbagai model dan dataset *machine learning*, termasuk klasifikasi citra dan pemrosesan bahasa alami. Seluruh komponen dalam mekanisme *federated learning*, seperti algoritma, klien, server, *trainer* model, sumber data, dan *sampler* data (`iid` atau `noniid`), diimplementasikan sebagai kelas yang dapat diperluas. Desain berbasis *inheritance* dan *callback* memudahkan pengembangan mekanisme baru.

3. Implementasi dunia nyata, bukan hanya emulasi
   Di Plato, komunikasi antara server pusat, server *edge*, dan klien menggunakan WebSockets standar industri sehingga dapat digunakan baik untuk emulasi maupun implementasi nyata. Server dapat dijalankan di pusat data *cloud* atau pada mesin lokal yang sama dengan klien untuk keperluan eksperimen. Pendekatan ini memungkinkan pengukuran waktu pelatihan secara akurat, termasuk waktu komunikasi.

4. Reproduksibilitas dalam eksperimen *federated learning*
   Dalam eksperimen *federated learning*, generator bilangan acak digunakan untuk memilih klien dan data lokal. Agar perbandingan antar mekanisme tetap adil, reproduksibilitas dijaga dengan mengatur, menyimpan, dan memulihkan *seed*. Plato mendukung penetapan *seed* serta penggunaan `random.getstate()` dan `random.setstate()` untuk melindungi proses acak dari pengaruh *framework* lain.

2.3 Non-Independent and Identically Distributed (Non-IID)

Dalam *machine learning*, asumsi *independent and identically distributed* (IID) menyatakan bahwa setiap sampel data bersifat independen satu sama lain dan berasal dari distribusi probabilitas yang sama. Asumsi ini umum digunakan dalam pelatihan model terpusat karena mempermudah proses optimisasi dan analisis konvergensi [22]. Namun, dalam skenario *federated learning* praktis, asumsi IID sering kali tidak terpenuhi [7].

Data dikatakan non-IID ketika terjadi heterogenitas statistik, yaitu kondisi di mana distribusi data antar klien berbeda, baik dari segi distribusi label, fitur, maupun jumlah data yang dimiliki oleh masing-masing klien [7]. Kondisi ini muncul karena setiap klien mengumpulkan data secara mandiri sesuai dengan preferensi dan konteks lokalnya [8]. Akibatnya, data yang tersimpan pada setiap klien cenderung merepresentasikan distribusi lokal yang spesifik dan tidak mencerminkan distribusi global secara keseluruhan [23].

Dalam konteks *federated learning*, heterogenitas statistik menyebabkan model lokal yang dilatih pada masing-masing klien memiliki arah pembaruan yang berbeda-beda, bahkan dapat saling bertentangan [8]. Hal ini berbeda dengan kondisi IID, di mana pembaruan model dari setiap klien cenderung konsisten terhadap tujuan optimisasi global. Oleh karena itu, keberadaan heterogenitas statistik menjadi salah satu tantangan utama dalam pengembangan algoritma *federated learning*, karena dapat memengaruhi stabilitas pelatihan dan kualitas model global yang dihasilkan [7].

2.3.1 Kategori

Distribusi data non-IID dalam *federated learning* dapat dipahami sebagai adanya *skew* atau ketidakseimbangan distribusi data antar klien, yang mencakup perbedaan pada fitur (*attribute*), label, karakteristik temporal, dan kuantitas data. *Skew* (kemiringan) ini muncul karena setiap klien memiliki distribusi lokal $P_k(x,y)$ yang berbeda satu sama lain sehingga data tidak lagi berasal dari distribusi yang identik. Berdasarkan bentuk ketidakseimbangan tersebut, data non-IID dapat dikategorikan ke dalam beberapa jenis berikut [7].

1. *Attribute skew*

   Attribute skew menunjukkan skenario di mana distribusi fitur $P_k(x)$ di seluruh atribut pada setiap klien berbeda satu sama lain. Perbedaan ini dapat berupa sebagai berikut.

   (a) *Non-overlapping attribute skew*: Fitur antar klien saling eksklusif, seperti pada skenario *federated learning* vertikal, di mana setiap klien memiliki *subset* fitur yang berbeda.

   (b) *Partial overlapping attribute skew*: Sebagian fitur dibagikan antar klien, namun distribusinya dapat konsisten atau tidak konsisten.

   (c) *Full overlapping attribute skew*: Seluruh klien memiliki fitur yang sama, tetapi distribusinya berbeda, misalnya akibat variasi kualitas data atau kondisi lingkungan.

2. *Label skew*

   Label skew menggambarkan skenario di mana distribusi label berbeda dari satu klien ke klien lainnya. Terdapat dua bentuk utama berikut.

   (a) *Label distribution skew*: Perbedaan proporsi label pada masing-masing klien, misalnya satu klien hanya memiliki sebagian kelas tertentu.

   (b) *Label preference skew*: Perbedaan pelabelan untuk data yang sama, yang dapat terjadi akibat preferensi pengguna atau *noise* pada proses anotasi.

3. *Temporal skew*

   Temporal skew muncul ketika distribusi data berubah terhadap waktu, terutama pada data deret waktu (*time-series*) atau data bertanda waktu. Dalam hal ini, distribusi $P_k(x, y|t)$ pada setiap klien dapat berbeda oleh karena perbedaan periode pengumpulan data atau dinamika lingkungan seiring waktu.

4. *Attribute and label skew*

   Pada beberapa kasus, perbedaan tidak hanya terjadi pada fitur atau label secara terpisah, tetapi keduanya sekaligus. Kondisi ini menyebabkan kompleksitas yang lebih tinggi karena setiap klien dapat memiliki jenis data, fitur, dan label yang berbeda secara bersamaan sehingga menyulitkan proses agregasi model global.

5. *Quantity skew*

   Quantity skew terjadi saat jumlah data yang dimiliki setiap klien berbeda. Ketidakseimbangan ini dapat memperbesar pengaruh klien tertentu dalam proses agregasi dan berpotensi menyebabkan bias pada model global.

2.3.2 Simulasi dengan Distribusi Dirichlet

Pada penelitian *federated learning*, skenario non-IID dapat disimulasikan dengan memanfaatkan distribusi Dirichlet. Pendekatan ini menjadi metode standar dalam simulasi heterogenitas statistik karena memungkinkan kontrol yang presisi terhadap perbedaan distribusi label (*label distribution skew*) antar klien. Distribusi Dirichlet digunakan untuk mengatur proporsi masing-masing label pada setiap klien sehingga tiap klien memiliki komposisi kelas yang berbeda [6, 24].

Metode ini bekerja dengan menentukan rasio setiap label kelas dalam dataset global, kemudian menyebarkan rasio tersebut ke setiap klien. Secara formal, misalkan $p = \{p_c\}_{c \in C}$ merupakan rasio setiap label c dalam dataset global D, di mana C adalah himpunan semua label. Pada setiap klien n, vektor distribusi label didefinisikan sebagai:

$$q_n = \mathrm{Dir}(\alpha p) = \{q_{n,c}\}_{c \in C} \tag{2.1}$$

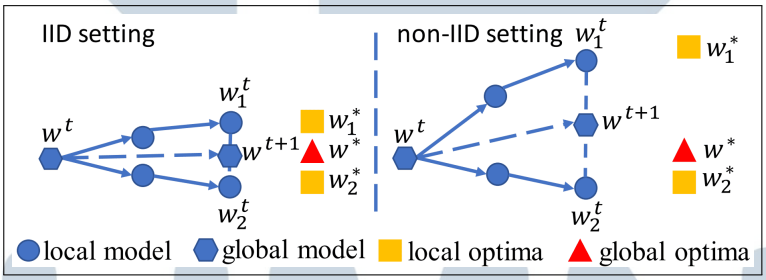
di mana $q_{n,c}$ menunjukkan proporsi data klien n yang berasal dari label c, $\alpha \geq 0$ adalah hiperparameter konsentrasi, dan $Dir(\cdot)$ adalah distribusi Dirichlet [24].

