SEC-301: Security Challenges in Modern AI Systems

Lecture 3 – AI-Specific Attack Prevention Techniques

Instructed By:

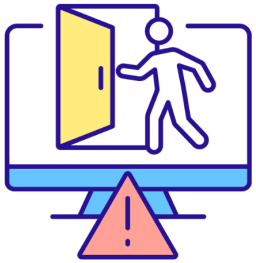
Dr. Charnon Pattiyanon

Assistant Director of IT and Instructor
CMKL University

Artificial Intelligence and Computer Engineering (AICE) Program

Recapitulations from the First Session

1. Data Poisoning
2. Model Inversion
3. Adversarial Examples
4. Evasion Attacks
5. Model Stealing
6. Backdoor Attacks
7. AI-Enhanced Social Engineering
8. Transfer Learning Attacks
9. Membership Inference Attacks

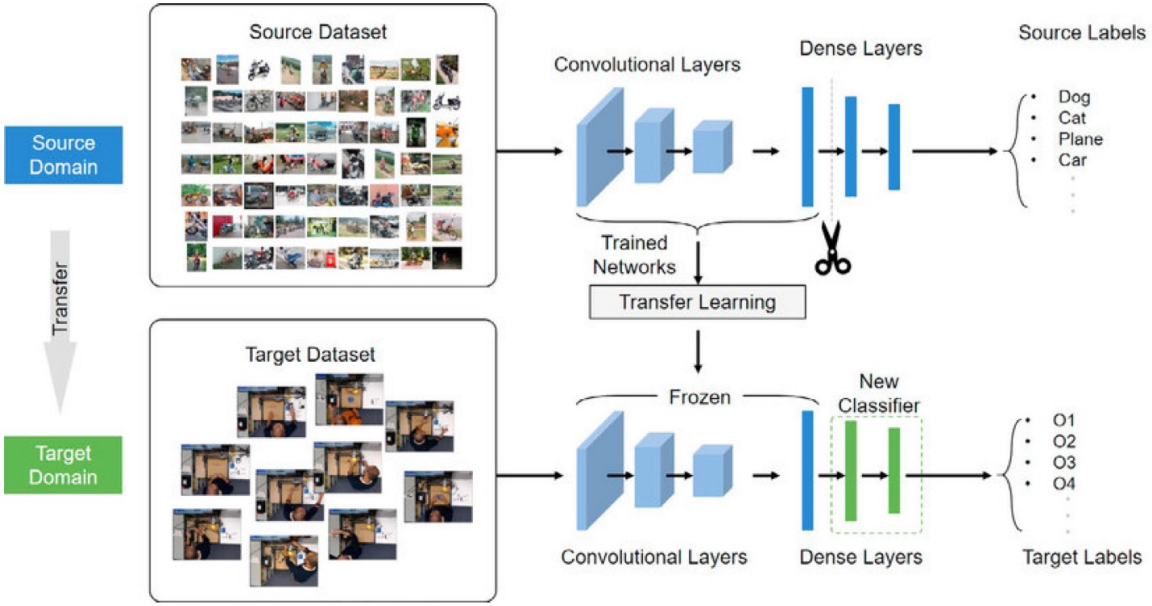


Class Outline

- Today, we will discuss some of existing preventive techniques for AI security risks.
    1. An Introduction to **Transfer Learning Techniques** and Their Use Cases.
    2. An Introduction to **Federated Learning Techniques** and Their Use Cases.
    3. An Introduction to **Generative Adversarial Network (GAN)** and Their Use Cases.
    4. An Introduction to **Homomorphic Encryption** and Their Use Cases.
    5. An Introduction to **Differential Privacy** and Their Use Cases.

1. Transfer Learning

- **Transfer learning** is a technique to train a machine learning model (**Student Model**) by reusing an existing model (**Teacher Model**) and fine-tuning the weights on a specific data set.
- The overall goal of transfer learning is ***to boost performance of the learner task*** (*Student Task*) through knowledge transfer from another task (*Teacher Task*).



1. Transfer Learning : Formal Definition

- The **formal definition** of transfer learning considers the knowledge transfer in the context of **source D_S** and **target D_T domains** with their corresponding tasks T_S , T_T.
- A domain D, a task T and the transfer learning process are defined as follows:

Definition 1: **Domain $D = \langle \mathcal{X}, P(X) \rangle$**, where $\mathcal{X}$ is a feature space, $P(X)$ a marginal distribution, and X is a learning sample with n feature vectors $X = \{x_1, x_2, \ldots, x_n\} \in \mathcal{X}$.

Definition 2: **Task $\mathcal{T} = \langle \mathcal{Y}, f(\cdot) \rangle$**, where $\mathcal{Y}$ is *a label space* and $f(\cdot)$ is **a decision function** learned from the feature vector and label pair $\{x_i , y_i\}$ where $x_i \in X$ and $y_i \in \mathcal{Y}$.

Definition 3: Given D_S , T_S and D_T , T_T , **transfer learning** is a process of improving the target decision function $f_T(\cdot)$ utilizing the knowledge from source **domain D_S** and *task* T_S , where $D_S \neq D_T$ or $T_S \neq T_T$.

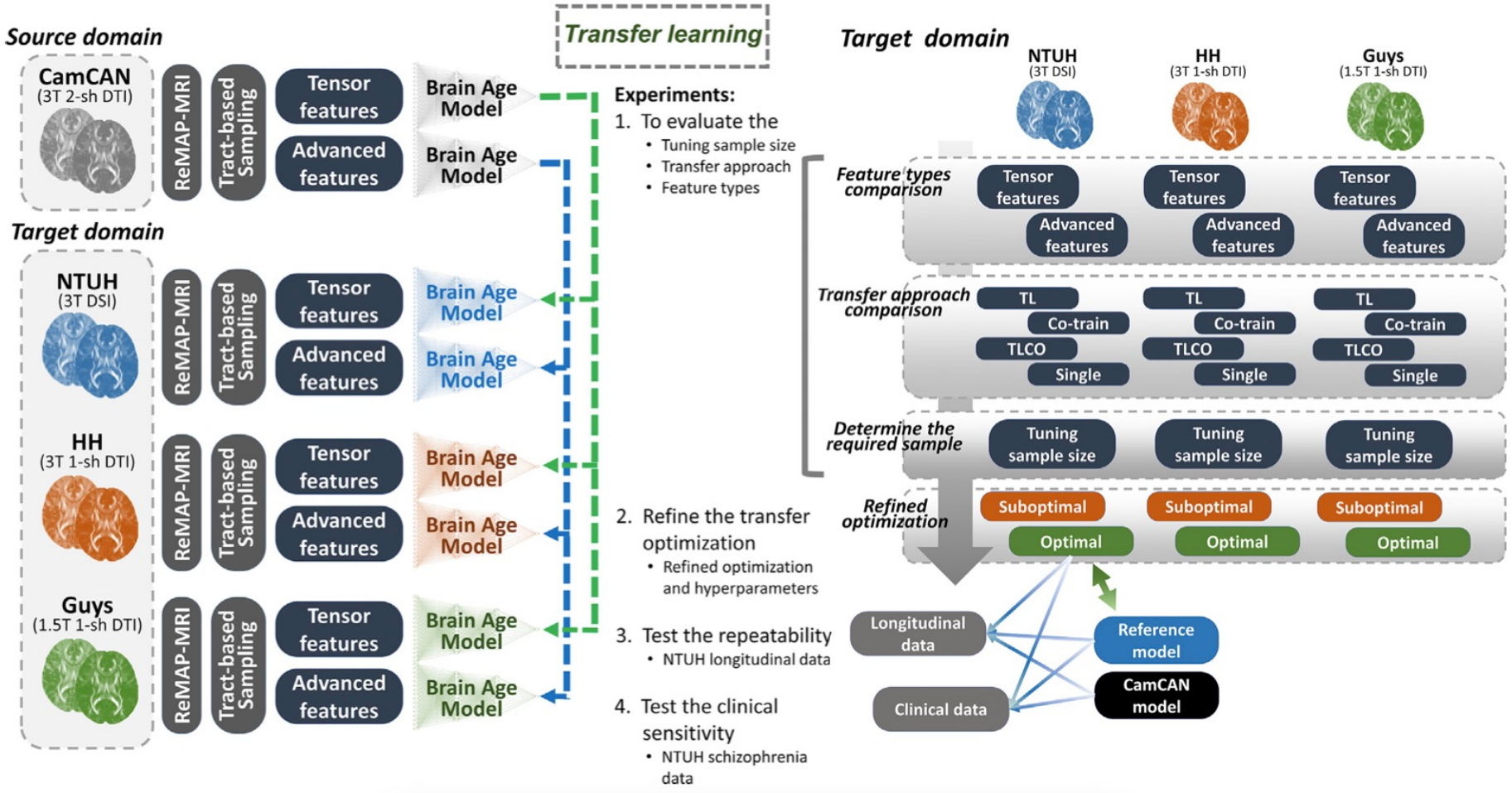
1. Transfer Learning : Categories

- There are **four categories** of approach to knowledge transfer between source and target tasks:
  - **Instance-based:** Directly reuse selected instances. Useful when source and target domains have the same features and labels. (For example, *a model instance to recognize traffic signs is transferred for parking signs*)
  - **Feature-based:** This approach aims at learning effective (or common) feature representations. (For example, *feature representation of images can be transferred between the classification of dogs and cats*)
  - **Parameter-based:** This approach is used when the source domain model and target domain model have some of the same optimal parameters. (For example, *the same image size is optimal for AI medical imaging and AI surveillance camara analysis*)
  - **Relational-based:** This approach is used when the relationships among the data in the source and target domains are similar, knowledge about these relationships can be transferred from the source domain to the target domain. (For example, *relationships between image colors and classes are similar for fruit and pet classifications*)

