Distribute.AI Whitepaper

Version 1.0

1 Introduction

Over the past decade, machine learning has experienced a rapid growth in popularity, permeating various aspects of daily life. As the demand for compute resources for machine learning tasks continues to surge, existing centralized inference mechanisms are being pushed to their limits. Simultaneously, supply chain shortages have led to year-long back orders on cutting edge hardware while hundreds of thousands of users already have mid-tier to high-end excess compute sitting idle. By connecting this unused capacity via a distributed network, a higher global compute efficiency can be achieved. This approach not only ensures cost-efficiency but also delivers reliability and performance to a broader spectrum of users. The Distribute.ai machine learning inference platform offers distributed Machine Learning as a Service (MLaaS) capabilities. The platform enables individuals to securely execute machine learning inference tasks with an expanding repertoire of popular models such as Llama [1] and Stable Diffusion XL [2], among others.

2 WebGPU-based Machine Learning

Despite the clear benefits of a distributed AI inference platform, the implementation of such a network requires a standardized system for all potential compute nodes to execute. Existing solutions have levered Docker as a means of deploying work units to service providers, offering decentralized physical infrastructure networks (DePIN), which presents its own issues. Despite the flexibility that this approach offers, it comes at the cost of ease of setup and deployment as well opening up additional attack vectors via running unverified code. By focusing only on ML inference, rather than generic compute, a custom platform can be implemented in the browser by leveraging targeted APIs.

Utilizing existing WebGPU APIs for ML represents a significant advancement for large scale, distributed, edge compute by exposing low-level graphics and general purpose computation [3]. Via these APIs, web applications are able to harness the full power of modern GPUs for accelerated computing tasks, which includes machine learning inference. This advancement allows for the flexibility to run larger, previously desktop-bound, machine learning model in an existing, proven, browser environment. Moreover, WebGPU offers cross-platform support for hardware acceleration, allowing for consistent performance and implementation across a variety of devices. Thus, WebGPU is well-poised as a tool for browser-based MLaaS for making inference accessible for a wide range of applications.

The Distribute.ai platform leverages WebGPU technology to enable efficient and scalable distributed machine learning within a web browser environment. Providers are able to exchange access to their WebGPU compute for Distribute.ai token. Specifically, service providers run inference tasks with WebGPU on behalf of users in exchange for token. The emergence of browser-based machine learning signifies a substantial leap in the access of AI technologies, offering low-latency and flexible processing directly within existing web applications.

3 Task Allocation

Inference tasks can be allocated to service providers in exchange for tokens. Specifically, this model involves users exchanging tokenized tasks or requests in the form of digital assets, which can serve as a tradable unit of value for the execution of a specific machine learning model by service providers. Machine learning service providers can participate in the network by offering their computational resources to run inference on an input, and they are rewarded with tokens in exchange.

A significant challenge in designing a task-allocation system is that determining the optimal task allocation solution is NP-Hard [4]. We implement the differential privacy-based combinatorial double auction algorithm proposed by Zhai et al. between users and service providers, and trades are executed when the bid price of a user matches the seller's ask price [5].

4 Combinatorial Double Auction Algorithm

Consider a vector of service providers $\boldsymbol{R} = \langle R_1, R_2, ..., R_n \rangle$ and inference tasks $\boldsymbol{T} = \langle T_1, T_2, ..., T_m \rangle$. Additionally, each inference task should be represented as $T_i = \langle C_{i,1}t_1, C_{i,2}t_2, ...C_{i,k}t_k \rangle$ where $C_{i,j}$ represents the demand for each of the k inference task types (e.g. each of the machine learning model types). Let v_i represent the adjusted valuation of computing the sub-task T_i, which reflects factors imposed by the demand such as the slashing mechanism discussed in Section 6. Let $\boldsymbol{S}$ be the set $S_j = \left(\boldsymbol{s_j}, w_j, c_j\right)$ where $\boldsymbol{s_j}$ represents the j-th resource (e.g. CPU processing power). w_j represents the maximum number of units of service that device j can provide for each resource type $s_{j,i}$ and c_j represents the true adjusted valuation of the i-th device's cost.

We can construct a matrix $A \in M_{\{m \times n\}}(\mathbb{1})$ such that

$$A_{i,j} = \begin{cases} 1 \ R_j \ \text{ accepts the } \ i \text{ -th task} \\ 0 \ \text{else} \end{cases}$$

Indeed, $\sum_{k=1}^{m} A_{k,j} \leq 1, \ \text{for} \ \ 1 \leq j \leq n$ since a model can strictly accept only one model inference. Let the additional cost function be defined by $e_i = t_i \log_\alpha(t_i) + \beta$ where $\alpha, \beta \in \mathbb{Z}^+$. Then, a utility function for users can be defined as

$$\text{utility}_i^u = \begin{cases} v_i - (\text{trade price}_i + e_i) & \text{if the } \ i \text{ -th task wins} \\ 0 & \text{else} \end{cases}$$

Similarly, the utility function for service providers can be defined as

$$\text{utility}_j^p = \begin{cases} \text{trade price}_j - \sum_{i=1}^{n} A_{i,j} & \text{if the } \ j \text{ -th task wins} \\ 0 & \text{else} \end{cases}$$

We must optimize the function $\max\left(\sum_{i=1}^{m} \text{utility}_i^u - \sum_{j=1}^{n} \text{utility}_j^p\right)$ such that:

$$\sum_{i=1}^{m} A_{i,j} \leq w_j \, \text{For each user device}$$

$$A_{i,j} \in [0,1], \forall i \in [1,m], j \in [1,n]$$

If the provider fails to complete the task within the alloted time, the task will be inserted back into the auction with a higher demand v_i.