Hiperparameter konsentrasi α memiliki peran krusial dalam menentukan tingkat heterogenitas data. Nilai α yang semakin tinggi menghasilkan distribusi label pada setiap klien yang semakin mirip dengan distribusi global sehingga data menjadi semakin seimbang (mendekati kondisi IID). Sebaliknya, nilai α yang semakin rendah menyebabkan distribusi label pada tiap klien menjadi semakin berbeda sehingga memperkuat kondisi *label distribution skew* [6, 24]. Dalam praktik, nilai α umumnya berkisar antara $0,1$ hingga $1,0$, dengan $\alpha = 0,5$ sering digunakan sebagai nilai standar pada evaluasi algoritma *federated learning* dalam skenario non-IID yang moderat [25].

Keunggulan pendekatan distribusi Dirichlet terletak pada fleksibilitas dan kesederhanaan matematisnya. Peneliti dapat dengan mudah mengontrol tingkat *label distribution skew* dengan mengubah hiperparameter α tanpa memerlukan modifikasi metodologi yang kompleks. Oleh karena itu, pendekatan ini banyak digunakan untuk merepresentasikan kondisi dunia nyata dalam berbagai penelitian *federated learning* [6, 24].

2.3.3 Dampak pada Federated Learning

Keberadaan data yang terdistribusi secara non-IID di antara klien yang berpartisipasi merupakan salah satu tantangan dalam sistem *federated learning* karena dapat memengaruhi akurasi. Objektif lokal dari tiap klien menjadi tidak konsisten dengan optima global karena distribusi dari tiap dataset lokal sangat berbeda dari distribusi global. Maka dari itu, terdapat *drift* (penyimpangan) dalam pembaruan lokal yang berarti bahwa dalam tahap pelatihan lokal, model lokal diperbarui menuju optima lokal, yang dapat jauh dari optima global. Model rata-rata juga mungkin jauh dari optima global, terutama ketika pembaruan lokal besar (misalnya, jumlah *epoch* lokal yang besar) [6]. Gambar 2.2 menunjukkan masalah pada *federated learning* dengan kondisi data non-IID. Pada kondisi IID, optima global w^* dekat dengan optima lokal w_1^* dan w_2^* sehingga model rata-rata w^{t+1} juga dekat dengan optima global. Namun, dalam kondisi non-IID, w^{t+1} jauh dari w^* karena w^* jauh dari w_1^* [6].



Gambar 2.2. Contoh *drift* pada kondisi non-IID
Sumber: [6]

Penelitian menunjukkan bahwa kehadiran data non-IID menyebabkan penurunan pada akurasi model global dibandingkan dengan skenario IID [6, 7]. Selain itu, proses pembelajaran menjadi tidak stabil dengan fluktuasi konvergensi yang tajam di berbagai ronde komunikasi, terutama ketika tingkat heterogenitas data sangat tinggi [6]. Fenomena ini tercermin dalam perilaku *loss* yang tidak menurun secara konsisten melainkan mengalami osilasi atau stagnasi pada nilai yang lebih tinggi dibandingkan kondisi IID. Dalam kasus ekstrem di mana setiap klien hanya memiliki data dari satu kelas, model global bahkan dapat gagal untuk konvergen sama sekali [7]. Ketidakstabilan ini mengakibatkan diperlukan lebih banyak ronde komunikasi untuk mencapai akurasi target yang sama sehingga meningkatkan biaya komunikasi dan waktu pelatihan keseluruhan dalam sistem *federated learning* [6].

2.4 Proximal Term pada FedProx

Tantangan non-IID memotivasi pengembangan FedProx, yang menambahkan *proximal term* pada objektif pelatihan lokal klien [8]. Secara teoritis, FedProx menyediakan garansi konvergensi untuk pembelajaran pada data dengan distribusi tidak identik (heterogenitas statistik). Secara praktis, FedProx menunjukkan konvergensi yang lebih tahan di berbagai dataset *federated learning* yang realistis, terutama dalam pengaturan yang sangat heterogen di mana FedProx meningkatkan akurasi pengujian absolut rata-rata sebesar 22% [8].

2.4.1 Formulasi

Modifikasi utama dari FedProx adalah penambahan *proximal term* pada objektif pelatihan lokal klien. Dengan menambahkan *proximal term*, setiap klien k melakukan optimasi pada objektif h_k:

$$\min_w h_k(w; w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2 \tag{2.2}$$

di mana $F_k(w)$ adalah fungsi objektif lokal pada klien k, w^t adalah model global dari iterasi t, w adalah variabel yang dioptimasi, dan $\mu > 0$ adalah hiperparameter penalti yang mengontrol kekuatan *proximal term*. Penambahan $\frac{\mu}{2}\|w - w^t\|^2$ berfungsi untuk membatasi jarak antara pembaruan lokal dan model global awal sehingga menjaga pembaruan lokal tetap dekat dengan titik awal [8].

Proximal term mengatasi heterogenitas statistik dengan membatasi deviasi pembaruan lokal tanpa perlu menetapkan jumlah *epoch* lokal (E) secara manual untuk setiap klien. Dalam pengaturan heterogen dengan data non-IID, melakukan terlalu banyak iterasi lokal dapat meningkatkan *drift* model lokal. Dengan menambahkan *proximal term*, regularisasi implisit diciptakan yang mencegah *drift* berlebihan sekaligus memungkinkan perangkat melakukan jumlah iterasi yang berbeda tanpa perlu koordinasi ketat mengenai jumlah *epoch* [8].

Hiperparameter μ memainkan peran kritis dalam FedProx. Nilai μ yang terlalu besar dapat memperlambat konvergensi dengan memaksa pembaruan tetap sangat dekat dengan model global. Sementara itu, nilai μ yang terlalu kecil mungkin tidak memberikan regulasi yang cukup untuk mencegah *drift* dan mengatasi heterogenitas statistik. Pemilihan μ yang optimal bergantung pada tingkat heterogenitas data dan karakteristik sistem dalam jaringan *federated learning* yang spesifik [8].

2.4.2 Implementasi di Plato

Plato dirancang dengan arsitektur modular dan dan berbasis komposisi (*composable design*) yang memungkinkan integrasi berbagai algoritma pembelajaran terdistribusi secara fleksibel. *Framework* ini mengadopsi desain berbasis strategi, di mana komponen pelatihan seperti fungsi *loss* dapat dikustomisasi melalui objek "strategy" yang *plug-and-play*. Implementasi FedProx di Plato tidak dilakukan melalui modifikasi *optimizer* secara langsung, melainkan melalui pendekatan berbasis fungsi *loss* dengan menambahkan *proximal term* ke dalam fungsi *loss* lokal pada klien [26].