1. Transfer Learning : Benefits

- On a direct **performance** level, transfer learning is **beneficial** for:
  - Boosting generalization
  - Improving performance on the target task
  - Speeding up convergence
- **Transfer learning provides the most beneficial** when the model is pre-trained on **a source domain**, which is **very similar** to the **target domain**.
- **Other benefits:**
  - Saving cost,
  - Overcoming the lack of required dataset due to reasons related to:
    - ethics, data protection, and intellectual property.
  - In competitive settings, the element of speed in model development is an additional reason why business stakeholders use transfer learning.

1. Transfer Learning : Use Cases



1. Transfer Learning : How-To

- **Step 1: Pick the Right Source Model**

Domain	Common Practices	Key Resources
Computer Vision	Use ImageNet-trained backbones (ResNet, EfficientNet, Vision-Transformer).	PyTorch/TensorFlow Hub, HuggingFace 🤗 Models
Natural Language Processing	Fine-tune BERT, RoBERTa, GPT-2, or domain-specific models (ClinicalBERT, BioBERT).	Transformers library, HuggingFace 🤗 Models
Time-Series / Tabular	Leverage auto-encoders or pre-trained embeddings from large-scale datasets (e.g., M4, Kaggle).	AutoGluon, MMDL
Cross-Modal	Use multi-modal models (CLIP, ViLBERT) if the target task involves both vision & text.	CLIP, ViLBERT repos

1. Transfer Learning : How-To

- **Step 2: Prepare the Target Dataset**
  - Clean & Label – remove duplicate or noisy samples; ensure labels are correct.
  - Split – training (70–80 %), validation (10–15 %), test (10–15 %), ensuring no leakage.
  - Pre-process – match preprocessing of the source model:
    - Images – resize, normalize using the same mean & std.
    - Text – tokenization using the source tokenizer (BERTTokenizer etc.).
    - Tabular – scaling, one-hot encode categorical features.
  - Data Augmentation (optional) – e.g., random crops, flip, MixUp.

1. Transfer Learning : How-To

- **Step 3: Choose the Fine-tuning Strategy**

Strategy	When to Use	What to Freeze
Feature Extraction	Target dataset very small; source captures most knowledge.	Freeze all convolutional layers (backbone). Replace final dense/softmax layer.
Partial Fine-tuning	Medium-size dataset; need adaptation near output.	Unfreeze last X layers (e.g., last 2 ResNet blocks).
Full Fine-tuning	Large target dataset; architecture is appropriate.	Unfreeze all weights.
Parameter-Efficient Methods	Limited GPU memory, quick iterative experiments.	Use LoRA, Adapters, or PrefixTuning. Freeze backbone, train only low-rank adapters.

- Define learning rate schedule:
  - Lower LR for frozen layers (or zero).
  - Higher LR for newly added head or adapter layers.
  - Use cosine annealing or step-decay.

1. Transfer Learning : How-To

- **Step 4:** **Training and Validate**

```python
# Example PyTorch sketch
model = torchvision.models.resnet50(pretrained=True)

# 1. Replace the head
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, num_classes)

# 2. Freeze backbone if feature extraction
for param in model.parameters():
    param.requires_grad = False
for param in model.fc.parameters():   # unlock head
    param.requires_grad = True

# 3. Optimizer
optimizer = torch.optim.AdamW(filter(lambda p: p.requires_grad, model.parameters()),
                              lr=1e-3, weight_decay=0.01)

# 4. Scheduler
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=10)

# 5. Training loop with early-stopping
```

1. Transfer Learning : How-To

- **Step 5: Advanced Transfer-Learning Tweaks**

Technique	Use-Case	How to Apply
Knowledge Distillation	Compress a large teacher to a small student.	Train the student to match the teacher's logits while still training on the target labels.
Multimodal Transfer	Transfer from vision to vision-text.	Attach a cross-attention layer that fuses features, fine-tune end-to-end.
Domain Adaptation	Shift from source domain to target domain with different data distribution.	Use unsupervised adversarial networks (Domain-Adversarial Neural Network) or domain-adaptive batch norm.
Continual Transfer	Incrementally add new labeled data over time.	Use Elastic Weight Consolidation to prevent forgetting.

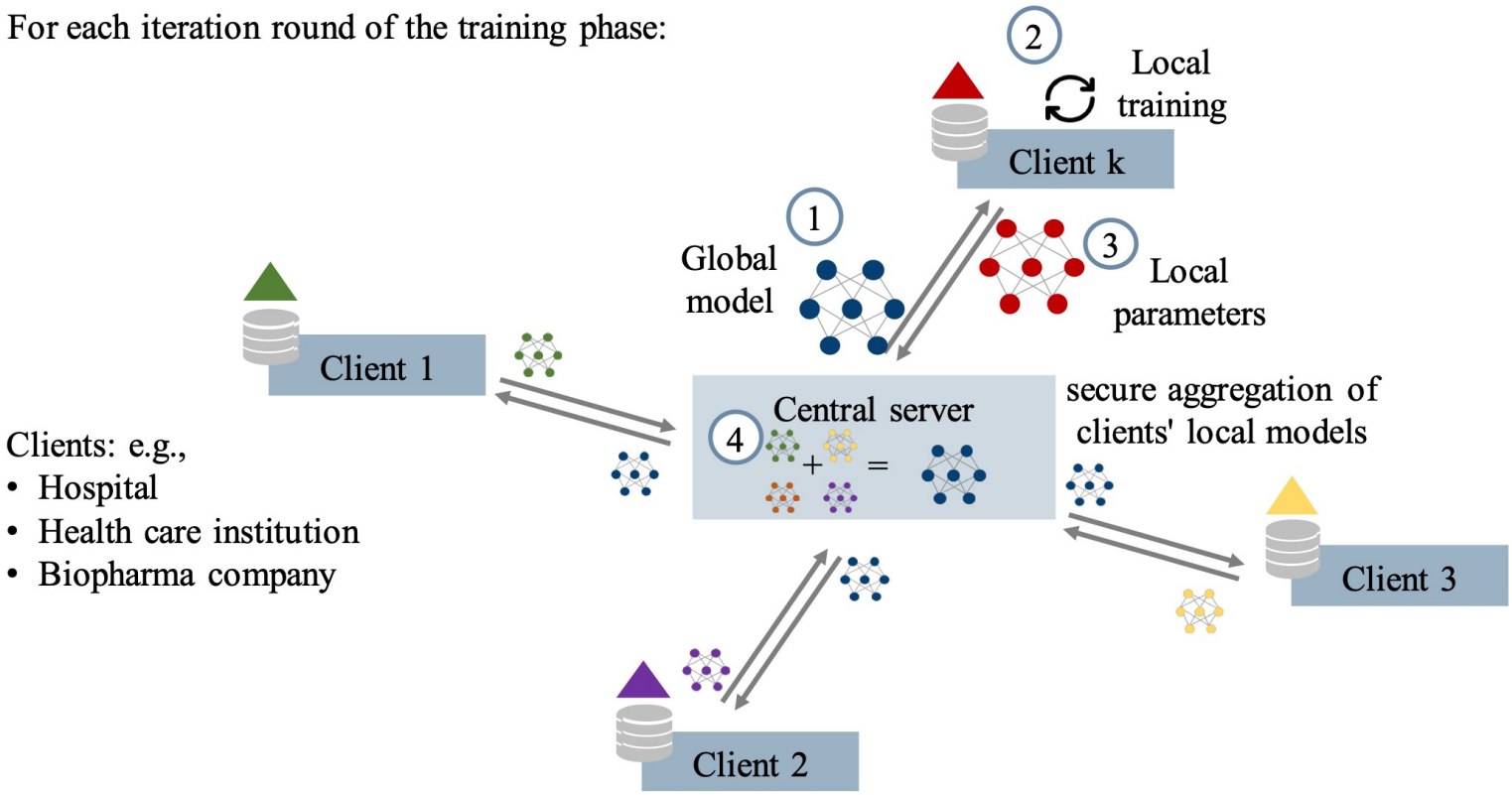
2. Federated Learning

- In some tasks or domains, **confidential data is necessary for training a ML model** that results the sufficient performance.
- In real-world, such *confidential data is distributed over various companies or data owners*.
  - For example, banking companies hold their own **customer financial data**.
  - These **confidential data cannot be shared directly** to conduct centralized learning.
- This is where the **federated learning** comes into plays.

