ChatGPT

MASKCRYPT 联邦学习实现框架

以下代码框架基于 MASKCRYPT 论文 [1] [2] 中的设计思想，使用开源联邦学习框架 PLATO [3] 和 TenSEAL 同态加密库 [4]。代码结构如下：

- **main.py**：主程序，控制联邦训练流程。
- **model.py**：定义模型架构（如简单的神经网络）。
- **client.py**：MaskCrypt 客户端类，实现本地训练、梯度掩码选择和部分模型加密。
- **server.py**：MaskCrypt 服务器类，实现掩码共识、模型聚合和解密流程。
- **utils/**：存放工具模块：
- **mask_selection.py**：梯度引导掩码选择算法（算法2）。
- **mask_consensus.py**：掩码共识算法（算法1）。
- **encryption_utils.py**：TenSEAL 加密上下文和加密/解密辅助函数。

```
maskcrypt_fl/
├── main.py
├── model.py
├── server.py
├── client.py
└── utils/
    ├── mask_selection.py
    ├── mask_consensus.py
    └── encryption_utils.py
```

联邦学习架构 (基于 PLATO)

根据 MASKCRYPT 论文 [3]，我们基于 PLATO 框架实现联邦学习。PLATO 提供 Server/Client 的抽象接口，在此基础上定义 `MaskCryptServer` 和 `MaskCryptClient`：

```python
# server.py
import plato
class MaskCryptServer(plato.Server):
    def __init__(self, model, num_clients):
        super().__init__(model)
        self.num_clients = num_clients
        self.global_weights = model.get_weights()
        # 存储各轮聚合结果等

    def aggregate(self, updates):
        # 聚合逻辑（见下文模型聚合模块）
        pass

# client.py
import plato
class MaskCryptClient(plato.Client):
```

```
    def __init__(self, client_id, model, data):
        super().__init__(model, data)
        self.client_id = client_id
        # TenSEAL 上下文将在 utils 中生成并传入
        self.context = None

    def local_update(self, global_weights):
        # 将服务器模型加载到本地模型
        self.model.set_weights(global_weights)
        # 在本地数据上训练（例如一个或几个 epoch）
        grads, local_weights = self.model.train(self.data)
        # 计算掩码并加密部分权重（见下文掩码选择和加密）
        ...
```

PLATO 框架通过配置文件或代码将多个客户端并行启动，并自动处理模型广播和聚合，本实现中通过 `main.py` 手动模拟这一流程。下文各模块详细描述了 MaskCrypt 的核心逻辑。

梯度引导掩码选择 (Gradient-Guided Mask Selection)

根据 MASKCRYPT 论文，客户端在本地训练后使用梯度指导选择加密掩码（算法2） [5] 。具体地，客户端计算当前梯度 $g=\nabla L(w^k_t,D_k)$ 和模型更新差值 $\delta=w^k_{\text{exp},t}-w^k_t$，得到向量 $v=g\odot\delta$，然后选取 v 中最大的 ρN 个元素对应的索引作为掩码。

```
# utils/mask_selection.py
import numpy as np

def gradient_guided_mask_selection(exposed_weights, updated_weights, gradients, rho):
    """
    梯度引导掩码选择算法（算法2 [5] ）
    输入:
      exposed_weights: 客户端暴露的旧权重 w_k_exp,t
      updated_weights: 训练后本地更新的权重 w_k_t
      gradients: updated_weights 对应的梯度 g
      rho: 加密比例 ρ
    输出:
      mask_indices: 加密掩码索引列表 m_k
    """
    # 计算模型差值 δ = w_exp - w_local
    delta = exposed_weights - updated_weights  # np.ndarray
    # 计算元素级乘积 v = g ⊙ δ
    v = gradients * delta
    # 选取前 rho * N 个最大元素对应的索引
    N = len(v)
    top_k = int(rho * N)
    mask_indices = np.argsort(v)[::-1][:top_k]
    return set(mask_indices)
```

上述实现中，`mask_indices` 是加密比例 ρ 指定数量的模型参数索引。这样客户端得到的掩码将用于指定需要加密的权重部分 [5] 。

掩码共识机制 (Mask Consensus)

为了让所有客户端使用一致的加密掩码，MASKCRYPT 设计了掩码共识算法（算法1） [6] 。服务器收集所有客户端的掩码提议（集合 $m_1, m_2, \dots, m_K$）并运行如下步骤：先将所有提议合并、去重，然后选取前 $ρN$ 个索引作为最终掩码 $\bar m$ [6] 。

```
# utils/mask_consensus.py
def mask_consensus(mask_list, rho, N):
    """
    掩码共识算法（算法1 [6] ）
    输入:
      mask_list: 来自各客户端的掩码提议列表 [m1, m2, ..., mK]，其中每个 mk 是索引集合
      rho: 加密比例 ρ
      N: 模型总参数个数
    输出:
      final_mask: 最终掩码集合 \bar{m}
    """
    # 合并所有提议并去重
    combined = set().union(*mask_list)
    # 简化实现：对所有索引排序并截取前 rho*N 个
    sorted_idxs = sorted(combined)
    top_k = int(rho * N)
    final_mask = set(sorted_idxs[:top_k])
    return final_mask
```