Komponen inti implementasi adalah kelas `FedProxLossStrategy`, yang mengelola dua metode utama, yaitu `setup()` dan `compute_loss()`. Pada awal pelatihan lokal, metode `setup()` terlebih dahulu menginisialisasi fungsi *loss* dasar (*base loss*) yang akan digunakan selama proses optimisasi model. Setelah itu, *snapshot* dari bobot model global ditangkap agar digunakan sebagai titik referensi untuk *proximal term*. Mekanisme ini menyimpan w^t (model global dari iterasi saat ini) yang akan digunakan sepanjang ronde pelatihan lokal [26]. Metode `setup()` ditunjukkan pada Algoritma 1.

Algorithm 1 Setup FedProx Loss Strategy

Input: Training context containing model w^t
Output: Global model weights w^t stored for proximal term

if `base_loss_fn` is not defined **then**
    loss function ← `CrossEntropyLoss`
else
    loss function ← `base_loss_fn`
end if
Initialize `global_weights` as an empty collection
model ← `training_context.model`
if model is None **then**
    Raise an error
end if
for each parameter of the model **do**
    Store a detached copy of the parameter in `global_weights`
end for

Metode `compute_loss()` mengintegrasikan *proximal term* ke dalam fungsi *loss* yang akan di-*backpropagate*. Rumus yang diimplementasikan adalah:

$$\min_{w} h_k(w; w^t) = F_k(w) + \frac{\mu}{2}\sqrt{\sum_i (w_i - w_i^t)^2} \tag{2.3}$$

di mana $F_k(w)$ adalah *loss* dasar pada data lokal klien k, w adalah bobot model saat ini, w^t adalah bobot global yang disimpan, dan μ adalah hiperparameter *proximal term* [26]. Implementasinya ditunjukkan pada Algoritma 2.

Algorithm 2 Compute FedProx Loss

Input: Model outputs, labels, training context
Output: Total loss

Compute base loss using the loss function
Initialize `squared_diff_sum` $\leftarrow 0$
model $\leftarrow$ `training_context.model`
global_weights $\leftarrow$ stored global model weights
for each parameter in model **do**
    if parameter requires gradient and exists in `global_weights` **then**
        Retrieve the corresponding global parameter
        if `norm_type` is `"l2"` **then**
            `squared_diff_sum` $\leftarrow \sum_i (w_i - w_i^t)^2$
        end if
    end if
end for
if `norm_type` is `"l2"` **then**
    proximal term $\leftarrow \frac{\mu}{2}\sqrt{\texttt{squared\_diff\_sum}}$
end if
total loss $\leftarrow$ base loss + proximal term
return total loss

Dalam *loop* pelatihan, *proximal term* secara otomatis diterapkan pada setiap *batch* melalui mekanisme komposisi berikut. Mekanisme ini memastikan bahwa selama pelatihan lokal pada klien, pembaruan bobot tetap *proximity-constrained* terhadap model global, mencegah *drift* lokal [26].

1. Inisialisasi: *Trainer* membuat strategi *loss* dengan bobot global dari mode global
2. *Forward pass*: Model memprediksi keluaran untuk *batch* data lokal
3. Komputasi *loss*: `compute_loss()` menjumlahkan *loss* dasar dengan *proximal term*

4. *Backward pass*: Gradien dihitung dengan mempertimbangkan kedua komponen *loss*

5. *Optimizer*: *Stochastic gradient descent* (SGD) melakukan pembaruan bobot dengan gradien yang telah dimodifikasi oleh *proximal term*

2.4.3 Perbedaan Proximal Term pada Penelitian FedProx dengan Implementasi di Plato

Implementasi FedProx di *framework* Plato mengadopsi ide utama dari penelitian Li et al. (2020) mengenai penambahan *proximal term*. Namun, terdapat beberapa perbedaan antara formulasi teoritis di penelitian tersebut dengan realisasi praktis di Plato. Pada penelitian FedProx, *proximal term* dirumuskan pada fungsi objektif lokal menggunakan *squared* L2 *norm* [8], seperti yang ditunjukkan pada Rumus 2.2. Sebaliknya, implementasi pada Plato menggunakan L2 *norm* tanpa kuadrat pada fungsi *loss*, seperti yang ditunjukkan pada Algoritma 2 dan Rumus 2.3. Komentar pada kode sumber menyebutkan bahwa pemilihan ini dilakukan untuk menjaga kompatibilitas *backward* [26].

Selain itu, terdapat perbedaan juga pada referensi bobot global. Dalam formulasi teoritis, server mengirimkan model global w^t pada setiap ronde komunikasi. Setiap klien k menggunakan w^t tersebut sebagai titik referensi untuk menghitung *proximal term* sepanjang proses pelatihan lokal. Pada ronde berikutnya, w^t diperbarui menjadi w^{t+1} berdasarkan agregasi dari semua klien. Hal ini mencerminkan dinamika pembaruan model dalam skenario *federated learning*. Namun, pada Plato, bobot global hanya disalin satu kali pada awal pelatihan lokal. Nilai tersebut kemudian digunakan secara tetap sepanjang proses pelatihan lokal pada ronde tersebut. Dengan demikian, referensi w^t dalam implementasi Plato bersifat statis selama satu siklus pelatihan lokal. Bobot yang ditangkap disimpan dalam memori sepanjang ronde pelatihan lokal dan tidak diperbarui. *Proximal term* selalu mengacu pada bobot inisial tersebut, bukan model global yang terus berkembang di server.

Perbedaan ini menghasilkan perbedaan matematis dalam kekuatan penalti. *Proximal term* Plato berkembang lebih lambat terhadap perubahan parameter, mengizinkan divergensi lokal yang lebih besar sebelum penalti. Oleh karena itu, implementasi ini lebih tepat disebut sebagai strategi *loss* yang terinspirasi oleh FedProx daripada implementasi FedProx sesuai formulasi asli. Meskipun demikian, implementasi Plato tetap mempertahankan manfaat utama dari *proximal*

regularization, yaitu membatasi *drift* lokal dan meningkatkan stabilitas pelatihan dalam kondisi heterogen.

2.5 Homomorphic Encryption

Selain heterogenitas data yang memengaruhi proses optimasi, *federated learning* juga menghadapi tantangan terkait perlindungan privasi selama pertukaran parameter model. Meskipun data pelatihan tetap berada pada perangkat klien, parameter atau gradien yang dikirimkan masih berpotensi mengungkap informasi sensitif [19, 20]. Salah satu pendekatan yang digunakan untuk mengatasi permasalahan tersebut adalah *homomorphic encryption*.

Homomorphic encryption adalah skema enkripsi khusus yang memungkinkan komputasi langsung atas data terenkripsi tanpa perlu melakukan dekripsi terlebih dahulu. Konsep ini pertama kali diperkenalkan oleh Rivest, Adleman, dan Dertouzos pada tahun 1978 sebagai *privacy homomorphism* dalam konteks sistem data bank yang tersentralisasi [27]. Motivasi dari *homomorphic encryption* berasal dari skenario di mana pengguna ingin memercayakan data sensitif kepada server yang tidak sepenuhnya dapat dipercaya. Data sensitif pengguna dapat disimpan dalam bentuk terenkripsi di *cloud* dan server dapat melakukan komputasi tanpa pernah mengakses data *plaintext* [27]. Ini mencegah kebocoran data bahkan jika server dikompromikan atau memiliki administrator yang tidak terpercaya.

Secara formal, skema enkripsi *homomorphic encryption* didefinisikan oleh empat algoritma utama, yaitu *KeyGen*, *Enc*, *Dec*, dan *Eval*. Algoritma *KeyGen* menghasilkan pasangan kunci publik-privat (pk, sk). *Enc* mengenkripsi *plaintext* atau data biasa (m_1, m_2) menjadi *ciphertext* atau data terenkripsi (c_1, c_2) menggunakan kunci publik. *Dec* mendekripsi *ciphertext* kembali menjadi *plaintext* menggunakan kunci privat. Algoritma yang membedakan *homomorphic encryption* dari enkripsi konvensional adalah *Eval*, yang mengevaluasi fungsi $f()$ pada *ciphertext* terenkripsi tanpa melihat *plaintext* [28].