> **Federate [V.]:** form or be formed into a single centralized unit, within which each state or organization keeps some internal autonomy.

> **Federated Learning (FL)** is based on the idea that multiple parties *do not directly share training data*, but only the insights obtained from the data that are necessary for learning.
> - Intuitively, this increases privacy as *no data directly leaves the data owners' secure storage*.

2. Federated Learning : An Overview



An overview of the most common federated learning setup. We depict **horizontal** federated learning in cross-silo setting here. A learning round starts by downloading the global model (**step 1**), then clients train the model locally on their data (**step 2**) and upload the new model parameters (**step 3**) to the server, which aggregates all clients' parameters into a new global model (**step 4**).

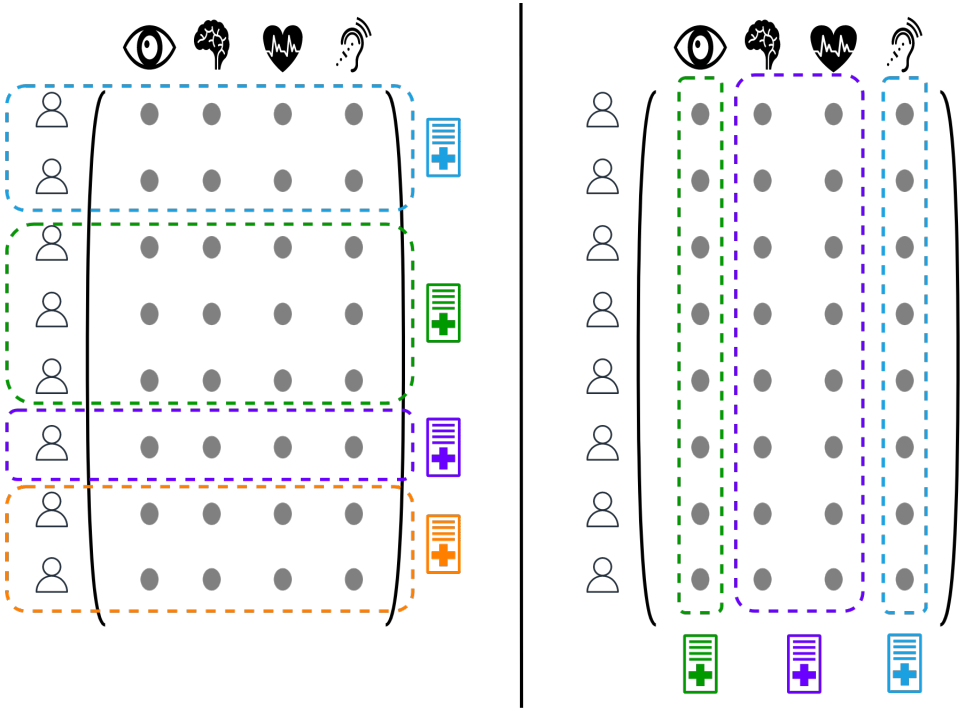
2. Federated Learning : Goals

- **Goal:** The local training and aggregation of results are **repeated until meeting convergence criteria**.
  - The accuracy on a held-out validation dataset stops increasing.
  - After a fixed number of rounds or computation time.

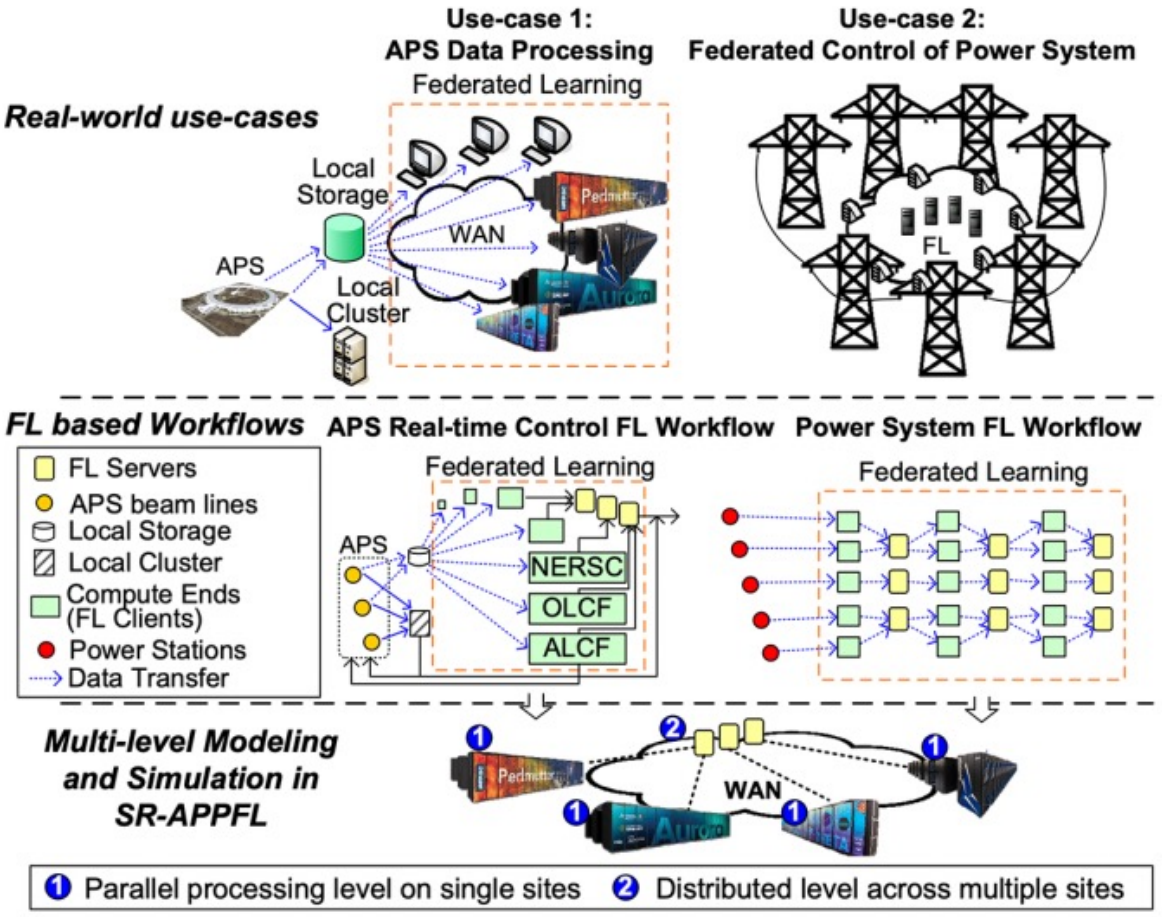
- **Aggregation** can either be done by:
  - Computing a (weighted) average of the local models learned parameters, often referred to as federated averaging in the literature.
  - If the model is trained with stochastic gradient descend (SGD), it is also possible to share the loss of the gradients instead, such that the gradients are averaged and not the learned weights.

2. Federated Learning : Setup Types

- Another important distinction in federated learning **setups** is between **horizontal** and **vertical** learning.
  - In **horizontal** learning, all clients *share the same feature space* but have *different data* generated from different entities
    - **Example**: Credit scoring tasks where datasets of different clients would typically contain data of different customers.
  - In **vertical** learning, all clients *share data about the same entities* but have *different feature spaces*.



2. Federated Learning : Use Cases



2. Federated Learning : How-To

1. **Define the Collaboration Scope**
   - **Horizontal FL** – same feature space, different subsets of users (e.g., smartphones).
   - **Vertical FL** – different feature sets, overlapping entities (e.g., hospital + insurer).
   - **Cross-Device FL** – many low-power devices with sporadic connectivity.
2. **Choose the Federated Learning Framework**
   - *TensorFlow Federated (Python)* – good for research and prototyping.
   - *PySyft (Python)* – supports secure aggregation, differential privacy.
   - *OpenMined Fractional* – lightweight for edge devices.
   - *Flower (Python)* – framework-agnostic, supports PyTorch, TensorFlow, Keras.

