BAB 2
LANDASAN TEORI

2.1 Penelitian Terdahulu

Federated Learning berkembang dari kebutuhan untuk melatih model dari data terdistribusi tanpa memusatkan *raw data*. FedAvg menjadi salah satu fondasi utama karena menunjukkan bahwa model global dapat dibentuk melalui agregasi berbobot dari pembaruan model lokal [2]. Tantangan *data heterogeneity*, khususnya *label skew* dan *data scarcity*, menjadi perhatian karena dua kondisi tersebut dapat menyebabkan *local model drift* dan menurunkan kemampuan generalisasi model global [3, 4, 5].

Beberapa pendekatan berupaya menangani heterogenitas data dengan memperbaiki *objective* lokal, misalnya FedProx yang menambahkan regularisasi terhadap jarak antara model lokal dan model global [19], serta FedNTD yang mempertahankan *global knowledge* melalui *distillation* pada kelas *non-ground-truth* [20]. Pendekatan lain menggunakan informasi tambahan untuk membantu *local training*, seperti FedMix, FedGen, FedFed, dan FLea. Kelompok metode ini relevan karena menunjukkan bahwa informasi di luar data lokal dapat membantu *client* ketika distribusi kelas tidak seimbang atau jumlah sampel lokal terbatas [1, 21, 22, 23].

Di sisi *privacy*, tidak membagikan *raw data* belum selalu cukup untuk menghilangkan risiko kebocoran informasi. Gradien, parameter model, *output* model, maupun *intermediate activation* dapat membawa informasi dari *raw input* dalam kondisi tertentu [8, 9, 10, 11]. *Differential Privacy* digunakan untuk membatasi informasi yang dapat disimpulkan dari data atau representasi yang dibagikan [14, 15]. Pada *Gaussian Mechanism*, besar *Gaussian noise* dikalibrasi terhadap sensitivitas L2 dari nilai yang dilepas. Dalam *deep learning*, *clipping* per sampel dapat digunakan untuk memberikan batas sensitivitas sebelum *noise* ditambahkan [15, 18]. Tabel 2.1 merangkum penelitian terkait *Federated Learning*, *feature sharing*, dan mekanisme *privacy* yang menjadi dasar posisi penelitian ini.

Tabel 2.1. Ringkasan penelitian terdahulu terkait *Federated Learning*, *feature sharing*, dan *privacy*

Literatur	**Tahun**	**Fokus Penelitian**	**Metode/Model**	***Dataset***	**Keterbatasan**
FedAvg [2]	2017	*Training* model dari data terdistribusi	*Federated Averaging*	MNIST, CIFAR-10, dan lainnya	Rentan terhadap data non-IID dan *client drift*
FedProx [19]	2020	Optimisasi FL pada data heterogen	*Proximal regularization*	*Federated benchmark*	Fokus pada stabilisasi *local update*, bukan *data scarcity* atau *feature sharing*
FedNTD [20]	2022	Menjaga *global knowledge* pada non-IID FL	*Not-true distillation*	CIFAR-10, CIFAR-100, Tiny-ImageNet	Tidak dirancang untuk membagikan representasi *feature* antar-*client*
FedMix [22]	2021	*Data augmentation* pada FL	*Mean augmented mixup*	CIFAR-10 dan *benchmark* visual	Informasi global yang dibagikan masih bergantung pada kualitas augmentasi
FedFed [21]	2023	*Feature distillation* untuk *data heterogeneity*	*Feature distillation*	CIFAR-10, CIFAR-100, dan lainnya	Perlindungan *privacy feature* tidak diformulasikan sebagai DP dengan *privacy budget* yang jelas
FLea [1]	2024	*Data scarcity* dan *label skew* pada FL	*Feature buffer* dan *feature augmentation*	CIFAR-10, UrbanSound8K, UCI-HAR	*Shared feature* masih memerlukan analisis *privacy* lebih lanjut
Deep Learning with DP [18]	2016	*Differential Privacy* untuk *deep learning*	DP-SGD	MNIST, CIFAR-10	Melindungi proses *training* berbasis *gradient*, bukan secara langsung *shared activation*
FedLA [24]	2026	*Privacy* dan efisiensi pada *federated distillation*	Mekanisme *privacy* dan *active data sampling*	CIFAR-10, CIFAR-100	Mekanisme *privacy* diterapkan pada *logits/output*, bukan *intermediate activation*

Tabel 2.1 menunjukkan bahwa penelitian sebelumnya dapat dikelompokkan ke dalam tiga arah utama. FedAvg, FedProx, dan FedNTD berfokus pada mekanisme *training* atau stabilisasi pembaruan model. FedMix, FedFed, dan FLea menambahkan informasi di luar data lokal untuk mengurangi dampak heterogenitas data. Sementara itu, Deep Learning with DP dan FedLA menunjukkan bahwa mekanisme *privacy* berbasis *noise* dapat digunakan untuk membatasi informasi yang dilepas. Posisi penelitian ini berada pada irisan antara *feature sharing* dan mekanisme *privacy*, yaitu melindungi *activation* yang masuk ke *feature buffer*.

2.2 *Federated Learning*

Federated Learning merupakan paradigma pembelajaran mesin terdistribusi yang memungkinkan sejumlah *client* melatih model bersama tanpa memindahkan *raw data* ke *server* pusat. Dalam skema ini, data tetap berada pada *device* atau

institusi pemilik data, sedangkan proses kolaborasi dilakukan melalui pertukaran parameter, gradien, atau pembaruan model. Pendekatan ini dikembangkan untuk menjawab keterbatasan pembelajaran terpusat ketika data tersebar pada banyak sumber dan tidak selalu dapat dikumpulkan secara langsung ke satu *server* [2, 3, 25].

Secara umum, *Federated Learning* melibatkan dua komponen utama, yaitu *server* pusat dan sejumlah *client*. *Server* bertugas menginisialisasi model global, memilih *client* yang berpartisipasi pada suatu *communication round*, mengirimkan model global kepada *client* terpilih, lalu menerima pembaruan model dari *client*. Sementara itu, setiap *client* melakukan *training* secara lokal menggunakan *dataset* masing-masing. Setelah *local training* selesai, *client* mengirimkan hasil pembaruan model tanpa menyertakan *raw data*. Proses ini dilakukan secara berulang sampai model global mencapai performa yang diharapkan [2, 3, 26].

Metode dasar yang banyak digunakan dalam *Federated Learning* adalah *Federated Averaging* atau FedAvg. FedAvg menjalankan proses *training* dengan cara mengirimkan model global ke *client*, melakukan beberapa langkah optimisasi lokal pada masing-masing *client*, kemudian menggabungkan parameter model lokal di *server*. Dengan demikian, pembaruan model global tidak dihitung dari seluruh data secara terpusat, tetapi dari hasil *training* lokal yang dikirimkan oleh *client* terpilih [2].