2.5.1 Operasi Aritmatika pada Data Terenkripsi

Suatu skema enkripsi disebut *homomorphic* terhadap operasi $\star$ jika memenuhi persamaan berikut.

$$E(m_1) \star E(m_2) = E(m_1 \star m_2), \quad \forall m_1, m_2 \in M \tag{2.4}$$

di mana E adalah fungsi enkripsi, M adalah himpunan semua pesan yang mungkin, dan $\star$ adalah operasi matematika arbitrer. Ini berarti mengenkripsi m_1 dan m_2 lalu melakukan operasi $\star$ pada hasilnya, sama dengan melakukan operasi $\star$ pada m_1 dan m_2 terlebih dahulu lalu mengenkripsi hasilnya [28, 29, 30].

Operasi penjumlahan pada *ciphertext* dapat didefinisikan sebagai berikut.

$$E(m_1) + E(m_2) = E(m_1 + m_2) \tag{2.5}$$

Hasil operasi penjumlahan dua *ciphertext* menghasilkan *ciphertext* baru yang, ketika didekripsi, memberikan hasil penjumlahan dari kedua *plaintext* asli [31]. Sementara itu, operasi perkalian pada *ciphertext* dapat ditulis sebagai:

$$E(m_1) * E(m_2) = E(m_1 * m_2) \tag{2.6}$$

Perkalian dua *ciphertext* menghasilkan *ciphertext* yang merepresentasikan perkalian dari *plaintext* aslinya [31].

Berdasarkan jenis operasi yang dapat dijalankan pada data terenkripsi, *homomorphic encryption* dibagi menjadi tiga kategori utama, yaitu *partially homomorphic encryption* (PHE), *somewhat homomorphic encryption* (SWHE), dan *fully homomorphic encryption* (FHE). PHE hanya mendukung satu operasi (penjumlahan atau perkalian) dengan efisiensi tinggi, SWHE mendukung keduanya namun terbatas oleh kedalaman komputasi akibat *noise*, sedangkan FHE memungkinkan komputasi arbitrer tanpa batas tetapi dengan *overhead* yang sangat tinggi.

2.5.2 Penerapan pada Federated Learning

Homomorphic encryption melindungi model pada *federated learning* dengan memungkinkan server agregasi melakukan komputasi tanpa pernah mengakses data *plaintext*. Keunggulan skema ini adalah seluruh proses agregasi terjadi di domain terenkripsi sehingga server agregasi tidak pernah melihat gradien *plaintext* individual atau agregat. Ini mencegah serangan inferensi keanggotaan dan inversi gradien yang mengkompromikan privasi data pelatihan. Mekanisme perlindungan ini bekerja dalam tiga tahap utama berikut [32].

1. Enkripsi gradien pada klien

   Setiap klien melakukan komputasi gradien secara lokal menggunakan dataset

pribadinya. Klien menghitung perubahan model berdasarkan fungsi *loss* yang dievaluasi pada data setempat. Hasil komputasi gradien tersebut langsung dienkripsi menggunakan kunci publik bersama sebelum ditransmisikan ke server pusat. Enkripsi dilakukan pada klien untuk memastikan bahwa gradien individual tidak pernah terekspos dalam bentuk *plaintext* selama proses transmisi. Mekanisme ini mencegah penyadap atau pihak ketiga lainnya untuk mengamati nilai gradien asli. Klien kemudian mengirimkan *ciphertext* (gradien terenkripsi) ke server agregasi, bukan nilai gradien mentah.

2. Agregasi *homomorphic* di server
   Server pusat menerima *ciphertext* dari berbagai klien dan melakukan penjumlahan langsung pada *ciphertext* yang diterima. Penjumlahan *ciphertext* menghasilkan *ciphertext* baru yang merepresentasikan enkripsi dari jumlah gradien asli. Server pusat tidak pernah mendapatkan akses ke nilai gradien individual maupun agregat dalam bentuk *plaintext* selama tahap agregasi ini. Proses agregasi ini terjadi sepenuhnya dalam domain terenkripsi.

3. Dekripsi dan pembaruan model
   Dekripsi dilakukan secara kolaboratif oleh *subset* klien yang tersedia. Skema menggunakan *secret sharing* sehingga tidak ada klien yang memegang kunci dekripsi lengkap. Sebaliknya, setiap klien memiliki *share* kunci privat, dan dekripsi hanya dapat dilakukan dengan menggabungkan *share* dari beberapa klien. Klien terpilih menggunakan *share* lokal untuk menghasilkan *shares* dekripsi dari *ciphertext* agregat. Server mengumpulkan *shares* dekripsi tersebut dan merekonstruksi gradien agregat dalam bentuk *plaintext*. Selanjutnya, server memperbarui model global dan menyiarkannya kembali ke semua klien.

2.5.3 Keterbatasan

Implementasi dari *homomorphic encryption* menghadapi tantangan berikut terkait *overhead* [3].

1. *Overhead* komputasi
   Operasi *homomorphic encryption* memerlukan biaya komputasi yang signifikan dibandingkan dengan komputasi pada *plaintext*. Sebagai contoh, operasi PHE didasarkan pada *modular exponentiation* dan *modular*

multiplication yang memiliki kompleksitas tinggi. Sementara itu, operasi FHE berbasis polinomial menjadi lebih kompleks daripada komputasi *plaintext*. Penelitian menunjukkan bahwa penggunaan Paillier (skema PHE) dapat meningkatkan waktu pelatihan hingga 96-135× dibandingkan dengan pembaruan *plaintext* biasa. *Overhead* komputasi ini menjadi hambatan dalam implementasi *privacy-preserving federated learning*.

2. *Overhead* komunikasi
   Ukuran *ciphertext* dalam *homomorphic encryption* jauh lebih besar dibandingkan *plaintext* sehingga transmisi gradien terenkripsi memerlukan *bandwidth* komunikasi yang lebih besar. Ketika beberapa klien mengirimkan model lokal terenkripsi ke server agregasi, volume data transfer meningkat drastis. Pertumbuhan ukuran pesan terenkripsi ini menjadi *bottleneck* dalam sistem *federated learning*, terutama pada lingkungan *cross-device* dengan konektivitas terbatas atau pada *edge devices* dengan kapasitas komunikasi yang rendah. Beban komunikasi yang tinggi secara langsung memengaruhi total waktu pelatihan dan konsumsi energi pada perangkat klien.

Homomorphic encryption yang diperlukan untuk perlindungan privasi penuh menyebabkan degradasi dalam efisiensi sistem. Semakin kuat tingkat privasi yang diinginkan (misalnya melalui parameter keamanan yang lebih besar), semakin tinggi *overhead* komputasi dan komunikasi yang dihasilkan [3]. Ini menciptakan dilema di mana organisasi harus memilih antara perlindungan privasi yang komprehensif dengan biaya komputasi dan *bandwidth* yang substansial, atau mengorbankan privasi demi efisiensi [28].