2. Federated Learning : How-To

3. Set Up the Infrastructure

- **Central Orchestrator** – server that aggregates gradients/weights.
- **Client SDK** – lightweight library that runs on each device/edge server.
- **Network** – typically HTTPS with mutual TLS or VPN for privacy.
- **Security Controls** – encrypted communication, secure enclaves for sensitive operations.

4. Design the Model and Training Loop

- **Global Model** – define initial weights or initialize randomly.
- **Client-Side Dataset** – ensure no personal identifiers can be inferred from the data itself.
- **Local Epochs** – each client trains for E epochs on its local data.
- **Maximal Rounds (R)** – number of communication rounds before convergence.

2. Federated Learning : How-To

5. Transport and Aggregation

- **Secure Aggregation** – use cryptographic protocols (additive homomorphic encryption or secure multi-party computation) to hide individual client updates.
- **Differential Privacy** – apply gradient clipping & Gaussian noise before sending to server.
- **Compression / Sparsification** – reduce bandwidth (e.g., top-k sparsification, quantization).

6. Model Update Workflow (Per Round)

```
for round in 1..R:
    1. Server broadcasts current global weights w_t to a random subset of clients C_t
    2. Each client i in C_t:
        a. Load local data D_i
        b. Train local model w_i = Train(w_t, D_i, E)
        c. Compute update Δ_i = w_i − w_t
        d. Apply clipping/DP, compress if necessary
        e. Send Δ_i (or w_i) back to server
    3. Server aggregates updates:
        w_{t+1} = Aggregate({Δ_i})   # e.g., weighted average by local dataset size
    4. Evaluate w_{t+1} on a held-out validation set (if available)
```

2. Federated Learning : How-To

7. Convergence Monitoring

7. Track global loss/accuracy on a public validation set provided by the orchestrator.
8. Stop when loss plateaus or reaches a target threshold.
9. Optional: early stopping based on maximum round count or communication budget.

8. Deployment of the Global Model

7. Broadcast the final weights to all clients.
8. Clients integrate the model into their local inference pipeline (e.g., on-device recommendation).
9. Periodically re-initiate FL cycles to adapt to drift.

9. Privacy & Security Audits

- Verify that secure aggregation works by checking for correct sum (e.g., via randomly inserted cryptographic beacons).
- Conduct a model inversion risk assessment: ensure DP noise or homomorphic encryption level meets the required budget.
- Validate that no data can be reconstructed from the compressed updates (tune compression rate accordingly).

10. Operational Considerations

- **Connectivity –** clients should fallback to local inference when offline.
- **Resource Constraints –** limit CPU/GPU usage and memory footprint of local training.
- **Device Heterogeneity –** use model personalization schemes (e.g., FedAvg+Personalized Layers or Per-Client Adapter Layers).

2. Federated Learning : How-To

Example Code Snippet (Flower, Python):

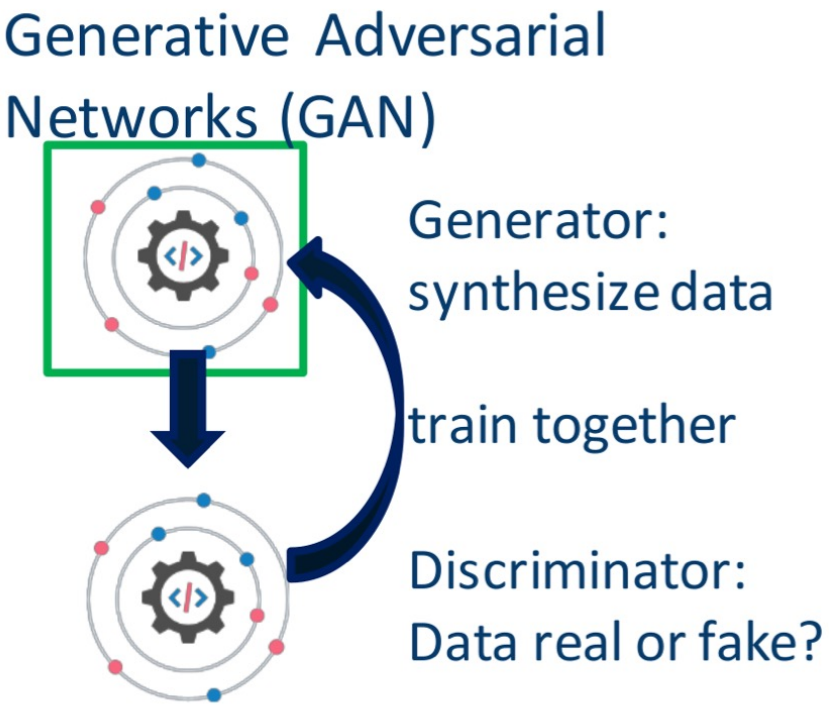
```python
import flwr as fl
import torch
import torch.nn as nn

# ------------ 1. client ------------ #
class StoreClient(fl.client.NumPyClient):
    def get_parameters(self, config):
        return [param.cpu().numpy() for param in model.parameters()]
    def fit(self, parameters, config):
        model.load_state_dict({k:v for k,v in zip(model.state_dict().keys(), parameters)})
        train()
        return [p.cpu().numpy() for p in model.parameters()], len(dataset), {}
    def evaluate(self, parameters, config):
        model.load_state_dict({k:v for k,v in zip(model.state_dict().keys(), parameters)})
        loss, acc = test()
        return loss, len(dataset), {"accuracy": acc}

# ------------ 2. server ------------ #
def main():
    client = StoreClient()
    fl.client.start_numpy_client(server_address="127.0.0.1:8080", client=client)

# Run client's training loop locally
if __name__ == "__main__":
    main()
```

3. Generative Adversarial Networks : An Overview

- Instead of training jointly on private data, a one-time effort can be made to train a model that can **generate an unlimited amount of synthetic data**.
- Such synthetic data should have **similar characteristics to the private training data**, but it should be generated in a privacy-preserving way.
- Synthetic data generation can be achieved using **Generative Adversarial Networks (GANs)** or **Autoencoders**.

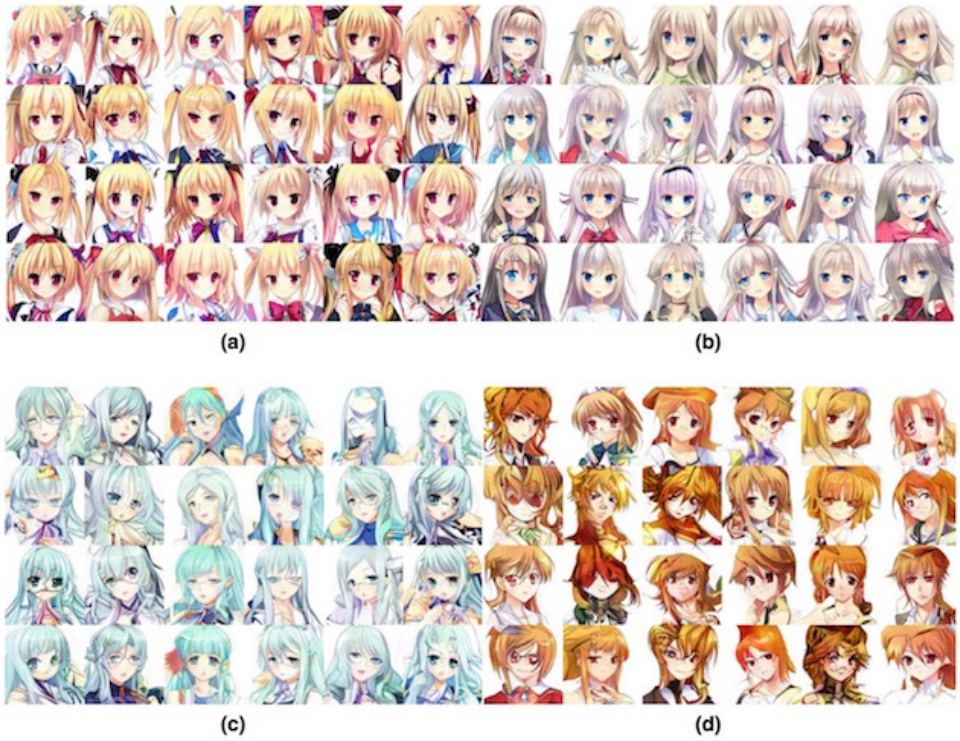
3. Generative Adversarial Networks : An Overview

- **Generative Adversarial Networks (GANs)** are composed of **two neural networks** that are trained jointly.
  - The **generator** *synthesizes data*, and the ***discriminator*** *judges how realistic the synthesis is* by learning to distinguish synthesized and real training data.
  - *A good prediction accuracy of the discriminator is fed back into the generator as a penalty, teaching the generator to synthesize more realistic outputs which the discriminator cannot differentiate from real data.*
  - **Only the generator is released** which can synthesize realistic data points given just randomly generated noise as input.