Jika $\theta^{(t)}$ menyatakan parameter model global pada *communication round* ke-t, $K^{(t)}$ menyatakan himpunan *client* yang terpilih pada *communication round* tersebut, D_k menyatakan *dataset* lokal pada *client* ke-k, dan θ_k menyatakan parameter model lokal setelah *local training*, maka agregasi FedAvg mengikuti bentuk pada Persamaan 2.1.

$$\theta^{(t+1)} = \sum_{k \in K^{(t)}} \frac{|D_k|}{\sum_{k \in K^{(t)}} |D_k|} \theta_k \tag{2.1}$$

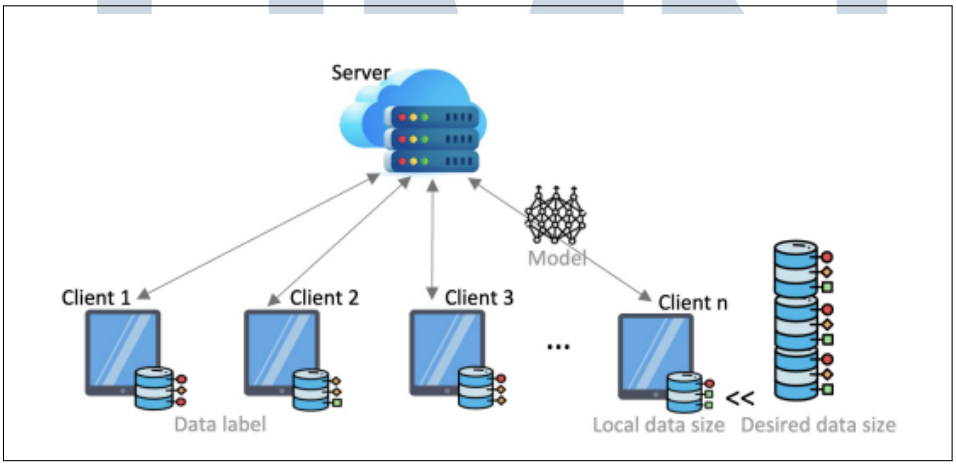
Pada Persamaan 2.1, $|D_k|$ adalah jumlah sampel pada *dataset* lokal *client* ke-k, sedangkan $\sum_{k \in K^{(t)}} |D_k|$ adalah total sampel dari seluruh *client* yang berpartisipasi pada *communication round* ke-t. Kontribusi setiap *client* terhadap model global ditentukan oleh proporsi jumlah data lokalnya terhadap total data *client* terpilih. *Client* dengan data lebih banyak memiliki bobot agregasi lebih besar, sedangkan *client* dengan data lebih sedikit memiliki bobot lebih kecil. Rumus ini menjadi dasar bagi banyak metode *Federated Learning* karena menjelaskan mekanisme inti

pembentukan model global dari sekumpulan model lokal [1, 2, 3].

2.2.1 *Label Skew* pada *Federated Learning*

Label skew merupakan salah satu bentuk heterogenitas data pada *Federated Learning*. Kondisi ini terjadi ketika distribusi kelas pada setiap *client* tidak sama dengan distribusi kelas global. Pada skenario tersebut, sebagian *client* dapat memiliki sampel dari beberapa kelas tertentu saja, sedangkan kelas lain tidak muncul atau hanya muncul dalam jumlah sangat kecil. Akibatnya, *training* lokal pada *client* tidak selalu merepresentasikan keseluruhan *task* klasifikasi yang ingin dipelajari oleh model global [3, 4, 5]. Dampak utama dari kondisi ini adalah munculnya bias pada model lokal. Karena model lokal hanya melihat distribusi data masing-masing *client*, pembaruan model dapat bergerak mengikuti kelas yang dominan pada *client* tersebut. Pergeseran ini dapat menghasilkan *local model drift*, sehingga model global hasil agregasi menjadi kurang optimal ketika informasi kelas tertentu hilang atau kurang terwakili pada sebagian *client* [1, 19].

Label skew banyak muncul pada skenario *edge device* karena setiap *device* dapat mengumpulkan data dari konteks penggunaan yang berbeda. Satu *device* dapat lebih sering merekam aktivitas atau objek tertentu, sedangkan *device* lain merekam kategori yang berbeda. Gambar 2.1 menunjukkan kondisi ketika data lokal pada *client* memperlihatkan *label skew* dan *scarcity*.



Gambar 2.1. *Edge devices* sebagai *client* dalam *Federated Learning*, dengan *local data* yang mengalami *label skew* dan *scarcity*

Sumber: [1]

Gambar 2.1 menekankan dua kondisi yang menjadi dasar masalah, yaitu

distribusi label yang tidak merata dan jumlah data lokal yang terbatas. Gambar tersebut menunjukkan posisi *client* sebagai *device* yang menyimpan data lokal masing-masing. Karena data tidak dikumpulkan secara terpusat, setiap *client* dapat memiliki cakupan kelas dan jumlah sampel yang berbeda. Kondisi ini membuat agregasi model lokal pada FL menjadi lebih sulit dibandingkan pembelajaran terpusat.

Pendekatan pertama yang digunakan untuk mengurangi *client drift* akibat *label skew* adalah perbaikan *learning objective* selama *local training*. Selain *classification loss*, FedProx [19] menambahkan regularisasi untuk membatasi perbedaan antara parameter model lokal dan parameter model global. MOON [27] memanfaatkan *contrastive learning* untuk memperbesar jarak antara *low-dimensional features* dan kelas lain sehingga proses pembelajaran *feature* dapat ditingkatkan. FedDecorr [28] menambahkan regularisasi terhadap *dimensional collapse* untuk mengompensasi kategori yang tidak tersedia pada data lokal, sedangkan FedNTD [20] memberikan penalti terhadap perubahan distribusi logit kelas *non-ground-truth* yang diprediksi oleh model global dan model lokal. FedLC [29] melakukan *re-scaling* pada logit untuk menghasilkan *calibrated loss*, sementara FedRS [30] membatasi *softmax* agar pembaruan terhadap kelas yang tidak muncul pada data lokal dapat dikurangi.

Pendekatan kedua yang digunakan untuk menangani *label skew* adalah *data augmentation* dengan memanfaatkan informasi tambahan dari distribusi global. FedData [4] membagikan sebagian kecil data lokal secara global bersama parameter model sehingga performa FedAvg dapat ditingkatkan, tetapi pengumpulan *private raw data* tersebut dapat mengurangi manfaat *privacy* dari *Federated Learning*. Oleh karena itu, beberapa pendekatan lain menggunakan representasi global tambahan yang dianggap memiliki risiko *privacy* lebih rendah dibandingkan *raw data*. FedMix [22] dan FedBR [31] membagikan data hasil agregasi *mini-batch* secara global, sedangkan CCVR [32] membagikan *low-dimensional features* kepada *server* untuk melakukan kalibrasi model global pada sisi *server*. *Low-dimensional features* tersebut juga dikenal sebagai *class prototypes* dan dapat digunakan untuk mengurangi *local classifier bias*. Selain itu, FedGen, VHL, dan FedFed memanfaatkan informasi tambahan berbasis representasi tambahan, data sintetis, atau *feature* untuk membantu proses *local learning* [1, 21, 23, 33]. Meskipun tidak membagikan *raw data* secara langsung, pendekatan yang bergantung pada data sintetis tetap dipengaruhi oleh jumlah dan kualitas data sintetis yang dihasilkan [1].

2.2.2 *Data Scarcity* pada *Federated Learning*

Data scarcity pada *Federated Learning* merujuk pada kondisi ketika setiap *client* hanya memiliki jumlah data lokal yang terbatas. Kondisi ini umum terjadi pada *edge device* karena data dikumpulkan secara individual oleh *device* atau pengguna tertentu, sehingga jumlah sampel yang tersedia pada satu *client* dapat jauh lebih kecil dibandingkan kebutuhan ideal untuk melatih model lokal. Keterbatasan data juga dapat terjadi karena proses akuisisi data tidak selalu berlangsung terus-menerus, label sulit diperoleh, atau hanya sebagian kecil data yang dapat digunakan untuk *training* [1, 26].