2.6 MaskCrypt

Overhead komputasi dan komunikasi tinggi yang disebabkan oleh *homomorphic encryption* memunculkan pertanyaan apakah seluruh bobot model perlu dienkripsi, atau cukup sebagian kecil yang dipilih secara strategis. Gagasan ini mendorong munculnya pendekatan *selective homomorphic encryption* (SHE) untuk menyeimbangkan keamanan dan efisiensi [4]. SHE mengadopsi konsep *encryption sparsity*, yaitu hanya mengenkripsi sebagian bobot model. Melalui mekanisme *mask*, klien memilih *subset* bobot untuk dienkripsi, sementara sisanya tetap dalam bentuk *plaintext*. Pendekatan ini memungkinkan pengaturan *trade-off* yang fleksibel antara tingkat privasi dan efisiensi komunikasi [4].

MaskCrypt merupakan implementasi dari pendekatan ini dalam konteks *federated learning*. Mekanisme ini menggunakan pemilihan *mask* yang dipandu gradien (*gradient-guided*) untuk menentukan bobot yang paling sensitif terhadap *loss*. Dengan *encrypt ratio* yang kecil (misalnya 20%), MaskCrypt mampu mengurangi *overhead* komunikasi, namun tetap mempertahankan tingkat perlindungan yang kompetitif terhadap berbagai serangan privasi [4]. Implementasi MaskCrypt tersedia dalam *framework* Plato [21]. Integrasi ini memungkinkan eksperimen terhadap berbagai strategi enkripsi selektif dalam lingkungan yang modular dan mudah dikembangkan.

2.6.1 Selective Homomorphic Encryption

Selective homomorphic encryption (SHE) adalah pendekatan optimasi terhadap *homomorphic encryption* tradisional yang mengatasi masalah *overhead* komunikasi dengan cara mengenkripsi hanya sebagian kecil dari bobot model, bukan keseluruhan. Pendekatan ini didasarkan pada intuisi bahwa tidak perlu mengenkripsi semua bobot model untuk mencapai tingkat keamanan privasi yang memadai. Dalam implementasinya, MaskCrypt menggunakan skema enkripsi Cheon-Kim-Kim-Song (CKKS) yang memungkinkan operasi aritmatika pada bilangan *floating-point* terenkripsi [4].

Pada *homomorphic encryption* tradisional, setiap bobot model $w_t^k[i]$ untuk $i = 1,\ldots,N$ dienkripsi secara keseluruhan sebelum dikirim ke server. Hal ini mengakibatkan inflasi ukuran *ciphertext* hingga puluhan atau ratusan kali lipat. SHE mengubah paradigma ini dengan menggunakan mekanisme *mask* enkripsi m^k yang didefinisikan sebagai *subset* dari indeks bobot model:

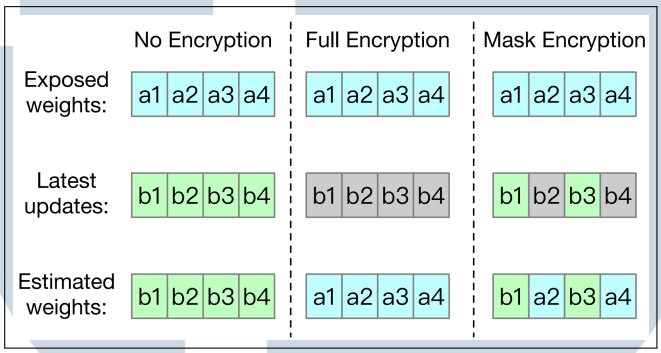
$$m^k \subset \{0,1,2,\ldots,N-1\}, \quad |m^k| = \rho N \tag{2.7}$$

di mana ρ adalah *encrypt ratio* yang mengontrol proporsi bobot yang dienkripsi, dengan $0 \leq \rho \leq 1$ [4]. Bobot dengan indeks dalam m^k dienkripsi, sementara bobot lainnya dikirim sebagai *plaintext* [4]. Secara formal, transformasi bobot dapat dituliskan sebagai:

$$w_t^k[i] = \begin{cases} w_t^k[i], & i \notin m^k \\ \mathrm{ENC}(w_t^k[i]), & i \in m^k \end{cases} \tag{2.8}$$

di mana $\mathrm{ENC}(\cdot)$ adalah fungsi *homomorphic encryption* [4].

Gambar 2.3 mengilustrasikan kemampuan pihak lawan dalam memperkirakan bobot model pada beberapa skema enkripsi yang berbeda. Pada skema tanpa enkripsi, seluruh bobot dapat diamati langsung. Pada *homomorphic encryption* penuh, seluruh bobot terlindungi namun dengan *overhead* komunikasi tinggi. Sementara itu, SHE melindungi sebagian bobot yang dianggap sensitif melalui mekanisme *mask* sehingga diperoleh keseimbangan antara perlindungan privasi dan efisiensi komunikasi.



Gambar 2.3. Bobot model yang dapat diperkirakan oleh pihak lawan di bawah skema enkripsi yang berbeda

Sumber: [4]

A Pemilihan Mask yang Dipandu Gradien

Keberhasilan SHE sangat bergantung pada pemilihan bobot mana yang harus dienkripsi. MaskCrypt menggunakan mekanisme pemilihan *mask* yang dipandu gradien dengan memilih bobot berdasarkan kontribusi mereka terhadap peningkatan nilai *loss* dari bobot yang terekspos. Intuisi di balik pendekatan ini adalah untuk memaksimalkan *loss* pada model yang terekspos sehingga model tersebut terlihat tidak terlatih pada data pribadi klien [4]. Proses pemilihan tersebut dirangkum dalam Algoritma 3.

Algorithm 3 Pemilihan *Mask* yang Dipandu Gradien

Input: $w^k_{\text{exp},t}$ (exposed model weights), w^k_t (local model weights), D^k (local dataset)
Output: m (encryption mask)

1: Compute gradients $g = \nabla L(w^k_t, D_k)$
2: Compute model delta $\delta(1) = w^k_{\text{exp},t} - w^k_t$
3: Compute element-wise multiplication $v = g \odot \delta(1)$
4: Sort v in descending order and obtain indices v_{ind}
5: Select top ρN elements of v_{ind} as the mask proposal m^k
6: **return** m^k

Mekanisme dimulai dengan menghitung gradien $g = \nabla L(w_t^k, D_k)$, yaitu turunan dari fungsi *loss* L terhadap bobot lokal w_t^k pada dataset lokal D^k. Gradien ini merepresentasikan arah dan besarnya perubahan bobot yang paling berpengaruh dalam menurunkan nilai *loss*. Selanjutnya, dihitung delta model $\delta(1) = w_{\text{exp},t}^k - w_t^k$, di mana $w_{\text{exp},t}^k$ adalah bobot yang terekspos (tidak dienkripsi) dan w_t^k adalah bobot lokal hasil pelatihan pada klien ke-k di iterasi ke-t. Nilai $\delta(1)$ ini menunjukkan perbedaan kontribusi antara bobot terekspos dan bobot lokal.

Kemudian dilakukan perkalian elemen demi elemen antara gradien dan selisih bobot, yaitu $v = g \odot \delta(1)$, dengan $\odot$ menyatakan operasi *Hadamard product*. Vektor v ini mengukur seberapa besar kontribusi masing-masing bobot terhadap peningkatan nilai *loss* jika bobot terekspos digunakan. Semakin besar nilai pada elemen v, semakin besar pula pengaruh bobot tersebut dalam meningkatkan *loss*.

