Move to AGI

Decentralized HuggingFace (Nostr-based decentralized AGI model storage protocol + AGI Git version control + SAAS private model Tokenomics)

AGI as a Service, GIT as a Service

Babel Open-Source Community

2024/3/31

PROBLEMS AND SOLUTIONS

BABEL

Centralization Risks

A few powerful entities **monopolized** AI's future, limiting diversity and innovation.

→

Decentralized AGI Protocol

AGI Nostr

Blockchain technology **decentralizes** storage (data) and model training (computation)

Privacy Concerns

Inadequate **protections** and **compensation** for user data in AI development and application.

→

Personalization & Privacy

AGI Nostr+Token SAAS

Utilizes a foundation+personal model structure, differentiating between shared knowledge and individual, encrypted, **self-owned** personal models for trustless privacy.

Accessibility and Economic Barriers

High **entry costs** on capital prevent widespread participation in AGI's research & development – and applications.

→

Fair Contribution Incentive

Tokenomics

Implements a token economy with **Proof of Work of Compression (PoWoC)** for quantifiable and computable contributions, reducing AGI contributors' entry barriers and encouraging wider participation.

Blockchain Usability

Existing blockchain languages are **hard to learn** and **harder to use**.

→

A+B Integration

AGI +Move SmartContract

Integrating Large Language Model (LLM) **AGI with Blockchain** naturally solves usability issues, making blockchain applications **programmable with natural language**, fostering a sustainable, profitable, and universally fair open-source ecosystem.

Sustainability of Open-Source AGI

Challenges in maintaining an open-source model that is both inclusive and **economically viable**.