Pada FedAvg, kontribusi setiap *client* terhadap model global dipengaruhi oleh jumlah data lokal yang dimilikinya. *Client* dengan data lebih sedikit memperoleh bobot agregasi lebih kecil, sebagaimana ditunjukkan pada Persamaan 2.1. Namun, *data scarcity* tidak hanya memengaruhi bobot agregasi. Ketika data lokal terlalu sedikit, model lokal dapat terlalu menyesuaikan diri terhadap sampel *training* yang terbatas. Akibatnya, model lokal berpotensi mengalami *overfitting* dan tidak mampu melakukan generalisasi dengan baik terhadap data uji, meskipun data uji tersebut berasal dari distribusi yang sama [1, 2].

Masalah *data scarcity* menjadi lebih berat ketika kondisi tersebut terjadi pada banyak *client* secara bersamaan. Jika seluruh *client* memiliki data lokal yang kecil, agregasi model lokal tidak otomatis memperbaiki generalisasi model global. Hal ini terjadi karena *server* menggabungkan pembaruan dari model-model lokal yang sama-sama dilatih pada informasi terbatas. Dengan demikian, model global dapat mengalami konvergensi yang lebih lambat atau performa yang lebih rendah karena sinyal pembelajaran yang diterima dari *client* tidak cukup kaya untuk merepresentasikan distribusi global [1].

Salah satu cara untuk mengurangi dampak *data scarcity* adalah menyediakan informasi tambahan yang dapat membantu *local training* tanpa harus sepenuhnya memusatkan *raw data*. Informasi tambahan tersebut dapat berupa data global, data hasil augmentasi, *synthetic data*, *class prototypes*, atau representasi *feature* yang dibagikan. Pendekatan berbasis *data augmentation* seperti FedMix dan FedData menunjukkan performa yang lebih baik dibandingkan metode yang hanya memperbaiki *loss* karena *client* memperoleh sinyal tambahan untuk memperluas variasi *local training* [1]. Namun, bentuk informasi yang dibagikan tetap perlu dipertimbangkan karena semakin dekat informasi tersebut dengan *raw data*, semakin besar pula risiko *privacy* yang dapat muncul [21, 4].

2.2.3 *Privacy-Preserving Feature Sharing*

Intermediate representation pada jaringan saraf masih dapat memuat informasi dari *raw input*. Dalam pembelajaran terdistribusi, *activation* atau *smashed data* yang dibagikan kepada pihak lain dapat menimbulkan *information leakage* dan memungkinkan *reconstruction attack* [12, 13]. Representasi yang terbentuk pada jaringan konvolusional bersifat bertingkat. Lapisan awal umumnya masih menangkap pola dasar dari *input*, sedangkan lapisan yang lebih dalam membentuk representasi yang lebih abstrak dan *invariant* [34, 35].

Pada mekanisme *feature sharing*, objek yang dibagikan bukan *raw data*, melainkan representasi numerik yang dihasilkan oleh model. Meskipun demikian, representasi tersebut tetap berasal dari data lokal dan dapat mempertahankan pola tertentu dari *input*. Oleh karena itu, pemilihan objek yang dibagikan, posisi *layer*, dan metode pengukuran keterkaitan digunakan untuk mengevaluasi risiko *privacy*.

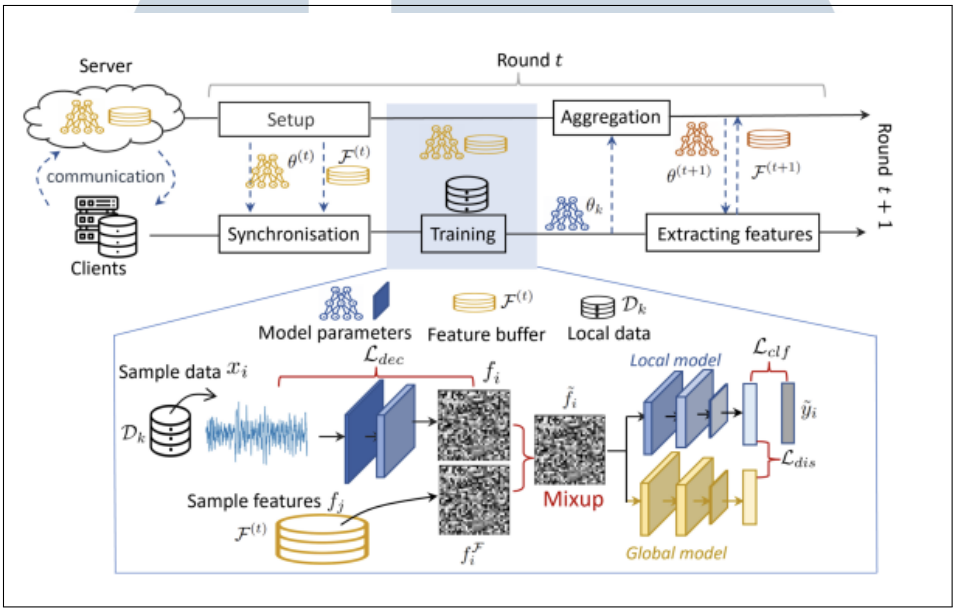
2.3 FLea

FLea merupakan metode *Federated Learning* yang dirancang untuk menangani *label skew* dan *data scarcity* tanpa membagikan *raw data* antar *client*. Pada kondisi tersebut, *local training* sering kali hanya memperoleh informasi dari kelas atau sampel yang terbatas, sehingga model lokal dapat mengalami *overfitting* dan bergerak menjauh dari distribusi global. FLea mengatasi masalah tersebut dengan menambahkan mekanisme *feature sharing* berbasis *global feature buffer*, yaitu penyimpanan *feature* yang berisi pasangan *activation* dan label dari beberapa *client* [1].

Global feature buffer digunakan sebagai sumber informasi tambahan selama *local training*. *Client* tidak hanya melatih model menggunakan *dataset* lokal D_k, tetapi juga memanfaatkan *feature* yang dikumpulkan dari *client* lain pada *communication round* sebelumnya. Dengan cara ini, *client* dapat memperoleh sinyal pembelajaran dari kelas yang lebih beragam tanpa harus menerima *raw data* milik *client* lain. *Feature* yang dibagikan tetap berada pada tingkat representasi model, sehingga informasi yang digunakan untuk membantu *training* tidak berbentuk *input* asli [1].

Alur umum FLea pada satu *communication round* ditunjukkan pada Gambar 2.2. Pada awal *training*, model global diinisialisasi secara acak dan *feature buffer* masih kosong. Oleh karena itu, *communication round* pertama hanya menggunakan

data lokal masing-masing *client*. Setelah *communication round* berikutnya dimulai, *server* mengirimkan parameter model global $\theta^{(t)}$ dan *feature buffer* $F^{(t)}$ kepada *client* terpilih $K^{(t)}$. Pada awal *communication round*, θ_k merupakan salinan lokal dari model global, yaitu $\theta_k \leftarrow \theta^{(t)}$. Setelah *local training* selesai, θ_k berubah menjadi parameter lokal hasil optimisasi *client* dan dikirim kembali ke *server* untuk diagregasi menjadi model global baru [1].