Setelah itu, elemen-elemen dalam v diurutkan secara menurun untuk memperoleh indeks v_{ind} yang merepresentasikan urutan bobot berdasarkan tingkat kontribusinya. Dari hasil pengurutan tersebut, dipilih ρN elemen teratas sebagai *mask* m^k. *Mask* m^k ini kemudian digunakan untuk menentukan bobot mana yang akan dienkripsi, dengan tujuan memaksimalkan *loss* pada model terekspos sehingga informasi sensitif pada data klien tetap terlindungi. Pendekatan ini terbukti dapat memaksimalkan batas bawah dari fungsi *loss* pada bobot terekspos:

$$L(w_{\text{exp},t}^k; D_k) \geq L(w_t^k; D_k) + \delta(m^k) \cdot g \tag{2.9}$$

dengan memaksimalkan *dot product* $\delta(m^k) \cdot g$ [4].

B Konsensus Mask

Dalam implementasi sebelumnya yang disebut *masking* individual, setiap klien dapat memilih *mask* enkripsi sendiri sesuai dengan preferensi privasi mereka. Pendekatan ini menghadapi dua masalah. Pertama, ketika klien yang berbeda memilih indeks enkripsi yang berbeda, hasil agregasi pada server akan memiliki posisi yang terenkripsi berbeda, menyebabkan inflasi *ciphertext* meningkat setelah agregasi. Kedua, sebagian besar skema *homomorphic encryption* menggunakan teknik *batching* untuk mengurangi ukuran *ciphertext*, yang berarti enkripsi dilakukan pada level vektor bukan elemen individual sehingga tidak mungkin mengekstraksi elemen terenkripsi spesifik dari *ciphertext* selama agregasi. MaskCrypt mengusulkan mekanisme konsensus *mask* untuk mengatasi masalah

ini dengan memastikan semua klien menggunakan *mask* enkripsi yang sama [4]. Mekanisme tersebut dipaparkan pada Algoritma 4.

Algorithm 4 Konsensus *Mask*

Input: $m^1, ..., m^k$ (mask proposals from clients), ρ (encrypt ratio)
Output: $\bar{m}$ (final mask)
1: Initialize m' as a list of zeros of length $\rho N \times K$
2: **for** $k = 1, 2, \ldots, K$ **do**
3: $\quad m'[k :: M] = m^k$ ▷ Interleave the mask proposals
4: **end for**
5: Remove the duplicated elements in m'
6: Select top ρN elements of m' as final mask $\bar{m}$
7: **return** $\bar{m}$

Proses konsensus *mask* dimulai dengan menginisialisasi m' sebagai sebuah vektor nol dengan panjang $\rho N \times K$, di mana K adalah jumlah klien yang berpartisipasi dalam *federated learning*. Vektor m' ini digunakan sebagai wadah untuk menggabungkan seluruh proposal *mask* dari setiap klien. Setiap klien ke-k menghasilkan proposal *mask* m^k, yang berisi indeks bobot yang dipilih untuk dienkripsi berdasarkan mekanisme sebelumnya.

Pada langkah iterasi, dilakukan operasi *interleaving* dengan menempatkan elemen m^k ke dalam m' menggunakan pola indeks $m'[k :: M] = m^k$, yang berarti elemen dari setiap klien disisipkan secara bergantian ke dalam posisi tertentu pada m'. Dalam hal ini, M menyatakan panjang dari setiap proposal *mask* m^k, yang bernilai ρN (jumlah elemen yang dipilih oleh masing-masing klien). Operasi ini memastikan bahwa elemen pertama dari semua klien ditempatkan terlebih dahulu, diikuti oleh elemen kedua dari semua klien, dan seterusnya. Proses *interleaving* tersebut dapat dinyatakan secara matematis sebagai berikut.

$$m' = \left[m^1[0], m^2[0], \ldots, m^K[0], m^1[1], m^2[1], \ldots\right] \tag{2.10}$$

Dengan demikian, setiap klien memiliki kontribusi yang tersebar secara merata dalam vektor gabungan m' sehingga tidak ada bias urutan terhadap klien tertentu dalam proses pembentukan *mask* akhir.

Setelah itu, dilakukan penghapusan elemen duplikat untuk memastikan bahwa setiap indeks bobot hanya muncul satu kali. Hal ini penting agar tidak terjadi redundansi dalam pemilihan bobot yang akan dienkripsi. Selanjutnya, dipilih ρN elemen teratas dari m' sebagai *mask* akhir $\bar{m}$. Pemilihan ini bertujuan untuk mempertahankan jumlah bobot yang dienkripsi sesuai dengan rasio yang telah

ditentukan, sekaligus memastikan bahwa *mask* akhir merepresentasikan konsensus dari seluruh klien. *Mask* akhir $\bar{m}$ inilah yang kemudian digunakan oleh semua klien dalam proses enkripsi sehingga posisi bobot yang dienkripsi menjadi konsisten dan kompatibel selama proses agregasi di server. Dengan *mask* yang sama, agregasi dapat dilakukan secara terpisah untuk bobot *plaintext* dan *ciphertext*:

$$w_{\text{plain},t} = \sum_{k=1,\ldots,K} p_k w^k_{\text{plain},t} \tag{2.11}$$

$$w_{\text{enc},t} = \sum_{k=1,\ldots,K} p_k \text{ENC}(w^k_{\text{enc},t}) \tag{2.12}$$

C Keamanan dan Efisiensi

Dari perspektif keamanan, SHE dalam MaskCrypt mempertahankan properti keamanan yang dikombinasikan dari *homomorphic encryption* dan *masking* selektif. MaskCrypt dikembangkan dengan asumsi bahwa server bersikap *honest-but-curious* dan skema enkripsi yang digunakan memenuhi properti ketidakmampuan untuk dibedakan di bawah serangan *plaintext* yang dipilih (IND-CPA), MaskCrypt mencapai kerahasiaan model terenkripsi selama server berkolusi dengan paling banyak $K-2$ klien dari total K klien. Informasi tentang bobot mana yang dienkripsi tidak memberikan keuntungan bagi musuh karena hanya mengungkapkan bahwa nilai *dot product* bobot terenkripsi dengan gradien lebih tinggi dari bobot *plaintext*, sesuatu yang tidak dapat diverifikasi tanpa akses ke gradien lokal klien [4].

Dari sisi efisiensi, eksperimen menunjukkan bahwa dengan *encrypt ratio* $\rho = 0,2$ (20%), MaskCrypt dapat mengurangi *overhead* komunikasi hingga 4,15 kali dibandingkan dengan enkripsi penuh pada model ResNet-18 untuk dataset CIFAR-10, sementara tetap memberikan perlindungan serupa terhadap serangan inferensi keanggotaan dan rekonstruksi data. Pada model yang lebih besar seperti DistilGPT2, pengurangan komunikasi mencapai 4,3 kali dengan *encrypt ratio* 0,2, menjadikan SHE pendekatan untuk skenario *federated learning* dunia nyata dengan model skala besar [4].

2.6.2 Alur Kerja

Alur kerja MaskCrypt terdiri dari serangkaian tahap yang dirancang untuk mengintegrasikan enkripsi selektif ke dalam proses pelatihan *federated learning*. Alur kerja diawali dengan tahap *setup* yang dilaksanakan hanya sekali sebelum pelatihan dimulai. Setelah *setup* selesai, proses pelatihan berjalan secara siklis dengan pola yang berbeda untuk ronde ganjil dan genap. Pada ronde ganjil, klien melaksanakan pelatihan lokal, kemudian menghitung konsensus mask. Pada ronde genap, klien mengenkripsi model lalu server mengagregasinya. Terakhir pada ronde ganjil berikutnya, klien melakukan dekripsi model sebelum digunakan pada pelatihan berikutnya [4, 26].