→

Sustainable Tokenomics

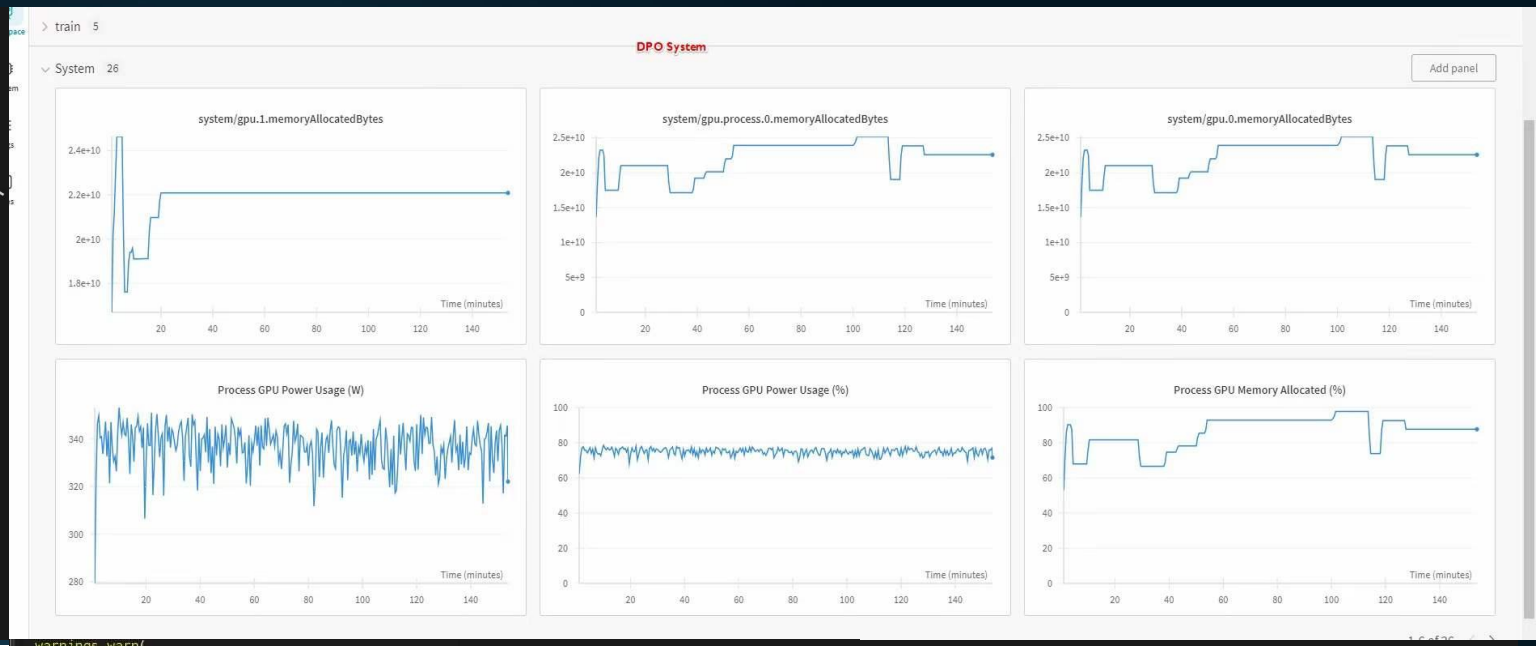
AGI Git+Token SAAS

Users (demand) transfer **tokens** to the blockchain for A+B integration; the blockchain transfers **tokens** to contributors (supply) for every incremental improvement of AGI.

AGI Traning & Storage（Size：N–> ≈√N）

AGI

- AGI模型数据由4部分数据组成（模型代码、模型参数、训练数据=验证数据），每个部分独立迭代升级。除了模型代码外另外几个都有巨大的数据量，所以要做分片（sharding）储存。
- 每部数据（模型参数、训练数据、验证数据）经过纠错码处理后拆分成k份，分布冗余储存在n个中继器里。从概率来说，用户只要从所有n个中继器里随机找出m个，就能完全还原完整的数据。(n>m>k)
  - 如果是用户的私人AGI模型，先对它进行加密后再经过纠错码处理后拆分成k份，分布冗余储存在n个中继器里。
- 数据主要分为两类：全人类共有的Foundation model和私人Domain specific model （DSM）。
  - Foundation model （巨大）分片储存在全网所有中继器里
  - DSM（较小）分片或不分片储存在一部分中继器里
- 私人的domain specific model之所以较小因为它指定某一版/当前版（version）的Foundation model作为master，然后私人model自己作为其小分岔（fork），储存数据时只储存私人model与Foundation model的$\Delta = dsm - fm$，所以得到一个可压缩矩阵。 [1]



GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection

Jiawei Zhao [1] Zhenyu Zhang [3] Beidi Chen [2 4] Zhangyang Wang [3] Anima Anandkumar [* 1] Yuandong Tian [* 2]

Abstract

Training Large Language Models (LLMs) presents significant memory challenges, predominantly due to the growing size of weights and optimizer states. Common memory-reduction approaches, such as low-rank adaptation (LoRA), add a trainable low-rank matrix to the frozen pre-trained weight in each layer, reducing trainable parameters and optimizer states. However, such approaches typically underperform training with full-rank weights in both pre-training and fine-tuning stages since they limit the parameter search to a low-rank subspace and alter the training dynamics, and further, may require full-rank warm start. In this work, we propose Gradient Low-Rank Projection (**GaLore**), a training strategy that allows *full-parameter* learning but is more *memory-efficient* than common low-rank adaptation methods such as LoRA. Our approach reduces memory usage by up to 65.5% in optimizer

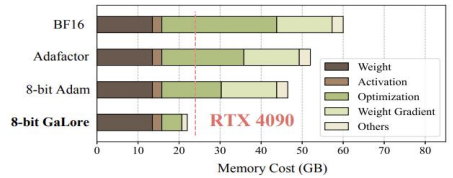


Figure 1: Memory consumption of pre-training a LLaMA 7B model with a token batch size of 256 on a single device, without activation checkpointing and memory offloading. Details refer to Section 5.5.

Algorithm 1: GaLore, PyTorch-like

```
for weight in model.parameters():
   grad = weight.grad
   # original space -> compact space
   lor_grad = project(grad)
   # update by Adam, Adafactor, etc.
   lor_update = update(lor_grad)
   # compact space -> original space
   update = project_back(lor_update)
   weight.data += update
```

```
  warnings.warn(
###Question: give me a C++ code about quick sort.
###Answer: Here is a C++ code for quick sort:

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivot = arr[low];
        int i = low + 1;
        int j = high;

        while (i < j) {
            while (arr[i] < pivot) {
                i++;
            }
            while (arr[j] > pivot) {
                j--;
            }
            if (i < j) {
                swap(arr[i], arr[j]);
                i++;
                j--;
            }
        }
        swap(arr[low], arr[j]);
        quickSort(arr, low, j - 1);
        quickSort(arr, j + 1, high);
    }
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    quickSort(arr, 0, 9);
    for (int i = 0; i < 10; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```
```