Gambar 2.2. Alur umum FLea pada *communication round* ke-t

Sumber: [1]

Gambar 2.2 memperlihatkan hubungan antara jalur model dan jalur *feature buffer*. Jalur model digunakan untuk mengirim parameter global, menjalankan *local training*, dan mengembalikan parameter lokal ke *server*. Jalur *feature buffer* digunakan setelah model global diperbarui, yaitu ketika *client* mengekstrak *activation* baru dan mengirimkannya untuk membentuk buffer *communication round* berikutnya. Alur ini menunjukkan bahwa $F^{(t)}$ digunakan untuk membantu *training* pada *communication round* ke-t, sedangkan *activation* baru yang diekstrak setelah agregasi digunakan untuk membentuk $F^{(t+1)}$.

Prosedur FLea pada Algoritma 1 memperlihatkan bahwa proses *training* terdiri dari dua tahap utama pada setiap *communication round*. Tahap pertama adalah *training* dan agregasi model, yaitu ketika *client* memperbarui parameter lokal menggunakan *loss* yang melibatkan data lokal dan *feature buffer*. Tahap kedua adalah pembaruan *feature buffer*, yaitu ketika *client* menerima model global terbaru, mengekstraksi *feature* tanpa menghitung gradien, lalu mengirimkan *feature* tersebut

ke *server* untuk membentuk buffer global pada *communication round* berikutnya [1].

Algorithm 1: FLea [1]

Input: Local *dataset* D_k, randomly initialized model $\theta^{(1)}$
Output: *Global model* $\theta^{(T)}$

foreach *round* $t = 1, 2, \ldots, T$ **do**
  Server samples *client*s $K^{(t)}$ and broadcasts $\theta_k \leftarrow \theta^{(t)}$
  Server broadcasts $F^{(t)}$ to $K^{(t)}$
    `// Skip if` $t = 1$
  foreach *client* $k \in K^{(t)}$ *in parallel* **do**
    for *local epoch* $e = 1, 2, \ldots, E$ **do**
      for *local batch* $b = 1, 2, \ldots$ **do**
        Sample one data *batch* B, one *feature batch* B^F, and $\beta_i \sim \text{Beta}(a, a)$
        Generate augmentation and update *local model*: $\theta_k \leftarrow \theta_k - \eta \nabla L(\theta_k)$ according to the FLea training objective
          `// Skip augmentation if` $t = 1$
    Client k sends θ_k to *server*
  Server aggregates θ_k to $\theta^{(t+1)}$ using FedAvg aggregation
  foreach *client* $k \in K^{(t)}$ *in parallel* **do**
    Client k receives the new model $\theta^{(t+1)}$
    Client k extracts without *gradient*s and sends $F_k^{(t+1)}$ to *server*
  Server aggregates $F_k^{(t+1)}$ and update the *global feature buffer* to $F^{(t+1)}$

Algoritma 1 memperjelas *input* dan *output* prosedur, yaitu *dataset* lokal D_k, model awal $\theta^{(1)}$, dan model global akhir $\theta^{(T)}$. Algoritma tersebut menunjukkan bahwa augmentasi dengan buffer belum dapat dilakukan pada *communication round* pertama karena $F^{(t)}$ masih kosong. Setelah *communication round* pertama, buffer mulai digunakan sebagai *feature batch* B^F pada *local training*. Urutan ini menunjukkan bahwa *feature buffer* bukan pengganti model agregasi, melainkan mekanisme tambahan yang masuk ke tahap *local training*.

Selain membantu *client* memperoleh informasi dari distribusi yang lebih luas, FLea juga memperhatikan risiko *privacy* dari *feature* yang dibagikan.

Feature yang dibagikan tidak hanya diekstraksi untuk mempertahankan informasi klasifikasi, tetapi juga dilatih dengan *decorrelation loss* agar hubungan antara *activation* dan *raw input* dapat dikurangi. Dengan demikian, FLea tidak hanya memanfaatkan *feature buffer* sebagai mekanisme untuk memperkaya *local training*, tetapi juga menempatkan pengurangan informasi yang dapat dikaitkan dengan *input* sebagai bagian dari desain *feature sharing* [1].

2.3.1 *Feature Buffer*

Feature buffer berisi pasangan *activation-label* yang dikumpulkan dari beberapa *client*. *Activation* merupakan *output intermediate layer* dari model, sedangkan label atau target label adalah kelas dari data lokal yang digunakan untuk menghasilkan *activation* tersebut. Jika parameter model dibagi pada *layer* ke-l, maka bagian awal model $\theta[:l]$ digunakan untuk mengekstrak *activation*, sedangkan bagian akhir model $\theta[l:]$ digunakan untuk melanjutkan proses prediksi dari *activation* tersebut. Jika $x_i \in D_k$ merupakan sampel lokal pada *client* ke-k, *activation* yang diekstraksi dari *layer* ke-l dapat ditulis sebagai $f_i^F = \theta[:l](x_i)$. Dengan demikian, *feature buffer* lokal berisi pasangan *activation* dan label, yaitu (f_i^F, y_i^F) [1].

Setiap *client* tidak membagikan seluruh data lokalnya, tetapi hanya memilih sebagian data lokal untuk diekstrak menjadi *feature*. Proporsi *feature sharing* dinyatakan dengan α dari data lokal *client*. Simbol α pada bagian ini merujuk pada rasio data lokal yang dipilih untuk membentuk *feature buffer*. Dengan demikian, hanya sebagian kecil *activation* yang dikirimkan ke *server*. Mekanisme ini berbeda dari pembagian *raw data* karena informasi yang dibagikan sudah berada pada ruang representasi model. Pada awal *communication round* ke-t, *server* memiliki *global feature buffer* $F^{(t)}$ yang dibentuk dari pembaruan buffer pada *communication round* sebelumnya. Buffer tersebut dikirimkan kepada *client* terpilih untuk membantu *local training* pada *communication round* ke-t. Setelah *local training* dan agregasi model selesai, *client* mengekstrak *activation* baru yang kemudian dikumpulkan *server* untuk memperbarui *global feature buffer* menjadi $F^{(t+1)}$ [1].

Pada sisi *client*, pembentukan buffer dimulai dari data lokal yang telah dipilih berdasarkan rasio *feature sharing*. Model lokal digunakan dalam mode ekstraksi *feature* untuk menghasilkan *activation* dari *intermediate layer*. *Activation* tersebut tetap dipasangkan dengan label sampel asalnya karena buffer harus dapat digunakan sebagai sumber sinyal klasifikasi pada proses *local training*. Pada alur

dasar, pasangan yang dikirim kepada *server* adalah *activation* dan label, bukan *raw input*, parameter model, atau gradien.

Pada sisi *server*, *activation* yang diterima tidak digunakan sebagai pembaruan parameter model, tetapi disimpan bersama label berdasarkan identitas *client*. Isi buffer kemudian digabungkan menjadi koleksi *activation-label pair* yang dapat dikirim kembali kepada *client* terpilih pada *communication round* berikutnya. Dengan demikian, *feature buffer* berfungsi sebagai penyimpanan representasi bersama, sedangkan agregasi parameter model tetap dilakukan melalui jalur agregasi federasi yang terpisah.