A Setup

Tahap *setup* merupakan tahap inisialisasi awal dalam alur kerja MaskCrypt sebelum pelatihan dimulai. Pada tahap ini, setiap klien k membangkitkan pasangan kunci publik-privat (pk_k, sk_k) menggunakan skema *homomorphic encryption*. Kunci publik pk_k kemudian disinkronisasi di antara semua klien, sementara kunci privat sk_k disimpan secara lokal. Mekanisme ini memungkinkan setiap klien mengenkripsi *subset* bobot model yang berbeda dengan kunci klien yang berbeda-beda, memastikan bahwa kompromi pada satu klien tidak membahayakan keamanan klien lainnya [4].

Dalam implementasi praktis pada *framework* Plato, manajemen kunci publik-privat (pk_k, sk_k) *homomorphic encryption* diabstraksi dalam lapisan eksternal melalui pustaka TenSEAL, yaitu sebuah pustaka kelas industri yang mengimplementasikan operasi *homomorphic* pada tensor berdasarkan Microsoft SEAL [33]. Plato menyediakan konteks TenSEAL yang diperlukan untuk operasi enkripsi dan dekripsi. Kunci diinisialisasi secara otomatis pada saat *framework* Plato *startup* dan ditangani secara transparan dalam lapisan prosesor, memastikan operasi enkripsi dan dekripsi dilakukan tanpa intervensi eksplisit di level kode *setup* klien MaskCrypt [26].

Selain manajemen kunci, tahap *setup* juga menginisialisasi *persistent state* dan mengonfigurasi parameter enkripsi. Inisialisasi *persistent state* dilakukan dengan mengisi *state dictionary* dengan nilai bawaan yang diperlukan selama alur kerja MaskCrypt. Sementara itu, konfigurasi parameter klien MaskCrypt memastikan setiap klien memiliki pengaturan yang konsisten dan siap melakukan

enkripsi selektif terhadap bobot modelnya sesuai dengan rasio dan mode yang telah ditentukan [26].

B Pelatihan Lokal

Tahap pelatihan lokal merupakan tahap di mana setiap klien yang terpilih menerima bobot model global dari server dan melakukan pelatihan lokal pada dataset pribadinya. Mekanisme ini terjadi pada setiap ronde t yang ganjil, dimulai dengan server mengirimkan bobot model global w_{t-1} kepada setiap klien k. Klien menerima muatan tersebut dan memulai proses pelatihan lokal. Klien k kemudian mendefinisikan fungsi *loss* $L(w, D^k)$ pada model w dan dataset lokal D^k, lalu melatih model dengan algoritma optimasi SGD pada dataset lokal D^k selama beberapa *epoch*, menghasilkan bobot model yang telah diperbarui secara lokal w_t^k [4].

Pada MaskCrypt, fungsi *loss* yang digunakan dalam proses pelatihan lokal adalah *cross-entropy loss* [26], yaitu fungsi yang mengukur seberapa jauh perbedaan antara prediksi model dan label sebenarnya. Secara matematis, *cross-entropy loss* untuk klasifikasi dengan K kelas dapat dinyatakan sebagai berikut:

$$l(\pi; y) = -\sum_{k=1}^{K} y_k \log \pi_k \tag{2.13}$$

di mana y_k adalah label sebenarnya dalam bentuk *one-hot* (bernilai 1 untuk kelas yang benar dan 0 untuk lainnya), sedangkan π_k adalah probabilitas prediksi model terhadap kelas ke-k. Fungsi ini akan memberikan nilai *loss* yang kecil jika model memberikan probabilitas tinggi pada kelas yang benar, dan sebaliknya akan menghasilkan *loss* yang besar jika prediksi model jauh dari label sebenarnya. Dengan demikian, selama proses pelatihan lokal, model akan terus diperbarui untuk meminimalkan nilai *loss* ini sehingga prediksi yang dihasilkan semakin mendekati distribusi label yang sebenarnya [34].

C Konsensus Mask

Tahap konsensus *mask* merupakan tahap yang dilaksanakan pada setiap ronde t yang ganjil, setelah klien menyelesaikan pelatihan lokal. Pada tahap ini, setiap klien yang terpilih menghitung proposal *mask* berdasarkan bobot yang telah diperbarui dan gradien yang diperoleh dari pelatihan lokal. Perhitungan proposal *mask* ini bertujuan menentukan bobot mana saja yang akan dienkripsi pada tahap

enkripsi model berikutnya. Setelah proposal *mask* dihitung, klien mengirimkannya ke server yang kemudian mengumpulkan proposal *mask* dari semua klien yang terpilih dan melakukan proses konsensus untuk menghasilkan *mask* akhir. *Mask* akhir ini akan digunakan oleh seluruh klien pada ronde pelatihan berikutnya [4].

Sebelum perhitungan proposal mask m^k dimulai, klien terlebih dahulu mengekstrak gradien lokal yang merupakan hasil perhitungan turunan fungsi *loss* terhadap bobot model pada dataset lokal klien. Kemudian, klien melakukan pra-pemrosesan terhadap bobot dan gradien dengan meratakan dan mengombinasikannya dengan estimasi bobot terekspos. Pemilihan *mask* berbasis gradien dilakukan dengan menghitung delta antara bobot yang terekspos dan bobot terbaru, kemudian melakukan perkalian elemen-demi-elemen antara delta dan gradien untuk mengukur kontribusi setiap bobot terhadap peningkatan nilai *loss*. Indeks-indeks dengan nilai kontribusi tertinggi dipilih hingga mencapai batas enkripsi yang ditentukan oleh *encrypt ratio*, menghasilkan proposal *mask* m^k yang merepresentasikan daftar bobot paling penting untuk dilindungi menurut preferensi klien. Proposal *mask* ini kemudian dikirimkan ke server untuk konsensus mask [26].

Server mengumpulkan semua proposal *mask* dari klien-klien yang terpilih dan menerapkan prosedur *interleaving* dan deduplikasi. Dengan memastikan bahwa preferensi setiap klien dihargai secara seimbang melalui proses *interleaving* bergilir, algoritma ini menghasilkan konsensus *mask* yang mencerminkan sensitivitas bobot yang disepakati oleh mayoritas klien. Indeks bobot yang muncul lebih awal dalam urutan *interleaving* dianggap lebih sensitif dan diprioritaskan dalam konsensus *mask* akhir sehingga memastikan semua klien menggunakan enkripsi *mask* yang sama pada tahap selanjutnya [26].

D Enkripsi Model

Tahap enkripsi model dilaksanakan pada setiap ronde t yang genap, setelah konsensus *mask* selesai pada ronde sebelumnya. Setelah server dan klien mencapai konsensus pada *mask* enkripsi akhir, klien-klien yang terpilih mengenkripsi sebagian bobot model yang telah ditentukan dalam $\overline{m}$ menggunakan kunci publik semua klien yang terlibat. Enkripsi dilakukan melalui skema *homomorphic encryption* dengan mekanisme pembagian *subset* untuk memastikan keamanan multi-pihak, di mana setiap klien memiliki tanggung jawab dalam proses enkripsi dan dekripsi. Mekanisme ini menjamin bahwa bobot-bobot yang dienkripsi dapat

didekripsi hanya oleh klien yang memiliki kunci privat yang sesuai, melindungi bobot model yang sensitif sebelum dikirimkan ke server [4].