在上述代码中，我们简单地合并并去重所有提议，然后选择前 $ρN$ 个索引作为最终掩码 `final_mask` [6] 。这实现了论文中“交错合并提议、去重并截取前 $ρN$ 元素”的思路。

选择性同态加密 (Selective Homomorphic Encryption)

论文中采用 TenSEAL 库的 CKKS 同态加密方案 [4] 对最终掩码指定的权重进行加密，其他权重保持明文。首先，初始化加密上下文：

```
# utils/encryption_utils.py
import tenseal as ts

def create_tenseal_context():
    """
    创建 TenSEAL CKKS 同态加密上下文（参考论文实验配置 [4] ）
    """
    context = ts.context(
        ts.SCHEME_TYPE.CKKS,
        poly_modulus_degree=8192,
        coeff_mod_bit_sizes=[60, 40, 40, 60]
    )
    context.generate_galois_keys()
```

```
    context.global_scale = 2**40
    return context
```

每个客户端初始化自己的 `context`（含私钥）并公开公钥给服务器和其他客户端。**模型加密：** 客户端根据共识得到的最终掩码 $\bar m$ 将掩码元素划分为 K 个子集，然后对每个子集使用对应客户端的公钥加密 [2]。下面是一个简化的加密示例：

```
# utils/encryption_utils.py
import numpy as np

def selective_encrypt(weights, mask, public_contexts):
    """
    部分权重加密函数
    输入:
      weights: 模型权重列表或数组
      mask: 最终加密掩码索引集合 \bar{m}
      public_contexts: 公钥上下文列表，长度为 K（每个上下文对应一个客户端公钥）
    输出:
      encrypted_parts: 字典，键为客户端索引 j，值为该子集加密后的 {idx: CKKS向量}
      plain_parts: 字典 {idx: value}，存储未加密的权重
    """
    K = len(public_contexts)
    # 将掩码划分为 K 个子集
    mask_list = list(mask)
    subsets = np.array_split(mask_list, K)
    encrypted_parts = {}
    for j, subset in enumerate(subsets):
        ctx = public_contexts[j]  # 客户端 j 的公钥上下文
        encrypted_parts[j] = {}
        for idx in subset:
            # 使用 CKKS 加密单个权重（以长度1向量形式）
            encrypted_parts[j][idx] = ts.ckks_vector(ctx, [weights[idx]])
    # 保留未加密的权重
    plain_parts = {i: weights[i] for i in range(len(weights)) if i not in mask}
    return encrypted_parts, plain_parts
```

如上所示，`encrypted_parts[j][idx]` 表示第 j 个公钥加密的索引 `idx` 处的权重（CKKS 向量）。该过程实现了论文所述“将最终掩码划分成 K 个子集，使用不同客户端公钥进行加密” [2]。

模型聚合 (Aggregation)

服务器接收来自各客户端的更新，其中包含加密部分和明文部分 [7]。根据 MASKCRYPT 设计，服务器对加密部分执行同态加法，对明文部分执行普通加法 [8]，分别得到加密的聚合结果和明文的聚合结果：

```
# server.py 聚合函数示例
def aggregate_updates(encrypted_list, plain_list):
    """
    输入:
```

```
    encrypted_list: 列表，每个元素为一个客户端提交的 encrypted_parts 字典
    plain_list: 列表，每个元素为一个客户端提交的 plain_parts 字典
  输出:
    agg_encrypted: 字典 {j: CKKS向量}，对每个客户端子集加密权重同态求和结果
    agg_plain: 字典 {idx: value}，明文权重的累加结果
  """
  # 同态聚合加密部分
  agg_encrypted = {}
  for j in encrypted_list[0].keys():
      # 将所有客户端在子集 j 上的加密权重向量累加
      sum_vec = None
      for enc_parts in encrypted_list:
          if sum_vec is None:
              sum_vec = enc_parts[j][list(enc_parts[j].keys())[0]].copy()
              # 复制第一个 CKKS 向量以开始累加
              for idx, vec in enc_parts[j].items():
                  if idx != list(enc_parts[j].keys())[0]:
                      sum_vec += vec
          else:
              for idx, vec in enc_parts[j].items():
                  sum_vec += vec
      agg_encrypted[j] = sum_vec

  # 普通加法聚合明文部分
  agg_plain = {}
  for plain_parts in plain_list:
      for idx, val in plain_parts.items():
          agg_plain[idx] = agg_plain.get(idx, 0) + val

  return agg_encrypted, agg_plain
```

上述代码中，对加密部分我们使用 CKKS 向量的加法（`sum_vec += vec`）实现同态累加；对明文部分则普通累加。最终得到的 `agg_encrypted` 是加密聚合结果（每个子集 j 对应一个 CKKS 加密向量），`agg_plain` 是明文累加结果 [8]。