Karena *local data* pada satu *client* dapat mengalami *label skew*, *feature* dari satu *client* belum tentu mencakup seluruh kelas. Namun, ketika *feature* dari beberapa *client* digabungkan, *global feature buffer* $F^{(t)}$ memiliki peluang lebih besar untuk mencakup kelas yang lebih beragam. Dengan demikian, buffer dapat membantu *client* yang kekurangan kelas atau sampel lokal. Isi $F^{(t)}$ diperbarui menjadi $F^{(t+1)}$ pada setiap *communication round* sehingga informasi yang dibagikan mengikuti model global terbaru [1].

2.3.2 *Client Local Training*

Pada *local training*, *client* terpilih menerima model global $\theta^{(t)}$ dan *feature buffer* $F^{(t)}$ dari *server*. Parameter *client* mula-mula disalin dari model global sebagai $\theta_k \leftarrow \theta^{(t)}$, kemudian diperbarui selama *local training*. Data lokal D_k dan *feature buffer* $F^{(t)}$ dibagi menjadi *batch* berukuran sama, yaitu *batch* data lokal B dan *batch feature* B^F. Augmentasi dilakukan pada *representation space*, bukan pada *raw input*. Data lokal terlebih dahulu diproses sampai *intermediate layer* untuk menghasilkan *feature* lokal, kemudian *feature* tersebut dicampur dengan *feature* dari buffer [1].

Augmentasi *feature* dilakukan dengan mencampurkan *feature* lokal dan *feature* dari *global feature buffer*. Untuk sampel ke-i, *feature* lokal f_i dicampurkan dengan *feature buffer* f_i^F menggunakan bobot β_i sehingga terbentuk *feature* hasil augmentasi $\tilde{f}_i$. Label juga dicampurkan dengan prinsip yang sama sehingga menghasilkan $\tilde{y}_i$. Proses tersebut mengikuti Persamaan 2.2.

$$\begin{aligned} \tilde{f}_i &= \beta_i f_i + (1-\beta_i) f_i^F, \\ \tilde{y}_i &= \beta_i y_i + (1-\beta_i) y_i^F. \end{aligned} \tag{2.2}$$

Pada Persamaan 2.2, f_i adalah *feature* lokal dari sampel ke-i yang dihasilkan oleh *intermediate layer*, sedangkan f_i^F adalah *feature* dari *feature buffer*. Simbol y_i adalah label data lokal dan y_i^F adalah label pasangan dari *feature buffer*. Nilai β_i diambil dari distribusi Beta simetris, yaitu $\beta_i \sim \text{Beta}(a,a)$, dan berada pada rentang $[0,1]$. Hasil augmentasi ditulis sebagai $\tilde{f}_i$ untuk *feature* dan $\tilde{y}_i$ untuk label. Karena *batch* data lokal dan *batch feature buffer* diacak, kombinasi antara *feature* lokal dan *feature* global dapat bervariasi pada setiap *batch*. Dengan demikian, *client* memperoleh variasi *training* yang lebih luas tanpa perlu menerima *raw data* dari *client* lain [1].

Objective local training terdiri dari tiga komponen *loss*. Komponen pertama adalah *classification loss*, yaitu *loss* klasifikasi pada *feature* hasil augmentasi dari *feature* lokal dan *feature buffer*. Komponen kedua adalah *distilling loss*, yaitu komponen berbasis KL-divergence yang menjaga kedekatan prediksi model lokal terhadap model global. Komponen ketiga adalah *decorrelation loss*, yaitu komponen berbasis *distance correlation* untuk menurunkan keterkaitan antara *input* dan *activation* [1].

Ketiga komponen tersebut diringkas sebagai *training objective* pada Persamaan 2.3. Rumus ini dipertahankan karena menunjukkan peran setiap komponen utama tanpa menampilkan seluruh bentuk matematis dari masing-masing *loss*.

$$L = L_{clf}(B, B^F) + \lambda_1 L_{dis}(B, B^F) + \lambda_2 L_{dec}(B) \tag{2.3}$$

Pada Persamaan 2.3, L adalah *objective local training*, $L_{clf}(B, B^F)$ adalah *classification loss* pada *feature* hasil augmentasi, $L_{dis}(B, B^F)$ adalah *distilling loss* yang menjaga kedekatan prediksi model lokal terhadap model global, dan $L_{dec}(B)$ adalah *decorrelation loss*. Simbol B menunjukkan *batch* data lokal, sedangkan B^F menunjukkan *batch feature* yang diambil dari *feature buffer*. Simbol λ_1 dan λ_2 merupakan bobot untuk *distilling loss* dan *decorrelation loss*. Parameter model lokal diperbarui melalui $\theta_k \leftarrow \theta_k - \eta \nabla_{\theta_k} L$, dengan η sebagai *learning rate*.

Secara konseptual, proses tersebut membuat *client* belajar dari dua sumber informasi. Sumber pertama adalah data lokal, yang tetap menjadi dasar *supervised learning* pada *client*. Sumber kedua adalah *feature buffer* $F^{(t)}$, yang dapat memperluas variasi representasi dan kelas yang tersedia selama *local training*. Tambahan *decorrelation loss* digunakan untuk menurunkan keterkaitan antara

activation yang dibagikan dan *input* asalnya [1].

2.3.3 *Model Aggregation* dan *Buffer Updating*

Setelah *local training* selesai, setiap *client* mengirimkan parameter model lokal ke *server*. Agregasi menggunakan strategi yang sama dengan FedAvg, yaitu rata-rata berbobot berdasarkan jumlah sampel lokal *client* yang berpartisipasi. Jika *client* memiliki jumlah data lebih besar, kontribusinya terhadap model global juga lebih besar. Dengan demikian, pembaruan model global $\theta^{(t+1)}$ tetap mengikuti mekanisme agregasi parameter yang sudah dijelaskan pada Persamaan 2.1 [1, 2].

Pada agregasi model, *server* mengumpulkan parameter dari setiap *selected client*, menghitung bobot berdasarkan jumlah sampel lokal, lalu membentuk parameter global baru sebagai rata-rata berbobot. Mekanisme ini mempertahankan prinsip dasar FedAvg. Perbedaan utamanya berada pada proses setelah agregasi, yaitu pembaruan *feature buffer* menggunakan *activation* yang diekstrak dari model global terbaru.

Setelah model global baru terbentuk, *server* mengirimkan kembali parameter terbaru kepada *client* terpilih untuk ekstraksi *feature*. *Client* mengekstrak *activation* tanpa menghitung gradien, kemudian mengirimkan *activation* dan label kepada *server*. *Server* mengganti buffer lama dengan *feature* dari *communication round* terbaru sehingga *global feature buffer* $F^{(t+1)}$ siap digunakan pada *communication round* berikutnya.

Dengan urutan tersebut, pembaruan model dan pembaruan buffer saling bergantung. Model global yang baru digunakan untuk mengekstrak *activation* yang akan masuk ke buffer berikutnya, sedangkan buffer berikutnya digunakan kembali untuk membantu *local training* pada *communication round* selanjutnya. Iterasi ini terus berjalan sampai jumlah *communication round* selesai atau model global mencapai konvergensi [1].

2.4 *Differential Privacy* dan *Gaussian Mechanism*

Differential Privacy (DP) adalah pendekatan *privacy* yang bertujuan agar distribusi *output* suatu mekanisme tidak terlalu bergantung pada satu unit data tertentu [14, 15]. Dengan kata lain, dua masukan yang hanya berbeda pada satu unit yang dilindungi seharusnya menghasilkan distribusi *output* yang berdekatan. Tujuan utamanya adalah membatasi tambahan informasi tentang unit tersebut yang

dapat diperoleh pihak lain dari *output* yang dilepas.