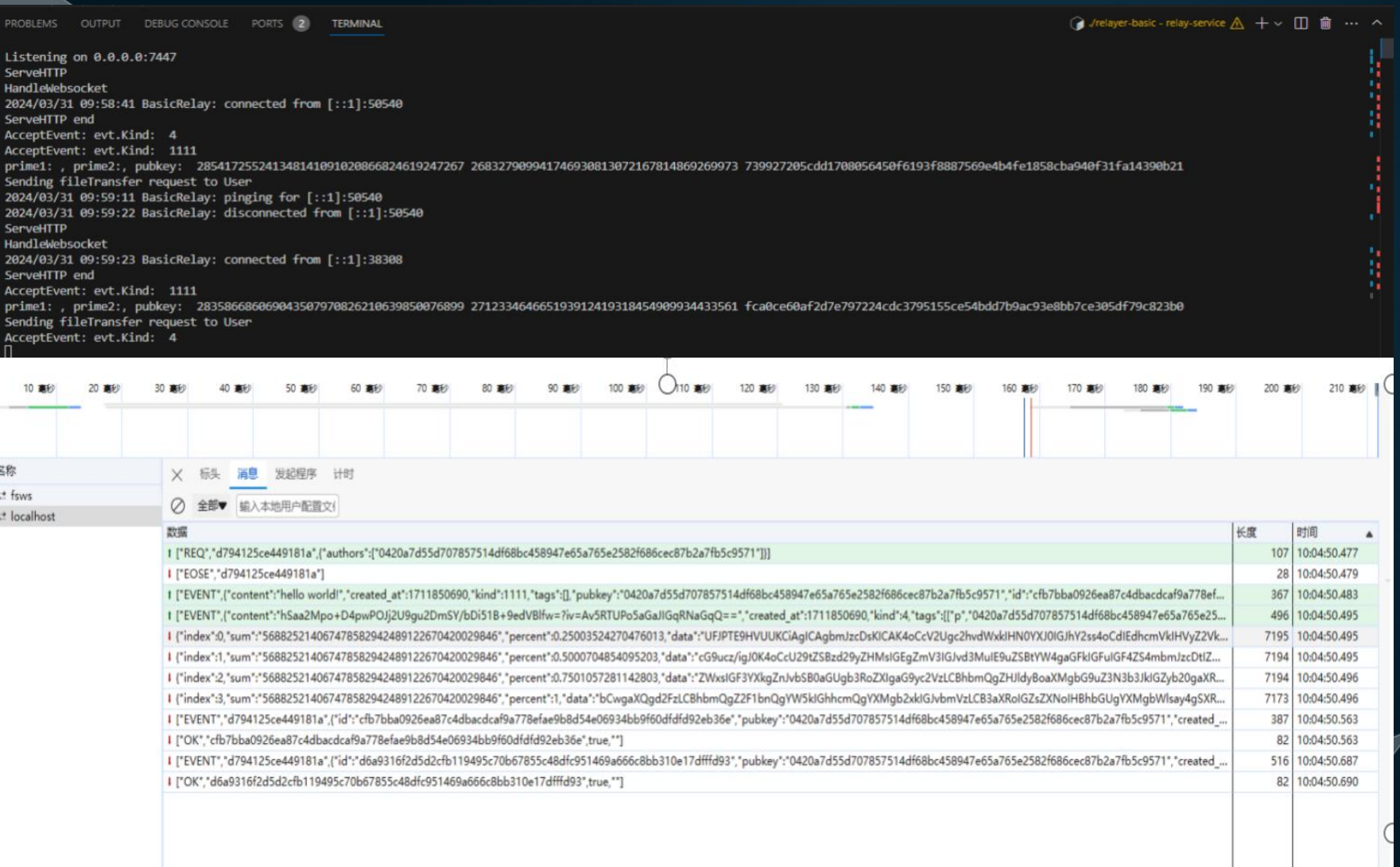
Nostr

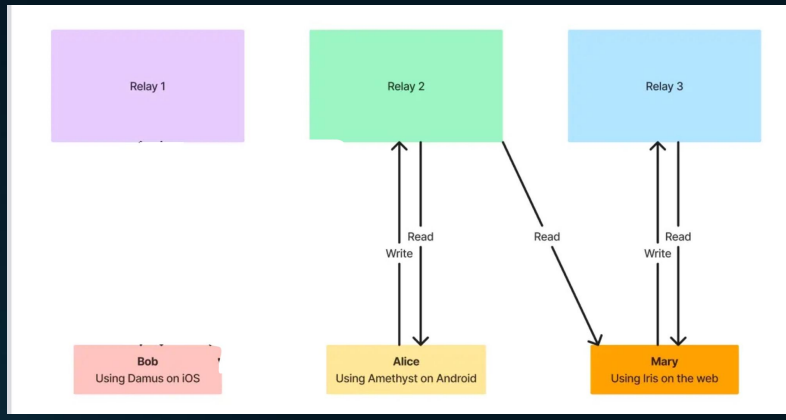
- Nostr 有两个组件：客户端和中继器。每个用户运行一个客户端，任何人都可以运行中继器。
- 每个用户都由公钥标识，每数据都有签名，每个客户端都会验证这些签名。
- 在启动时，客户端从它知道的所有m个中继器。
- 客户端从他们选择并信任的m个中继器中查询获取（fetch）它所关注的数据（Foundation model and/or domain specific model）