3. Generative Adversarial Networks : Use Cases

Example of Sketches to Color Photographs With pix2pix. Taken from Image-to-Image Translation with Conditional Adversarial Networks, 2016.

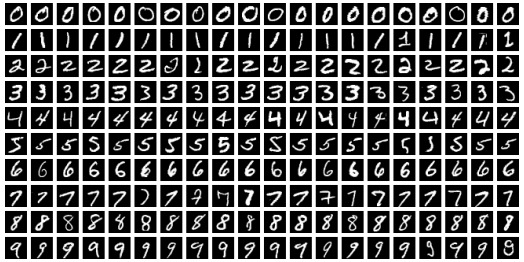


Example of GAN-Generated Anime Character Faces. Taken from Towards the Automatic Anime Characters Creation with Generative Adversarial Networks, 2017.

3. Generative Adversarial Networks : How-To & Example

- **Step 1: Decide on the Data & Architecture**

Item	What to choose	Reason
Dataset	MNIST (28×28 grayscale)	Small, well-known, fast to train
Latent vector size	100	Typical low-dimensional noise
Generator	3×Dense + Reshape → 3×Conv2DTranspose → Output	Produces 28×28 images, uses BatchNorm+ReLU
Discriminator	Conv2D → Conv2D → Flatten → Dense	Classifies “real” vs “fake”



3. Generative Adversarial Networks : How-To & Example

- **Step 2:** **Import Libraries**

```
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
import numpy as np
import matplotlib.pyplot as plt
```

- **Step 3:** **Load & Pre-process Data**

```
# MNIST: 0-1 pixel range
(x_train, _), _ = tf.keras.datasets.mnist.load_data()
x_train = (x_train.astype('float32') - 127.5) / 127.5   # [-1,1]
x_train = np.expand_dims(x_train, axis=-1)               # (60000, 28, 28, 1)
```

3. Generative Adversarial Networks : How-To & Example

- **Step 4:** Define the Generator

```
def build_generator(latent_dim=100):
    model = models.Sequential()
    model.add(layers.Dense(7*7*128, use_bias=False, input_shape=(latent_dim,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 128)))            # 7x7x128

    model.add(layers.Conv2DTranspose(128, (5,5), strides=(1,1), padding='same',
use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5,5), strides=(2,2), padding='same',
use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5,5), strides=(2,2), padding='same',
use_bias=False, activation='tanh'))
    return model
```

- **Step 5:** Define the Discriminator

```
def build_discriminator():
    model = models.Sequential()
    model.add(layers.Conv2D(64, (5,5), strides=(2,2), padding='same',
input_shape=[28,28,1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5,5), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))
    return model
```

3. Generative Adversarial Networks : How-To & Example

- **Step 6:** **Loss and Optimizer**

```python
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

def generator_loss(fake_output):
    # Adversarial loss only — wants discriminator to think fake is real
    return cross_entropy(tf.ones_like(fake_output), fake_output)

def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    return real_loss + fake_loss
```

```python
generator = build_generator()
discriminator = build_discriminator()

generator_optimizer = optimizers.Adam(1e-4)
discriminator_optimizer = optimizers.Adam(1e-4)
```

3. Generative Adversarial Networks : How-To & Example

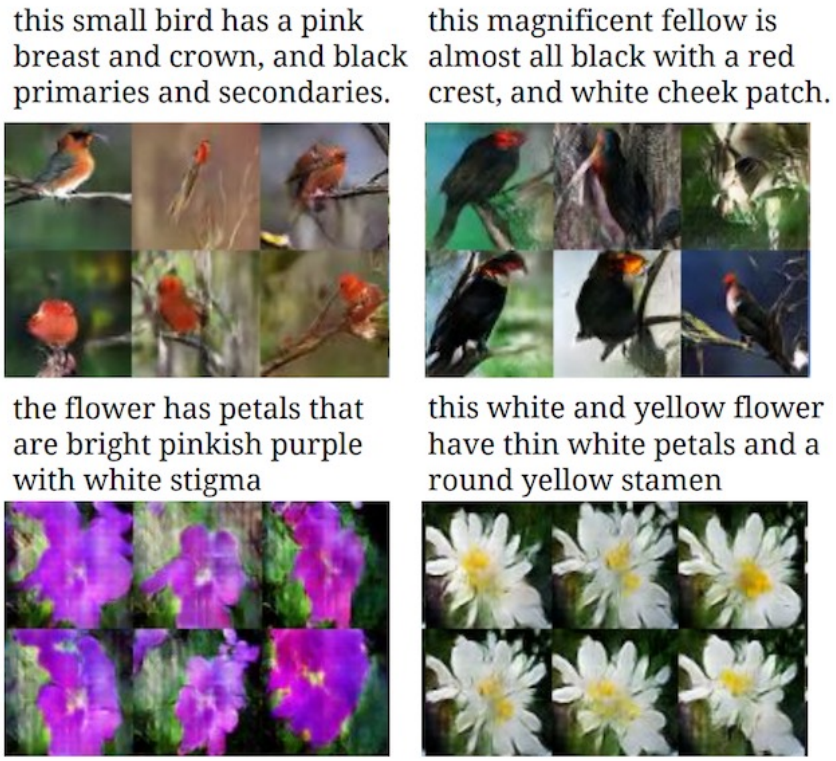
- **Step 7:** **Training Loop (Eager Execution)**

```python
BATCH_SIZE = 256
EPOCHS = 50
NOISE_DIM = 100
NUM_EXAMPLES_TO_GENERATE = 16

seed = tf.random.normal([NUM_EXAMPLES_TO_GENERATE, NOISE_DIM])

@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, NOISE_DIM])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)

        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)

        gen_loss = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss,
generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss,
discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator,
generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
discriminator.trainable_variables))
    return gen_loss, disc_loss
```

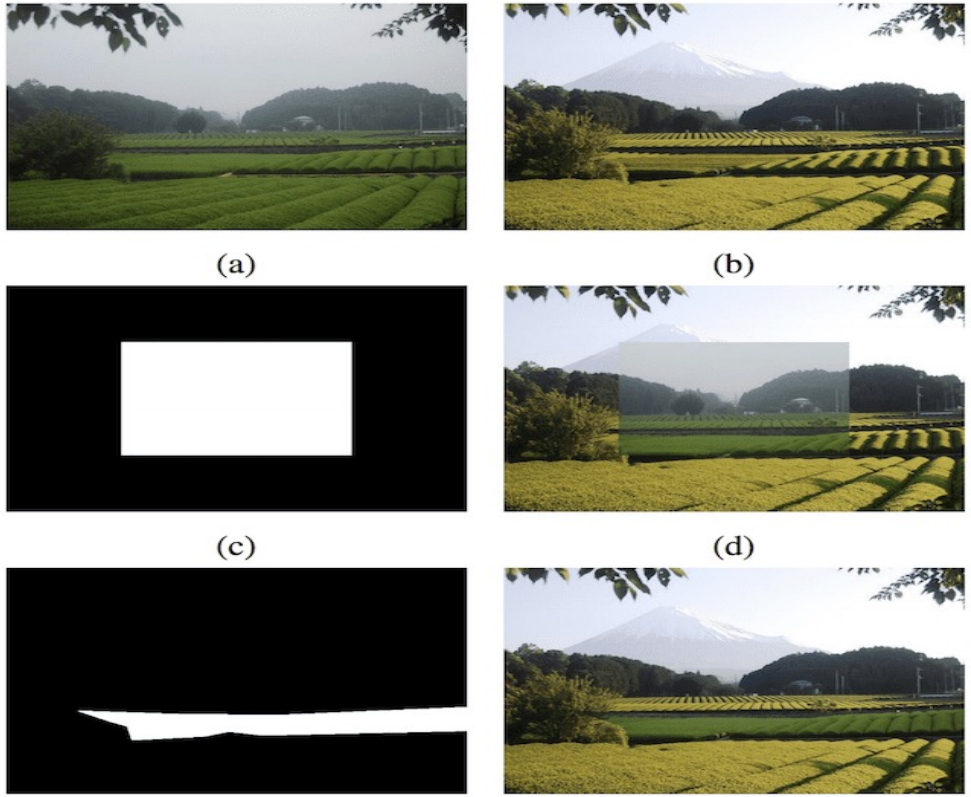
```python
def train(dataset, epochs):
    for epoch in range(epochs):
        for image_batch in dataset:
            g_loss, d_loss = train_step(image_batch)
        # generate and log images every epoch
        generate_and_log_images(generator, epoch + 1, seed)

def generate_and_log_images(model, epoch, test_input):
    predictions = model(test_input, training=False)
    fig = plt.figure(figsize=(4,4))
    for i in range(predictions.shape[0]):
        plt.subplot(4,4,i+1)
        plt.imshow((predictions[i, :, :, 0]*127.5 +
127.5).numpy().astype(np.uint8), cmap='gray')
        plt.axis('off')
    plt.suptitle(f'Epoch {epoch}')
    plt.savefig(f'images_at_epoch_{epoch:04d}.png')
    plt.close(fig)
```

```python
train_dataset =
tf.data.Dataset.from_tensor_slices(x_train).shuffle(60000).batch(BATCH_SIZE)
train(train_dataset, EPOCHS)
```

3. Generative Adversarial Networks : Use Cases



Example of Textual Descriptions and GAN-Generated Photographs of Birds and Flowers. Taken from Generative Adversarial Text to Image Synthesis.