Gaussian Mechanism adalah salah satu mekanisme dalam DP yang melindungi nilai numerik dengan menambahkan *noise* dari distribusi Gaussian atau distribusi normal [15]. Sebelum *noise* ditambahkan, nilai yang akan dilepas biasanya dibatasi terlebih dahulu agar sensitivitasnya terkontrol. Pada objek berbentuk vektor, pembatasan tersebut dapat dilakukan menggunakan *L2 clipping* [18]. Dengan demikian, *Gaussian Mechanism* menggabungkan pembatasan sensitivitas dan penambahan *Gaussian noise* agar *output* yang dilepas tidak terlalu mudah dikaitkan kembali dengan nilai aslinya.

Pada *feature sharing*, objek yang dilindungi adalah *intermediate activation* yang dibagikan melalui *feature buffer* [1]. *Intermediate activation* merupakan representasi numerik yang dihasilkan model setelah memproses data, tetapi sebelum menghasilkan prediksi akhir. Meskipun bukan *raw input*, representasi ini tetap dapat membawa informasi dari data asal dan berpotensi menimbulkan *information leakage* [12, 13]. Oleh karena itu, perlindungan pada bagian ini dilakukan dengan membatasi nilai *activation* menggunakan *L2 clipping*, lalu menambahkan *Gaussian noise* pada sisi *client* sebelum *activation* dikirim ke *server* [15, 18].

2.4.1 *Differential Privacy* pada Sisi *Client*

Differential Privacy merupakan kerangka *privacy* yang digunakan untuk membatasi informasi yang dapat disimpulkan dari *output* suatu proses komputasi. Prinsip utamanya adalah bahwa perubahan pada satu *input* tidak boleh membuat *output* berubah terlalu besar secara statistik [15]. Dengan kata lain, pihak yang melihat *output* diharapkan sulit memastikan apakah satu *input* tertentu ikut memengaruhi hasil atau tidak.

Dalam *Differential Privacy*, proses yang menghasilkan *output* disebut mekanisme acak atau *randomized mechanism*. Mekanisme ini disebut acak karena *output*-nya tidak hanya ditentukan oleh *input*, tetapi juga oleh nilai acak yang ditambahkan selama proses komputasi. Nilai acak tersebut biasanya berasal dari distribusi probabilitas tertentu, misalnya distribusi Laplace atau distribusi Gaussian [15]. Jadi, sifat acak pada mekanisme bukan berarti prosesnya tidak terkontrol, tetapi berarti terdapat *noise* yang dibangkitkan berdasarkan aturan distribusi tertentu.

Salah satu bentuk *Differential Privacy* adalah approximate (ε, δ)-DP, yaitu *Differential Privacy* dengan parameter ε dan δ. Parameter ε mengatur batas

perubahan multiplikatif pada distribusi *output* ketika unit yang dilindungi berubah. Parameter δ merupakan kelonggaran aditif yang mengizinkan sejumlah kecil massa probabilitas berada di luar batas multiplikatif tersebut; parameter ini bukan probabilitas bahwa pembangkitan *noise* gagal [15].

Nilai ε dan δ menentukan tingkat perlindungan *privacy* yang diinginkan, sehingga berpengaruh terhadap besar *noise* yang perlu ditambahkan pada *output*. Nilai ε yang lebih kecil biasanya membutuhkan *noise* yang lebih besar sehingga *output* lebih sulit dikaitkan dengan *input* tertentu, sedangkan nilai ε yang lebih besar membutuhkan *noise* yang lebih kecil tetapi perlindungan *privacy*-nya menjadi lebih lemah. Sebagai pedoman pemilihan parameter, nilai δ sebaiknya lebih kecil daripada kebalikan jumlah data *training*, yaitu $\delta < \frac{1}{N}$, dengan N menyatakan jumlah data *training* [18].

Pemilihan parameter DP bergantung pada konteks pelepasan informasi dan konsekuensi terhadap utilitas, sehingga tidak cukup ditentukan hanya dari satu metrik [36]. Penelitian ini lebih dahulu mengendalikan sensitivitas melalui pemilihan batas *clipping* C, kemudian menguji nilai δ, dan terakhir mengevaluasi *privacy budget* (ε). Pada *Gaussian Mechanism*, hubungan tersebut terlihat dari skala *noise* yang bergantung pada ketiga parameter tersebut. Untuk batas *clipping* dan δ yang tetap, nilai ε yang lebih kecil menghasilkan *noise* lebih besar sehingga representasi yang dibagikan mengalami perubahan lebih kuat, sedangkan nilai ε yang lebih besar menghasilkan *noise* lebih kecil sehingga utilitas model cenderung lebih terjaga [15].

Pada sisi *client*, *Differential Privacy* diterapkan sebelum suatu informasi dikirim kepada *server*. Tujuannya adalah agar *server* menerima *output* yang sudah diproses oleh mekanisme acak, bukan nilai asli yang langsung berasal dari *input* lokal. Dengan pendekatan ini, perlindungan diberikan pada informasi yang dilepas oleh *client*.

2.4.2 *Gaussian Mechanism* untuk Perlindungan Informasi Lokal

Informasi lokal yang berbentuk *activation vector* perlu diproses sebelum dilepas ke *server* karena nilai tersebut tetap berasal dari data lokal dan dapat membawa informasi dari sampel asal. Meskipun bukan *raw input*, *activation vector* masih merupakan representasi numerik yang dihasilkan model dari data pada sisi *client*. Oleh karena itu, perlindungan diperlukan agar nilai *activation* asli tidak dilepas secara langsung melalui *output* yang dikirim.

Perlindungan menggunakan *Gaussian Mechanism* dilakukan melalui dua tahap utama, yaitu membatasi norma *activation* menggunakan *L2 clipping*, kemudian menambahkan *Gaussian noise* yang dikalibrasi berdasarkan sensitivitas, ε, dan δ [15, 18]. Misalkan $f_i \in \mathbb{R}^d$ adalah *activation vector* untuk sampel ke-i. Simbol i menunjukkan indeks sampel, sedangkan $\mathbb{R}^d$ menunjukkan bahwa f_i merupakan vektor berdimensi d yang setiap elemennya berupa bilangan riil. Simbol C menyatakan batas maksimum norma L2 yang diperbolehkan. Hasil *clipping* terhadap f_i dinyatakan sebagai $\bar{f}_i$ pada Persamaan 2.4 [18].

$$\bar{f}_i = \frac{f_i}{\max\left(1, \frac{\|f_i\|_2}{C}\right)} \tag{2.4}$$

Pada Persamaan 2.4, $\|f_i\|_2$ merupakan norma L2 dari *activation vector* f_i, yaitu ukuran panjang vektor tersebut. Untuk suatu vektor $v = [v_1, \ldots, v_d]$, norma L2 dihitung menggunakan $\|v\|_2 = \sqrt{\sum_{j=1}^{d} v_j^2}$. Simbol j menunjukkan indeks elemen vektor, sedangkan d menunjukkan jumlah seluruh elemen dalam vektor.

Fungsi $\max(\cdot)$ memilih nilai yang lebih besar antara 1 dan $\frac{\|f_i\|_2}{C}$. Jika $\|f_i\|_2 \leq C$, penyebut bernilai 1 sehingga nilai f_i tidak berubah. Ketentuan ini juga berlaku ketika $\|f_i\|_2 = 0$, sehingga tidak diperlukan konstanta numerik tambahan pada penyebut. Sebaliknya, jika $\|f_i\|_2 > C$, seluruh elemen f_i diperkecil secara proporsional sampai norma hasilnya sama dengan C.