解密 (Decryption)

聚合完成后，服务器将每个加密聚合结果发送给相应客户端 j 进行解密 [9]。客户端使用其私钥对收到的 CKKS 向量解密，然后返回明文结果。服务器最后将这些解密后的值合并回全局模型。示例解密代码：

```
# utils/encryption_utils.py
def decrypt_aggregated(agg_encrypted, secret_contexts):
  """
  解密聚合后的加密权重
  输入:
    agg_encrypted: 字典 {j: CKKS向量}，来自服务器
    secret_contexts: 客户端私钥上下文列表，对应每个 j
  输出:
    dec_parts: 字典 {j: {idx: value}}，解密后的明文结果（每个子集 j 对应的权重值）
```

```
    """
    dec_parts = {}
    for j, enc_vec in agg_encrypted.items():
        ctx = secret_contexts[j]
        # 使用 TenSEAL CKKS 向量的 decrypt() 获得明文列表
        dec_vals = enc_vec.decrypt()
        # 假设每个向量只加密了一个权重，取第一个值
        dec_parts[j] = dec_vals[0]
    return dec_parts
```

解密后，服务器将得到每个子集的明文聚合结果 `dec_parts[j]`，并与之前的 `agg_plain` 合并。最终全局模型的各参数为：如果参数在掩码中，则使用对应客户端返回的解密值；否则使用明文聚合结果。这使得服务器拥有完整的聚合模型权重 [10] 。

训练主循环

主循环整合上述各部分模块来完成联邦训练流程 [11] [10] ：

```
# main.py
from server import MaskCryptServer
from client import MaskCryptClient
from utils.mask_selection import gradient_guided_mask_selection
from utils.mask_consensus import mask_consensus
from utils.encryption_utils import create_tenseal_context, selective_encrypt, decrypt_aggregated

# 初始化服务器和客户端
num_clients = K  # 假设有 K 个客户端
model = initialize_model()
server = MaskCryptServer(model, num_clients)
clients = []
for k in range(num_clients):
    client = MaskCryptClient(client_id=k, model=model, data=load_data(k))
    # 每个客户端创建 TenSEAL 上下文（含私钥），并收集公钥
    ctx = create_tenseal_context()
    client.context = ctx
    clients.append(client)
# 收集所有客户端的公钥（context）
public_contexts = [c.context for c in clients]

for t in range(T):  # 进行 T 轮迭代
    # 1. 服务器广播当前全局模型
    global_weights = server.global_weights

    # 2. 客户端本地训练并生成更新
    updates = []
    masks = []
    for client in clients:
        # 模拟客户端收到模型并训练
        exposed_weights = global_weights.copy()  # 暴露权重（可初始即为全局模型）
```

```
        grads, local_weights = client.model.train(client.data)
        # 掩码选择（算法2 5）
        mask = gradient_guided_mask_selection(exposed_weights, local_weights, grads, rho)
        masks.append(mask)
        # 部分加密
        encrypted_parts, plain_parts = selective_encrypt(local_weights, mask, public_contexts)
        updates.append((encrypted_parts, plain_parts))

    # 3. 掩码共识（算法1 6）
    final_mask = mask_consensus(masks, rho, len(global_weights))

    # 4. 聚合更新
    enc_list = [u[0] for u in updates]
    plain_list = [u[1] for u in updates]
    agg_enc, agg_plain = server.aggregate(enc_list, plain_list)

    # 5. 解密聚合结果（服务器发送给客户端 j）
    secret_contexts = [c.context for c in clients]
    dec_parts = decrypt_aggregated(agg_enc, secret_contexts)

    # 6. 更新全局模型权重
    new_weights = global_weights.copy()
    for idx in range(len(global_weights)):
        if idx in final_mask:
            # 参数在掩码中，使用解密结果（假设除以 num_clients 得到平均）
            # 这里简单取返回值再除以 num_clients
            # 注意：实际可将客户端返回值相加再平均
            new_weights[idx] = dec_parts[who_decrypted(idx)] / num_clients
        else:
            # 参数未加密，使用明文聚合再平均
            new_weights[idx] = agg_plain.get(idx, 0) / num_clients
    server.global_weights = new_weights
```

上述流程中：服务器初始化全局模型并广播给客户端 11；客户端本地训练并生成含有掩码的模型更新；服务器执行掩码共识、同态聚合和解密 8 10，最终更新全局模型。每个模块的实现逻辑对应了论文中描述的 MASKCRYPT 机制 1 2。

参考文献: 以上实现均参考 MASKCRYPT 论文 1 2 中算法设计，使用 PLATO 框架 3 和 TenSEAL（CKKS）库 4 进行仿真验证。

---

1 2 3 4 5 6 7 8 9 10 11 **iqua.ece.toronto.edu**
https://iqua.ece.toronto.edu/papers/chenghao-tdsc24.pdf