Example of GAN-based Photograph Blending. Taken from GP-GAN: Towards Realistic High-Resolution Image Blending, 2017.

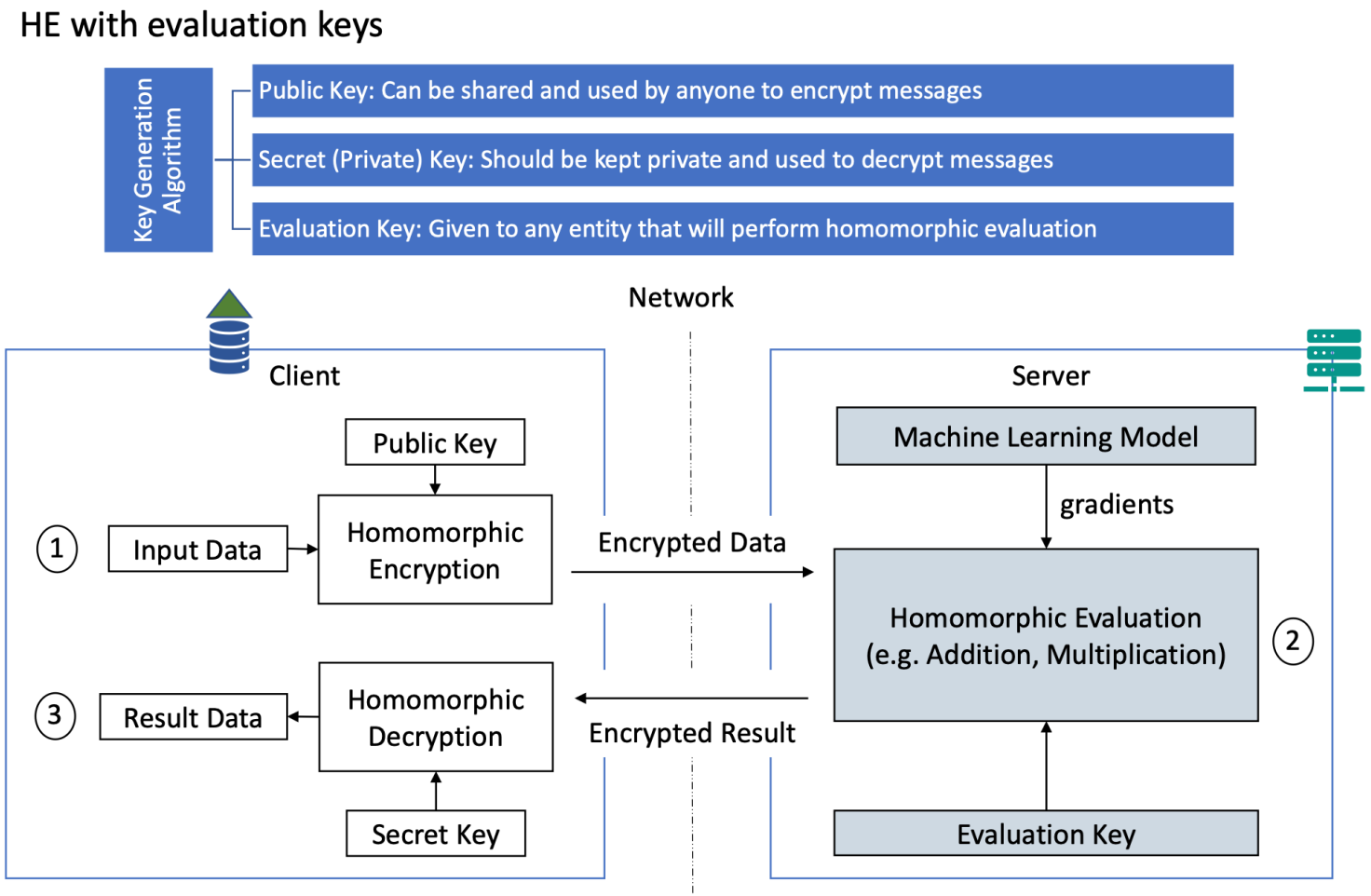
4. Homomorphic Encryption

- Another possible solution for providing confidential data to third parties is **the use of encrypted computation**.
- The main idea of encrypted computation is that *users share their data after they have encrypted it* in a way such that it cannot be decrypted.
- Subsequently, *the processing*, which includes the training and updating of ML models, *takes place on this encrypted data* and the corresponding results are extracted based on the shared user inputs.

- **Homomorphic Encryption (HE)** encompasses a set of encryption methods that *allow to perform computations on encrypted data*.
- Specifically, in homomorphic encryption schemes, the plaintext message is encrypted, and *computation is performed on the ciphertext*, so that the plaintext message remains secret.

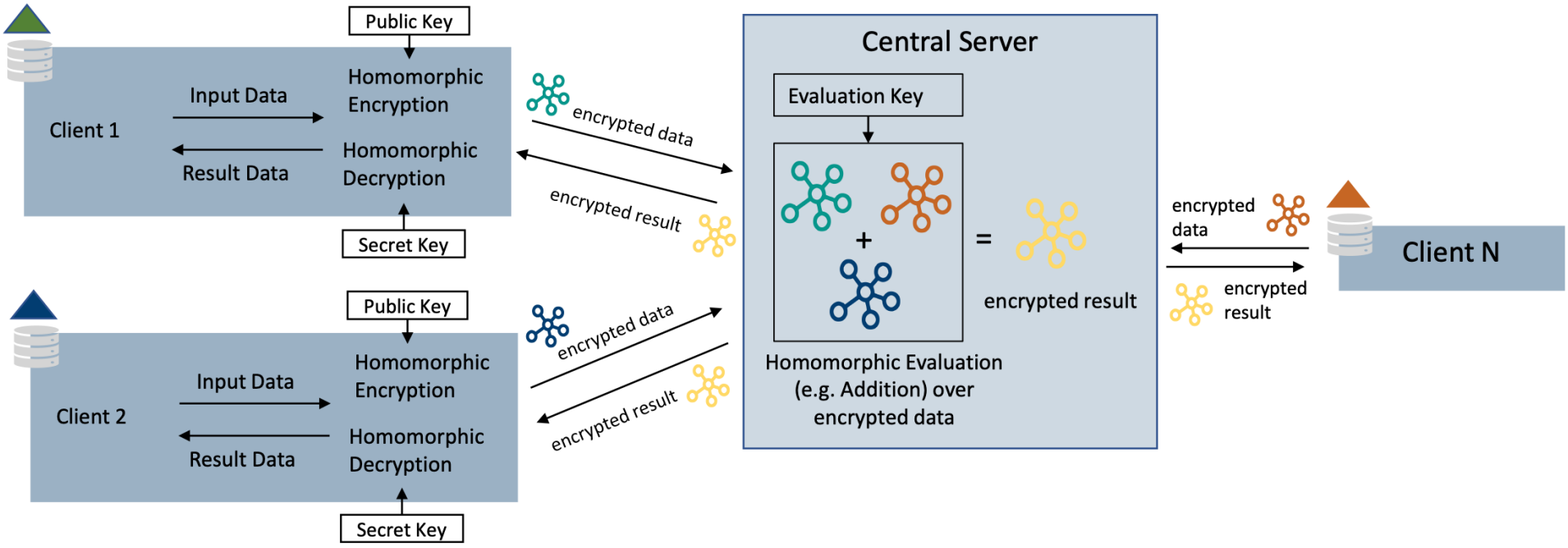
4. Homomorphic Encryption : An Overview

- **The Role of Homomorphic Encryption in Machine Learning:**



4. Homomorphic Encryption : An Overview

- **The Role of Homomorphic Encryption in Machine Learning:**



4. Homomorphic Encryption : Limitations

Limitations of Homomorphic Encryption for Machine Learning:

1. *Evaluating deep neural networks on homomorphically encrypted data is* ***expensive*** in terms of communication, memory as well as computation, making it a challenge to apply it in particular to deep learning which is characterized by its need for big data.
2. The ***computational overhead increases*** *with the message size* (i.e., the size of the ciphertext) and to the length of the keys that are being used for the cryptography.
3. The ***communication overhead*** *between client and server* ***increases*** with the length of the ciphertext.
4. A ***cryptographic infrastructure*** *for key generation and management is required* in order to enable HE-based ML processes.
5. Each *conversion from plaintext to ciphertext has some* ***noise*** associated with it. This can continue to increase each time that the ciphertexts are added or multiplied. As a result, the ***final ciphertext might no longer be decryptable***.