Operator pada Persamaan 2.4 menggunakan bentuk matematis *per-example clipping*, tetapi objek yang dibatasi berupa *activation vector* dari setiap sampel. Setelah proses *clipping*, hasilnya memenuhi $\|\bar{f}_i\|_2 \leq C$. Dengan demikian, setiap *activation vector* memiliki norma yang terbatas sebelum *noise* ditambahkan.

Batas *clipping* C memengaruhi tingkat perubahan terhadap *activation* dan besarnya *noise*. Nilai C yang lebih kecil menyebabkan lebih banyak *activation* mengalami *clipping*, tetapi menghasilkan batas sensitivitas yang lebih rendah. Sebaliknya, nilai C yang lebih besar mempertahankan lebih banyak nilai *activation* asli, tetapi menghasilkan batas sensitivitas dan skala akhir *noise* yang lebih besar untuk nilai ε dan δ yang sama. Oleh karena itu, nilai C perlu disesuaikan dengan distribusi norma objek yang diproses dan tidak memiliki satu nilai tetap yang berlaku untuk seluruh arsitektur atau lapisan model.

Sensitivitas L2 menunjukkan perubahan maksimum keluaran mekanisme ketika satu *activation vector* digantikan oleh *activation vector* lain berdasarkan *replace-one adjacency*. Karena setiap *activation vector* yang telah melalui proses

clipping memiliki norma paling besar C, jarak L2 antara dua *activation vector* yang bersebelahan dapat dibatasi menggunakan pertidaksamaan segitiga.

Misalkan $\bar{f}_i$ dan $\bar{f}'_i$ merupakan dua *activation vector* hasil *clipping* yang saling menggantikan. Karena $\|\bar{f}_i\|_2 \leq C$ dan $\|\bar{f}'_i\|_2 \leq C$, jarak L2 antara keduanya tidak melebihi $2C$. Dengan demikian, batas sensitivitas L2 dinyatakan sebagai $S_f = 2C$. Batas tersebut berlaku karena satu *activation vector* dapat digantikan oleh *activation vector* lain yang arahnya berbeda, sementara norma masing-masing telah dibatasi paling besar C.

Setelah sensitivitas ditentukan, besar *Gaussian noise* dikalibrasi menggunakan parameter ε dan δ. Parameter ε membatasi perubahan relatif probabilitas keluaran pada dua masukan bersebelahan, sedangkan δ menyatakan probabilitas kegagalan tambahan yang diperbolehkan dalam jaminan (ε, δ)-*differential privacy*. Pada kalibrasi klasik *Gaussian Mechanism*, simbol σ menyatakan *noise multiplier* dan dihitung menggunakan Persamaan 2.5 [15, 18].

$$\sigma = \frac{\sqrt{2\ln(1.25/\delta)}}{\varepsilon} \tag{2.5}$$

Persamaan 2.5 mendefinisikan σ sebagai *noise multiplier* yang dihitung berdasarkan ε dan δ. Untuk suatu *activation vector* berdimensi d, vektor *noise* n_i dibangkitkan menurut distribusi pada Persamaan 2.6.

$$n_i \sim \mathcal{N}\left(0, S_f^2 \sigma^2 I_d\right) \tag{2.6}$$

Pada Persamaan 2.6, parameter pertama bernilai nol dan menyatakan vektor rata-rata. Parameter kedua merupakan matriks kovariansi. Simbol I_d adalah matriks identitas berdimensi $d \times d$, yang menunjukkan bahwa setiap elemen *noise* memiliki varians yang sama dan tidak memiliki kovariansi dengan elemen lainnya. Dengan demikian, setiap elemen *noise* memiliki *standard deviation* $S_f\sigma$ dan varians $S_f^2\sigma^2$.

Nilai ε yang lebih kecil menghasilkan *noise multiplier* σ yang lebih besar ketika S_f dan δ tetap, sehingga *standard deviation* $S_f\sigma$ juga meningkat. Kondisi tersebut menghasilkan perubahan yang lebih besar pada *activation* akibat penambahan *noise*, tetapi dapat mengurangi informasi yang berguna bagi model. Sebaliknya, nilai ε yang lebih besar menghasilkan *noise* yang lebih kecil sehingga perubahan terhadap *activation* lebih terbatas, tetapi tingkat perlindungan *privacy*-nya menjadi lebih lemah.

Setelah proses *clipping* dan pembangkitan *noise*, *private activation* dibentuk dengan menambahkan n_i pada $\bar{f}_i$, sebagaimana ditunjukkan pada Persamaan 2.7.

$$f_i^{\text{priv}} = \bar{f}_i + n_i \tag{2.7}$$

Nilai f_i^{priv} merupakan *activation vector* yang telah mengalami pembatasan norma dan penambahan *noise* acak. Nilai tersebut dapat dilepas sebagai pengganti *activation* asli, sedangkan jaminan *privacy*-nya ditentukan oleh definisi *adjacency*, batas sensitivitas S_f, serta parameter ε dan δ yang digunakan.

2.5 Klasifikasi Human Activity Recognition

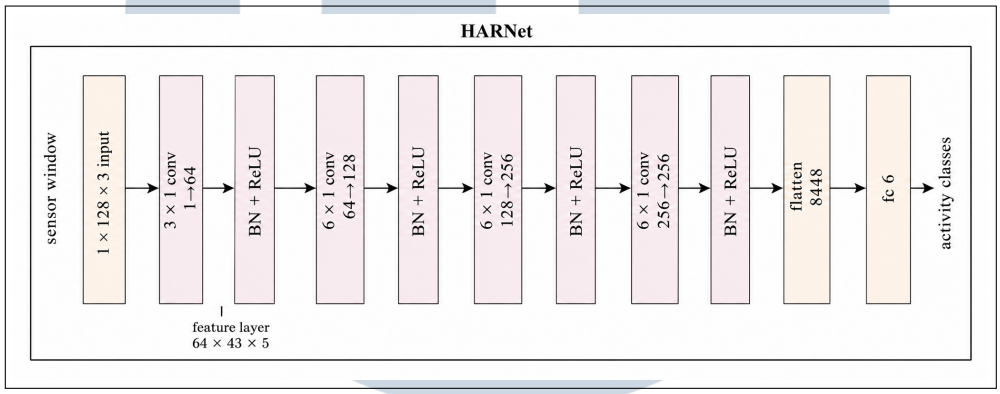
Human Activity Recognition merupakan tugas klasifikasi yang bertujuan mengenali aktivitas manusia berdasarkan data yang merekam gerakan tubuh. Pada sistem berbasis *mobile device* atau *wearable device*, data tersebut umumnya berasal dari sensor inersia seperti accelerometer dan gyroscope. Sensor tersebut menghasilkan sinyal deret waktu yang menggambarkan perubahan gerak, orientasi, atau percepatan tubuh selama pengguna melakukan aktivitas tertentu [37, 38].