- 并将根据Foundation model数据fine-tune后的私人DSM的数据发布到他们选择并的其他中继器。
- 中继器不与另一个中继器通信，仅直接与用户通信。

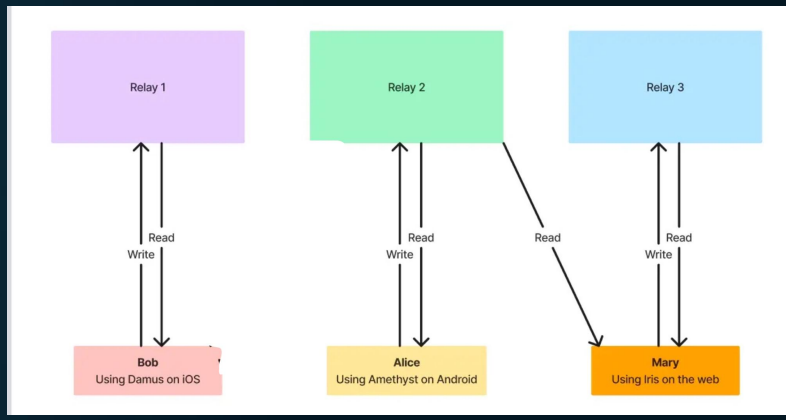
Nostr as AGI GIT

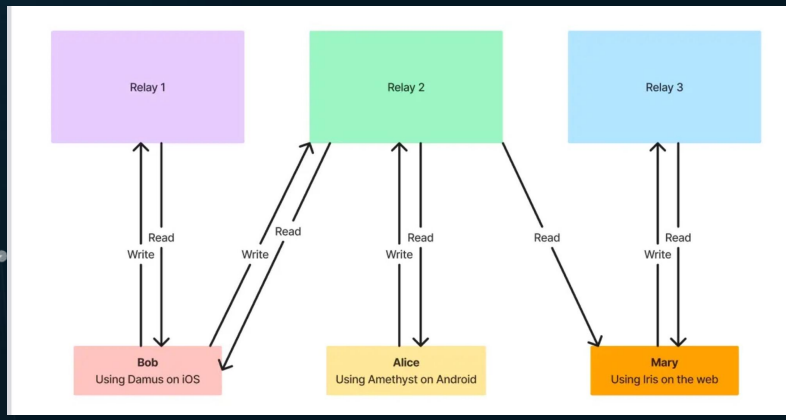
AGI Git

1.搭建nostr中继，并且连接postgresql数据库，前端
2.根据前端事件类型执行agigitPull命令
3.通过websocket的连接文件分块传输到客户端页面