4. Homomorphic Encryption : Use Cases

Government Sectors:

FHE streamlines the often cumbersome processes that were traditionally required to maintain the confidentiality of investigations. Our innovative Zero Footprint Investigations solution is revolutionizing the way government agencies handle sensitive data related to investigations. This solution enables agencies to keep the subjects of their investigations completely confidential while still accessing and analyzing crucial data sources.

Zero Footprint Investigations Solution

Financial Services:

By leveraging FHE's capabilities, financial institutions can enhance their fraud detection and prevention efforts by tapping into data they originally would not have access to. This innovative approach not only strengthens security measures but also fosters global cooperation in combating financial crimes.

Fraud Detection and Prevention from Sensitive Financial Data

Healthcare Industry:

FHE provides a secure framework for sharing and analyzing sensitive medical data, addressing challenges related to privacy and data protection. The global pandemic in 2020 underscored the pressing need for enhanced collaboration among healthcare researchers and organizations.

Sharing and Analysis of Sensitive Medical Data

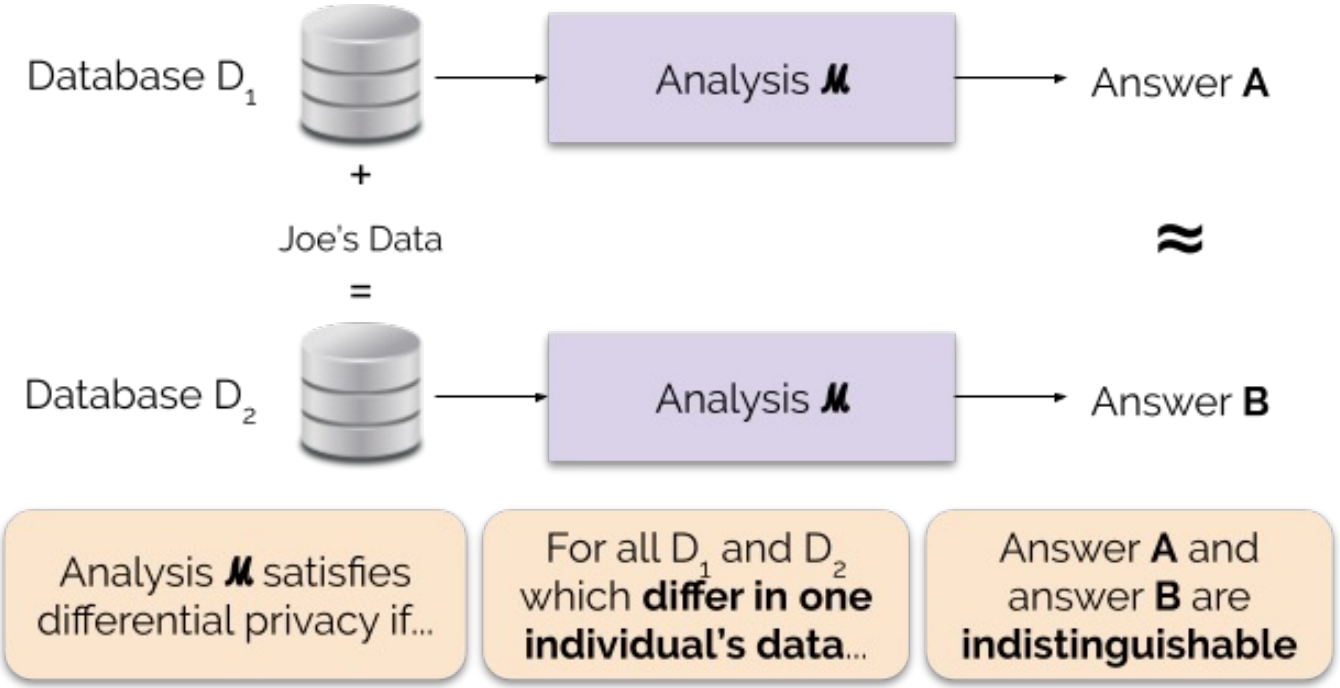
Information Service Providers and Data Brokers:

FHE empowers information service providers and data brokers to eliminate the need for large-scale data transfers by enabling secure computations on encrypted data, thus streamlining interactions with government and public entities.

Elimination of the Need for Large-Scale Sensitive Data Transfers

5. Differential Privacy

- **Differential Privacy** is a promising technique to mitigate *membership inference attacks*.
  - **Goal:** *Minimize differences between two databases/datasets* to preserve data privacy.



5. Differential Privacy

Since we want to preserve data privacy, why don't we just anonymize data?

The answer is "***Yes, we can! But it doesn't work***".

- **Netflix** attempted to anonymize the training data set for its recommender system by replacing user's name and randomize IDs. It turns out that *researchers can deanonymize users by linking the dataset with public IMdB review rating*.
- **Strava** got into hot water when it *released an anonymized aggregate heatmap of location data*, as it *inadvertently revealed the location of military bases*.

Every time we perform statistical analysis; we leak little information about the data.

5. Differential Privacy : Formal Definitions

Let's quantify the loss of privacy:

- **DP** helps quantify the privacy of *a mechanism M* that interacts with *data D* to produce output.
- In the case of ML, the *mechanism M* is *not just the final model*, it is the entire process of training a model on the *dataset D* and using that model's predictions.
- **Example: Training on a Cancer Dataset**
  - Says we are training a ML model on *a cancer dataset D*, and we ask it to predict if *a particular patient **Bob** has cancer*.
  - The trained model can predict with **55% confident**, as follows:

$$\Pr[M(D) = \text{“}Bob\ has\ cancer\text{”}] = 0.55$$

5. Differential Privacy : Formal Definitions

Example: Training on a Cancer Dataset

- If we then train the model with *the same dataset augmented with Bob's records*, it improves from **55%** to **57% confident**:

$$\Pr[M(D + Bob) = \text{“}Bob\ has\ cancer\text{”}] = \mathbf{0.57}$$

- Bob's ***privacy*** comes from *this plausible deniability* of him having cancer.
- What if it became that the confidence of "*Bob has cancer*" increases to **80%**?

$$\Pr[M(D + Bob) = \text{”}Bob\ has\ cancer\text{”}] = \mathbf{0.80}$$

- Just by adding Bob to the dataset, the outcome "*Bob has cancer*" has become **significantly more likely**, so *Bob does in fact have cancer*, and *the mechanism M* **leaked information about Bob**!

5. Differential Privacy : Formal Definitions

Example: Training on a Cancer Dataset

- We can **quantify** ***the privacy loss*** as the log of the ratio of the probabilities before and after Bob's record was added to the dataset: i.e.:

$$\log \frac{\Pr[M(D + Bob) = \text{“}Bob\ has\ cancer\text{”}]}{\Pr[M(D) = \text{”}Bob\ has\ cancer\text{”}]}$$

This definition of privacy loss comes from information theory. Let the events be defined as:

$$\boldsymbol{A} = [M(D) = \text{“}Bob\ has\ cancer\text{”}] \text{ and } \boldsymbol{B} = [M(D + Bob) = \text{“}Bob\ has\ cancer\text{”}]$$

Information associated to event $I(\boldsymbol{A}) = -\log \boldsymbol{A}$, likewise for $\boldsymbol{B}$.

The privacy loss is the difference in information between **two events**:

$$I(\boldsymbol{A}) - I(\boldsymbol{B}) = -\log\big(\boldsymbol{A} - (-\log \boldsymbol{B})\big) = \frac{\log \boldsymbol{B}}{\log \boldsymbol{A}}$$

5. Differential Privacy : Formal Definitions

Example: Training on a Cancer Dataset

- We can **quantify *the privacy loss*** as the log of the ratio of the probabilities before and after Bob's record was added to the dataset: i.e.:

$$\log \frac{\Pr[M(D + Bob) = \text{“}Bob\ has\ cancer\text{”}]}{\Pr[M(D) = \text{”}Bob\ has\ cancer\text{”}]}$$

- Let's put the confident values we have in the formula:

$$\log \frac{0.57}{0.55} = 0.0357$$

$$\log \frac{0.80}{0.55} = 0.357$$

- We can see that the second outcome has **10x privacy loss**.