Pada tahap enkripsi model, setelah klien menerima konsensus *mask* akhir $\overline{m}$ dari server, klien k memisahkan $\overline{m}$ ke dalam K subset yang sama besar $I_1, I_2, \ldots, I_K$ sehingga memenuhi kondisi $\bigcup_{j=1}^{K} I_j = \overline{m}$ dan $|I_1| = |I_2| = \cdots = |I_K|$. Setiap subset I_j mewakili indeks-indeks bobot yang akan dienkripsi dengan kunci publik pk_j milik klien j, memastikan bahwa hanya klien j yang memiliki kunci privat sk_j dapat mendekripsi bobot yang dienkripsi dengan kunci tersebut. Setelah pembagian *subset* selesai, klien k mengenkripsi elemen-elemen bobot yang sesuai menggunakan public key pk_j dari klien j melalui skema *homomorphic encryption* berikut [4].

$$w^k_{t,I_j} = \mathrm{ENC}(w^k_t[i], pk_j) \quad \forall i \in I_j \tag{2.14}$$

Proses ini menghasilkan K subset bobot terenkripsi $w^k_{t,I_1}, w^k_{t,I_2}, \ldots, w^k_{t,I_K}$ untuk setiap klien k, masing-masing dienkripsi dengan kunci yang berbeda. Strategi pembagian kunci ini memastikan keamanan multi-pihak karena dekripsi bobot memerlukan kolaborasi dari semua klien yang memiliki kunci dekripsi. Tidak ada satu klien pun yang dapat mendekripsi seluruh bobot terenkripsi tanpa kunci privat klien lainnya sehingga keamanan terhadap kolusi dipertingkat [4]. Pembaruan model lengkap yang dikirim klien k ke server terdiri dari bobot terenkripsi dan bobot yang tidak dienkripsi dengan bentuk berikut [4].

$$\{w^k_{t,I_j} | j = 1, \ldots, K\} \cup \{w^k_t[i] | \forall i \notin \overline{m}\} \tag{2.15}$$

Pada implementasinya, sebelum melakukan enkripsi, klien menerima *mask* akhir $\overline{m}$ dari server. Prosesor enkripsi mempersiapkan konteks CKKS pada konstruktor lalu mengekstrak informasi tentang jumlah parameter di setiap layer model. Bobot model diterima dalam bentuk *plaintext* dan bobot yang telah dienkripsi dikembalikan melalui operasi *homomorphic encryption*, dengan enkripsi dilakukan hanya pada elemen-elemen bobot yang teridentifikasi dalam *mask*.

Enkripsi dilakukan dengan melakukan beberapa langkah penting untuk mengenkripsi bobot model secara selektif. Pertama, fungsi meratakan semua tensor bobot dari setiap lapisan model menjadi satu vektor kontinu, yang diperlukan karena *mask* yang diterima berisi indeks-indeks pada bobot yang telah diratakan.

Kedua, fungsi memisahkan bobot menjadi dua kategori berdasarkan *mask*, yaitu bobot yang harus dienkripsi dan bobot yang tetap *plaintext*. Pemisahan dilakukan dengan menghapus indeks yang ada dalam *mask* dari vektor bobot lengkap untuk mendapatkan bobot yang tidak dienkripsi. Ketiga, bobot yang dipilih untuk dienkripsi melalui proses *homomorphic encryption* menggunakan algoritma CKKS, menghasilkan *ciphertext* yang telah diserialisasi. Terakhir, hasil enkripsi dibungkus bersama dengan bobot *plaintext* dan informasi indeks untuk menghasilkan keluaran yang berisi struktur lengkap dari pembaruan model, mempertahankan informasi tentang bobot mana yang dienkripsi sehingga server dapat melakukan agregasi dengan benar [26].

E Agregasi Model

Tahap agregasi model merupakan tahap yang dilaksanakan pada setiap ronde t yang genap, setelah klien-klien mengirimkan pembaruan model yang telah dienkripsi secara selektif. Pada tahap ini, server menerima pembaruan model dari semua klien yang terpilih. Karena bobot *ciphertext* dan bobot *plaintext* telah selaras pada posisi yang sama melalui penggunaan *mask* konsensus, server dapat mengagregasi keduanya secara terpisah [4].

Server melakukan operasi *homomorphic* pada bobot *ciphertext* w_{t,I_j} dan operasi aritmatika langsung pada bobot *plaintext* $w_t[i]$. Agregasi bobot plaintext dilakukan melalui penjumlahan aritmatika biasa dengan pembobotan kontribusi klien, menghasilkan bobot agregat dalam bentuk *plaintext*, seperti pada Rumus 2.11 [4]. Sementara itu, agregasi bobot *ciphertext* dilakukan melalui evaluasi operasi *homomorphic encryption* khusus yang sesuai dengan algoritma enkripsi yang digunakan, menghasilkan bobot agregat yang masih dalam bentuk *ciphertext*, seperti pada Rumus 2.12 [4].

Hasil agregasi lengkap pada tahap ini terdiri dari dua komponen, yaitu bobot *plaintext* yang telah diagregasi $w_{\text{plain},t}$ dan bobot *ciphertext* yang telah diagregasi $w_{\text{enc},t}$ yang masing-masing dienkripsi dengan kunci publik klien-klien yang berbeda [4]:

$$\{w_{t,I_j} | j = 1, \ldots, K\} \cup \{w_t[i] | \forall i \notin \overline{m}\} \tag{2.16}$$

Bobot *plaintext* yang telah diagregasi dapat langsung digunakan untuk ronde pelatihan berikutnya. Namun, setiap w_{t,I_j} masih tetap terenkripsi di bawah kunci publik klien j sehingga memerlukan satu langkah dekripsi tambahan sebelum dapat digunakan dalam pelatihan berikutnya. Mekanisme ini memastikan bahwa bobot

sensitif tetap terlindungi selama proses agregasi dan tidak ada satu entitas pun, termasuk server, yang dapat mengakses bobot *ciphertext* tanpa kolaborasi dari klien yang memiliki kunci dekripsi [4].

Tahap agregasi model diimplementasikan dengan terlebih dahulu memfilter pembaruan dari klien yang dianggap layak, yaitu klien yang mengirimkan bobot (bukan fitur), serta menghitung total jumlah sampel dan faktor pembobot p_k berdasarkan rasio sampel pelatihan lokal terhadap total sampel semua klien. Setelah penghitungan bobot agregasi, bobot terenkripsi dan tidak terenkripsi diakumulasi secara terpisah dengan algoritma *weighted averaging* standar FedAvg. Setelah semua klien diproses, hasil agregasi dipastikan memiliki tipe data yang sesuai dengan bobot *baseline* [26].

F Dekripsi Model

Tahap dekripsi model merupakan langkah akhir dalam pelatihan MaskCrypt, yang dilaksanakan pada ronde ganjil setelah server mengirimkan kembali hasil agregasi dari ronde sebelumnya. Karena *subset* bobot terenkripsi w_{t,I_j} masih berada dalam bentuk *ciphertext* dengan kunci publik klien j, bobot-bobot tersebut tidak dapat digunakan langsung untuk pelatihan berikutnya. Server mengirimkan setiap *subset* bobot terenkripsi kepada klien j yang memiliki kunci privat sk_j yang sesuai. Klien melakukan dekripsi menggunakan kunci privatnya dan server mengumpulkan semua *subset* bobot yang telah didekripsi lalu menggabungkannya dengan bobot *plaintext* dari tahap agregasi sebelumnya. Model agregat w_t yang telah lengkap dalam bentuk *plaintext* ini kemudian dapat digunakan sebagai inisialisasi ronde pelatihan berikutnya [4].