AGI Traning & Storage

Ecosystem of AGI trainers + blockchain miners + storage hosts

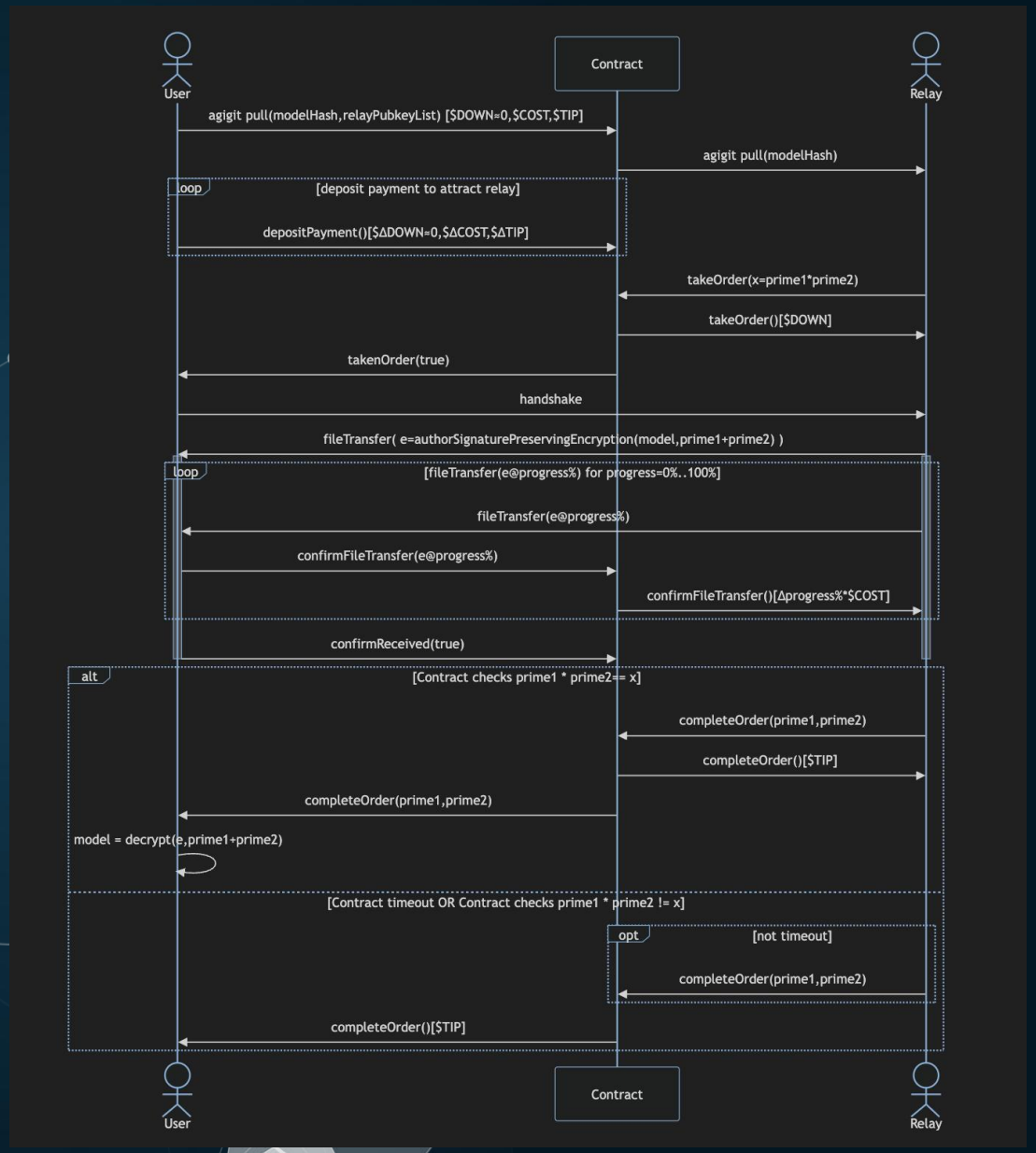
Tokenomics

agigit commands

Shell

Copy

```
agigit relay add --fetch $relayPK #PublicKey
agigit relay add --push $relayPK
agigit relay add $relayPK
agigit relay remove --fetch $relayPK
agigit relay remove --push $relayPK
agigit relay remove $relayPK
agigit fetch
agigit log
agigit pull
agigit pull $commitHash
agigit checkout $commitHash
agigit checkout master
agigit commit -m "commit message"
agigit push
agigit merge $commitHashA $commitHashB
agigit rebase
agigit rebase $nonHeadCommitHash
agigit log
```



Move to AGI

Git as a Service
AGI as a Service

1. 用户发布订单

```
public entry fun pull(
    account: signer,
    model_hash: u64,
    relay_pubkey_list: vector<u64>,
    down: u64,
    cost: u64,
    tip: u64
) acquires OrderList, ModuleData {

    let amount1 :u64 = down;
    let amount2 :u64 = cost;
    let amount3 :u64 = tip;
    let sum :u64 = down + cost + tip;

    let module_data :&mut ModuleData = borrow_global_mut<ModuleData>( addr: @nag);
    let resource_signer :signer = account::create_signer_with_capability( capability: &module_data.resource_signer_cap);
    coin::transfer<AptosCoin>( from: &account, to: signer::address_of( s: &resource_signer), sum);
```