Berbeda dari klasifikasi citra yang memproses piksel dua dimensi, klasifikasi aktivitas manusia berbasis sensor perlu membaca pola temporal pada sinyal. Data sensor biasanya dipotong menjadi beberapa *window* berukuran tetap agar setiap sampel dapat merepresentasikan segmen aktivitas dalam durasi tertentu. Setiap *window* kemudian digunakan sebagai *input* model untuk mempelajari pola gerakan, misalnya perubahan intensitas sinyal ketika pengguna berjalan, duduk, berdiri, atau melakukan aktivitas lain.

Tantangan utama pada klasifikasi *Human Activity Recognition* adalah kemiripan pola antaraktivitas dan variasi gerakan antarindividu. Aktivitas yang berbeda dapat menghasilkan sinyal yang saling mendekati, sedangkan aktivitas yang sama dapat tampak berbeda karena perbedaan gaya berjalan, posisi *device*, atau kondisi pengambilan data. Oleh karena itu, model klasifikasi perlu mampu mengekstrak *feature* yang stabil dari sinyal sensor, bukan hanya menghafal pola dari satu kelompok pengguna tertentu [38].

2.6 CNN HARNet

HARNet merupakan arsitektur CNN untuk klasifikasi aktivitas manusia berbasis sensor. Model ini memproses sinyal aktivitas sebagai *sensor window*, sehingga pola lokal pada dimensi temporal dapat dipelajari melalui *layer* konvolusi. CNN sesuai untuk data sensor karena filter konvolusi dapat menangkap pola gerakan lokal, sedangkan *layer* yang lebih dalam dapat membentuk representasi yang lebih abstrak dari rangkaian sinyal [39, 38]. Susunan umum blok konvolusi dan *classifier* HARNet ditunjukkan pada Gambar 2.3.



Gambar 2.3. Arsitektur HARNet untuk klasifikasi aktivitas manusia berbasis sensor

Sumber: [1]

Gambar 2.3 menunjukkan alur pemrosesan sinyal sensor dari *input window* menuju prediksi kelas aktivitas. Arsitektur tersebut memperlihatkan adanya beberapa blok konvolusi yang ditempatkan sebelum *classifier*. Setiap blok konvolusi berfungsi untuk mengekstraksi *feature* bertingkat dari sinyal input. Hasil ekstraksi tersebut kemudian diproses lebih lanjut oleh *classifier* untuk menghasilkan prediksi akhir berupa kelas aktivitas. Dengan demikian, model tidak hanya bergantung pada representasi input awal, tetapi juga pada transformasi *feature* pada beberapa tahap konvolusi.

Secara umum, HARNet menerima *input* dalam bentuk tensor yang merepresentasikan satu *sensor window*. Pada konfigurasi yang digunakan, bentuk *input* adalah $1 \times 128 \times 3$. Dimensi pertama menunjukkan jumlah *channel input* untuk `Conv2d`, sedangkan 128×3 menunjukkan panjang *window* dan jumlah kanal sensor. Empat blok konvolusi menghasilkan *feature* bertingkat dengan jumlah *channel* yang semakin besar. Setelah blok konvolusi terakhir, *output* diubah menjadi vektor melalui operasi *flatten*, kemudian dipetakan ke kelas aktivitas melalui *fully*

connected layer. Arsitektur tersebut diringkas pada Tabel 2.2.

Tabel 2.2. Arsitektur CNN HARNet [1]

Index	*Input*	Operator	Kernel	Stride
1	$1 \times 128 \times 3$	`conv2d`	$(3,1)$	$(3,1)$
2	$64 \times 43 \times 5$	`conv2d`	$(6,1)$	$(2,1)$
3	$128 \times 20 \times 7$	`conv2d`	$(6,1)$	$(2,1)$
4	$256 \times 9 \times 9$	`conv2d`	$(6,1)$	$(2,1)$

Tabel 2.2 memperinci bentuk *input*, operator, ukuran *kernel*, dan *stride* pada setiap blok konvolusi. Blok pertama menerima *input* berukuran $1 \times 128 \times 3$ dan menghasilkan *activation* berukuran $64 \times 43 \times 5$. Bentuk tersebut menjadi *input* bagi blok kedua sekaligus menjadi ukuran *activation* yang dibagikan melalui *feature buffer*. Selanjutnya, blok kedua dan ketiga menghasilkan *activation* berukuran $128 \times 20 \times 7$ dan $256 \times 9 \times 9$. Blok keempat menerima *input* berukuran $256 \times 9 \times 9$ dan menghasilkan *activation* akhir berukuran $256 \times 3 \times 11$.

2.7 Metrik Evaluasi

2.7.1 *Accuracy*

Accuracy digunakan untuk mengukur proporsi prediksi benar terhadap seluruh sampel pengujian. Pada klasifikasi multi-kelas, prediksi model ditentukan berdasarkan kelas dengan nilai *logit* atau probabilitas tertinggi. Prediksi dianggap benar apabila kelas yang diprediksi sama dengan label asli.

$$Accuracy = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\hat{y}_i = y_i) \tag{2.8}$$

Pada Persamaan 2.8, N adalah jumlah sampel pengujian, y_i adalah label sebenarnya untuk sampel ke-i, dan $\hat{y}_i$ adalah kelas yang diprediksi model. Fungsi indikator $\mathbf{1}(\hat{y}_i = y_i)$ bernilai 1 apabila prediksi benar dan 0 apabila prediksi salah. Dengan demikian, *accuracy* merupakan jumlah prediksi yang benar dibagi dengan jumlah seluruh sampel pengujian pada klasifikasi enam kelas.

2.7.2 *Distance Correlation*

Distance correlation merupakan ukuran ketergantungan statistik antara dua variabel acak. Berbeda dari korelasi linear, *distance correlation* dapat mendeteksi hubungan linear maupun *non-linear* melalui jarak antar-sampel [6, 40]. Nilainya berada pada rentang $0 \leq c \leq 1$. Nilai yang mendekati 0 menunjukkan keterkaitan yang lebih rendah, sedangkan nilai yang mendekati 1 menunjukkan keterkaitan yang lebih kuat.

Jika x menyatakan *raw input* dan f menyatakan *intermediate activation*, *distance correlation* dapat dituliskan melalui Persamaan 2.9.

$$c(x,f) = \frac{\nu^2(x,f)}{\sqrt{\nu^2(x,x)\nu^2(f,f)}} \tag{2.9}$$

Pada Persamaan 2.9, $\nu^2(x,f)$ merupakan *squared distance covariance* antara x dan f. Sementara itu, $\nu^2(x,x)$ dan $\nu^2(f,f)$ merupakan *squared distance variance* dari masing-masing variabel. $\sqrt{\nu^2(x,x)\nu^2(f,f)}$ berfungsi sebagai faktor normalisasi agar nilai korelasi tidak hanya dipengaruhi oleh skala jarak pada kedua variabel.

Perhitungan *distance correlation* diawali dengan membentuk matriks jarak *pairwise* dari setiap pasangan sampel pada x dan f. Kedua matriks jarak tersebut kemudian melalui proses *double centering* dengan mengurangi setiap elemen menggunakan rata-rata baris dan rata-rata kolom, kemudian menambahkan rata-rata keseluruhan matriks. Hasilnya digunakan untuk menghitung *squared distance covariance* dan *squared distance variance* pada Persamaan 2.9.

Dalam konteks pembagian *intermediate activation*, *distance correlation* digunakan sebagai indikator keterkaitan antara *raw input* dan representasi yang dibagikan [1, 12, 13, 41]. Nilai yang lebih rendah menunjukkan ketergantungan statistik yang lebih rendah pada sampel yang diukur.