5. Differential Privacy : Formal Definitions

- We can reformulate the definition of privacy loss here as: we look at *datasets* D and D' that differ in one data record (So, we add/remove a record). We consider **the privacy loss** on **a set of outcomes** S:

$$\log \frac{\Pr[M(D') \in S]}{\Pr[M(D) \in S]}$$

- Ideally, if we have **zero privacy loss**, the **probabilities of two events will be the same**.
- So, we must be a bit more flexible and *specify an upper-bound on our privacy loss*, which we denote with an ϵ called **a privacy budget**.

$$\log \frac{\Pr[M(D') \in S]}{\Pr[M(D) \in S]} \leq \epsilon$$

- We can balance **a trade-off** between *how much the mechanism learn* and *the privacy* by tuning ϵ.

5. Differential Privacy : Formal Definitions

ϵ-Differential Privacy: We can rearrange the privacy loss inequality to get our first mathematical definition of privacy.

$$\Pr[M(D) \in S] \leq \mathbf{e}^{\epsilon} \Pr[M(D') \in S]$$

So, if we can bound this privacy loss by ϵ for any *datasets* D and D' whatever the outcome we observe, we have a formal guarantee of privacy. We say *the mechanism* M is **ϵ-differentially private**.

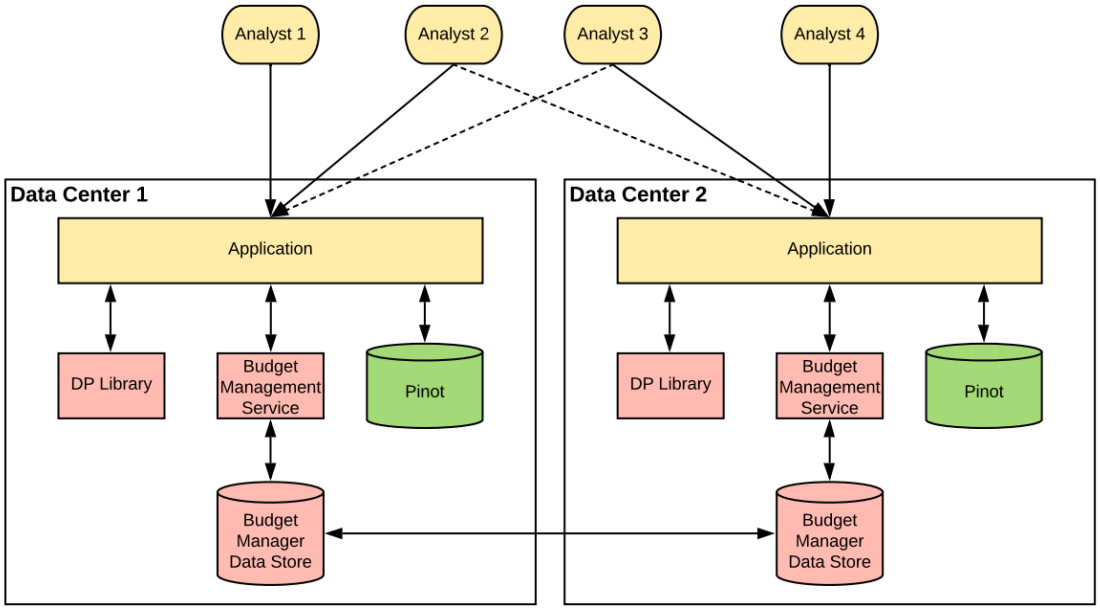
$(\epsilon - \delta)$-Differential Privacy: This bound is very ***tight***. In practice, we allow for ***a tiny failure probability*** δ (much less than the probability of a given individual).

$$\Pr[M(D) \in S] \leq \mathbf{e}^{\epsilon} \Pr[M(D') \in S] + \delta$$

We say *the mechanism* M is $(\epsilon - \delta)$-**differentially private**.

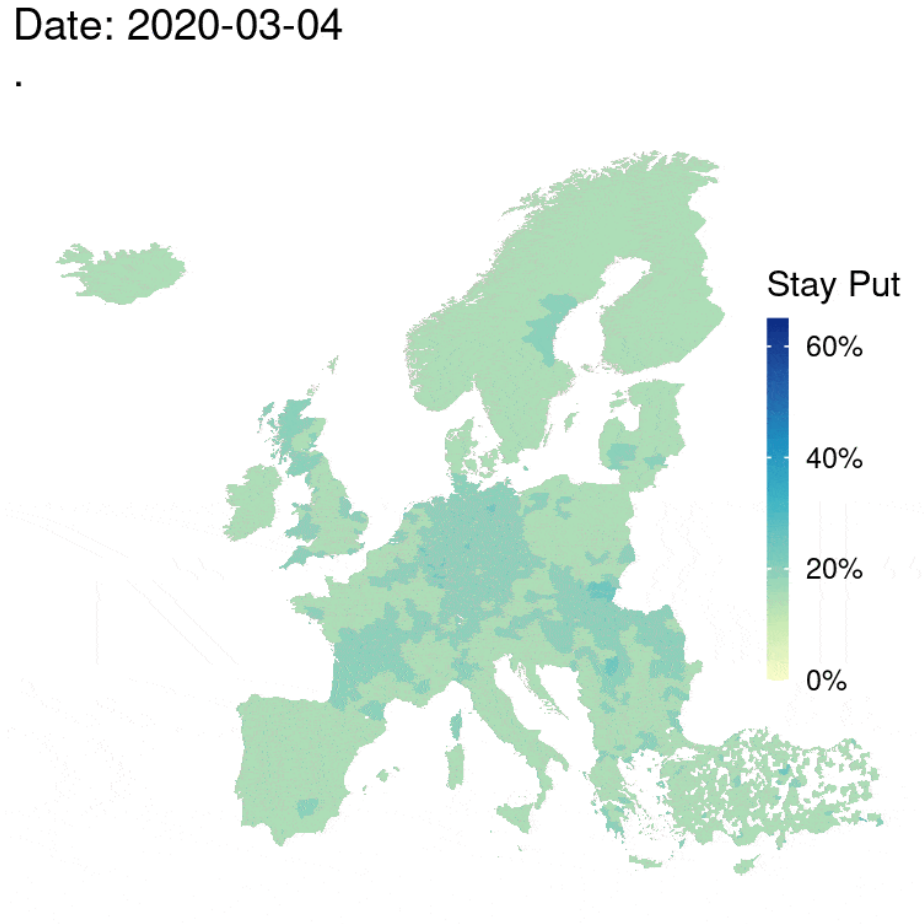
The next episode of this concept will be in SEC-401 Differential Privacy!

5. Differential Privacy : Use Cases



- The **Audience Engagements API** is the only interactive query system in this list. It allows marketers to get information about **LinkedIn** users engaging with their content. Each query returns $(\boldsymbol{\varepsilon}, \boldsymbol{\delta})$-DP with $\boldsymbol{\varepsilon} = 0.15$ and $\delta = 10^{-10}$, with a user as a privacy unit. Each analyst can send multiple queries, but a monthly cap limits how many: the total $(\boldsymbol{\varepsilon}, \boldsymbol{\delta})$ budget is $\varepsilon = 34.9$ and $\delta = 7 \times 10^{-9}$, with a privacy unit of user-month-analyst.

5. Differential Privacy : Use Cases



- The **Movement Range Maps** *quantify the changes in mobility of Facebook users during the COVID-19 pandemic*. There are **two** metrics: how much their users move during each day, and how many people are generally staying at home. Each metric uses a daily value $\varepsilon = 1$, so, the overall privacy budget is $\varepsilon = 2$ with user-day as a privacy unit.

More information about use cases of DP:
https://desfontain.es/blog/real-world-differential-privacy.html

Key Takeaways

- Modern AI systems are facing many security challenges as AI specific risks and threats.
- We have learned many kinds of **AI-specific risks and threats** in the form of **AI-specific attacks**, including:
  - *Data Poisoning, Model Inversion Attacks, Adversarial Examples, Model Stealing Attacks, Backdoor Attacks, Evasion Attacks, AI-Enhanced Social Engineering, Transfer Learning Attacks, and Membership Inference Attacks*
- We have also learned about preventions and mitigation techniques, including:
  - Transfer Learning (TL)
  - Federated Learning (FL)
  - Generative Adversarial Networks (GANs)
  - Homomorphic Encryption (HE)
  - Differential Privacy (DP)

End of the Lecture

Please do not hesitate to ask any questions to free your curiosity,

If you have any further questions after the class, please contact me via email (charnon@cmkl.ac.th).