2. 用户追加费用

```
public entry fun raise_payment(
    account: signer,
    order_id: u64,
    down: u64,
    cost: u64,
    tip: u64
) acquires OrderList, ModuleData {
    assert!(exists<OrderList>( addr: signer::address_of( s: &account)), 0);
    let order_list :&vector<Order> = &borrow_global<OrderList>( addr: signer::address_of( s: &account)).order_list;
    let length :u64 = vector::length<Order>(order_list);
    assert!(length >= order_id, 1);

    let order_list_mut :&mut vector<Order> = &mut borrow_global_mut<OrderList>( addr: signer::address_of( s: &account)).order_list;
    let order :&mut Order = vector::borrow_mut<Order>(order_list_mut, order_id);
    assert!(order.status == 0, 2);

    let amount1 :u64 = down;
    let amount2 :u64 = cost;
    let amount3 :u64 = tip;

    let sum :u64 = amount1 + amount2 + amount3;
    let module_data :&mut ModuleData = borrow_global_mut<ModuleData>( addr: @nag);
    let resource_signer :signer = account::create_signer_with_capability( capability: &module_data.resource_signer_cap);
    coin::transfer<AptosCoin>( from: &account, to: signer::address_of( s: &resource_signer), sum);

    order.down = order.down + amount1;
    order.cost = order.cost + amount2;
```

3. relay接收订单

```
public entry fun take_order(
    account: signer,
    order_id: u64,
    order_address: address,
    relay_pubkey: u64,
    x: u64,
)  acquires OrderList, ModuleData {
    assert!(exists<OrderList>(order_address), 0);

    let order_list_mut :&mut vector<Order> = &mut borrow_global_mut<OrderList>(order_address).order_list;
    let order :&mut Order = vector::borrow_mut<Order>(order_list_mut, order_id);
    assert!(order.status == 0, 2);
    assert!(vector::contains( v: &order.relay_pubkey_list, e: &relay_pubkey), 3);
    order.status = 1;
    order.taker = signer::address_of( s: &account);
    order.product = x;

    let module_data :&mut ModuleData = borrow_global_mut<ModuleData>( addr: @nag);
    let resource_signer :signer = account::create_signer_with_capability( capability: &module_data.resource_signer_cap);
    let down_amount :u64 = order.down;

    event::emit(TakeOrder{
        relay_address: order.taker,
        order_id: order.order_id,
        order_address: order_address,
    });

    coin::transfer<AptosCoin>( from: &resource_signer, to: signer::address_of( s: &account), down_amount);
```

4. 用户确认已接收数据

```
public entry fun confirm_received(account: signer, order_id: u64) acquires OrderList {
    assert!(exists<OrderList>( addr: signer::address_of( s: &account)), 0);
    let order_list :&vector<Order> = &borrow_global<OrderList>( addr: signer::address_of( s: &account)).order_list;
    let length :u64 = vector::length<Order>(order_list);
    assert!(length > order_id, 1);

    let order_list_mut :&mut vector<Order> = &mut borrow_global_mut<OrderList>( addr: signer::address_of( s: &account)).order_list;
    let order :&mut Order = vector::borrow_mut<Order>(order_list_mut, order_id);
    assert!(order.status == 1, 2);

    order.status = 2;
}
```

5. 中继确认订单完成

```
public entry fun complete_order(
    account: signer,
    order_id: u64,
    order_address: address,
    prime1: u64,
    prime2: u64
) acquires OrderList, ModuleData {
    assert!(exists<OrderList>(order_address), 0);
    let order_list :&vector<Order> = &borrow_global<OrderList>(order_address).order_list;
    let length :u64 = vector::length<Order>(order_list);
    assert!(length > order_id, 1);

    let order_list_mut :&mut vector<Order> = &mut borrow_global_mut<OrderList>(order_address).order_list;
    let order :&mut Order = vector::borrow_mut<Order>(order_list_mut, order_id);
    assert!(order.status == 2, 2);

    let x :u64 = prime1 * prime2;

    if (x == order.product) {
        order.status = 3;
        event::emit(CompleteOrder {
            order_address: order_address,
            order_id: order_id,
            prime1: prime1,
            prime2: prime2,
        });

        let module_data :&mut ModuleData = borrow_global_mut<ModuleData>( addr: @nag);
        let resource_signer :signer = account::create_signer_with_capability( capability: &module_data.resource_signer_cap);
        let amount :u64 = order.cost + order.tip;
```

THANK YOU