4.1 Pricing Model

We introduce a pricing model as a fair-enough price-determining mechanism to motivate providers and users to participate. We adopt the same pricing model as Zhai et. al. (2018) which considers a fixed allocation matrix. Then, the winning resources of each user will be as follows:

$$\text{resource}_i = \sum A_{i,j} \times \sum s_j, 1 \leq i \leq m, 1 \leq j \leq n$$

The following two equations are the per unit price for tasks and providers accordingly.

$$\text{price (task per unit)}_i = \frac{v_i}{\sum x_{i,j}{}^* \sum s_j}, 1 \leq i \leq m, 1 \leq j \leq n$$

$$\text{price (provider per unit)}_j = \frac{c_j}{\sum s_j}, 1 \leq j \leq n$$

4.2 Algorithm

We present a polynomial time combinatorial algorithm for Equation 1, which is NP-Hard. Following Zhai et. al. (2018) and Jiang et. al. (2021), our scheduling mechanism invokes a bid density (bd) mechanism to push bids for efficient allocations. Let M_i represent the compute power of R_i, which will be determined by benchmarking. Let $\mathbf{\Psi}$ be a constant representing the ability for a service provider to complete a task determined by an algorithm that considers GPU compute power, CPU compute power, etc. and let M_i, bd_i^{task}, and $bd_i^{provider}$ be defined as:

$$bd_i^{task} = \sqrt{\sum_{j=1}^{k} C_{i,j}^2 + v_i^2}$$

$$bd_i^{provider} = \sqrt{\sum_{j=1}^{k} 1/s_{i,j}^2 + \sum_{j=1} \Psi_{i,j}^j}$$

Setting the bidding density for service providers in this manner heavily incentivize providers that are able to run WebGPU inference on relatively large models such as Llama and SDXL.

The main combinatorial double auction resource allocation algorithm is presented below. As mentioned earlier, this auction mechanism is utilized to determine fair prices for users and service providers. An important qualification is that only users are able to influence the auction by increasing v_i through bidding more token, which increases the bidding density. However, providers themselves are not able to set the price of their own bid. Instead, it is computed within the platform and a fair price is assigned to the provider.

PRICES(N_{tasks}, $N_{\text{providers}}$, $B_{\text{providers}}$, B_{tasks}):

Users and service providers send their bids to the auctioneer.
Compute a sorted list of bids of tasks and providers
Normalize components of $\boldsymbol{\Psi}$ and normalize $\boldsymbol{s} \in M(\mathbb{R}), C$ and v to be in the range (0, 1)
$G \leftarrow \langle bd_i^{task} \rangle$ sorted in non-increasing order $H \leftarrow \langle bd_i^{provider} \rangle$ sorted in non-increasing order
$i = j \leftarrow 1$
Flag Price:
$k \leftarrow 1$
$X \in M_{m,n}(\mathbb{0})$ // Allocation Matrix
$P \in M_{m,n}(\mathbb{0})$ // Price Matrix
if $(C = 0) \vee (v_i < q_i)$:
    $k \mathrel{+}= 1$
$X[i, j] = q_i$
$P[i, j] = \big(\Pr(\text{provider per unit})_j + \Pr(\text{task per unit})_i\big)/2$
if $\neg\big(\Pr(\text{provider per unit})_j \leq P[i, j] \leq \Pr(\text{task per unit})_i\big)$:
    $P[i, j] = \Pr(\text{provider per unit})_j$
$v_i \leftarrow v_i - q_i$
$q_i \leftarrow 0$
If all of the requests of a single user are not satisfied:
    $j \mathrel{+}= 1$
    GOTO Price
If all of the requests of a single user are satisfied:
    Task trade price$_i = \sum_{m=1}^{n} P_{i,m} * \boldsymbol{s_j} * A_{k,j}$
    Device trade price$_j = \sum_{n=1}^{n} P_{n,j} * \boldsymbol{s_j} * A_{n,j}$
    $i \mathrel{+}= 1$
    GOTO Price
If all user requests are satisfied:
    return Task trade price, Device trade price

5 Slashing Mechanism

One of the most established security mechanisms are Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs). However, zkML is impractical for large machine learning inference tasks (e.g. LLMs, Image Generation, etc.) (see Table 1) [6]. Additionally, zk-SNARKS also have costly memory requirements and high service costs. For instance, the memory consumption for generating an arithmetic circuit in a zk-SNARK for the 7 billion parameter Llama model is in the order of terabytes, if not petabytes [6]. Specifically, the proof generation

time is a significant limitation, which is why we utilize optimize fault proofs. In this, we design a system to incentivize desired behavior to mitigate potentially invalid results.

	Optimistic Proof	**zkML**
Model size	Arbitrary size	Small/Limited
Proof Type	Fraud Proof	zk-SNARK (validity proof)
Proof Speed	Delayed due to challenges	Quick/No delays
Security	crypto-economic security	ZK/Cryptographic

Table 1: Tradeoffs for existing inference verification techniques [6]

We utilize fault proofs to protect users from misbehaving service providers in a slashing mechanism similar to that of Ethereum's Proof-of-Stake model. In this optimistic system, users are able to challenge the results of an inference. Additionally, the Distribute.ai system will also randomly initiate challenges. While a challenge is running, the challenged provider is temporarily barred from accepting tasks. We use a method similar to [7] and the interactive bisection scheme in [8] to determine the validity of the challenger's claim. Since the inference can be represented as a directed acyclic graph (DAG), we can fix the source of randomness to produce deterministic inference processes. Then, a single service provider is assigned to recompute each layer of the node in the topological order of the model DAG. Note that using multiple providers to verify the challenged inference requires full trust since 1-of-n dishonest parties nullifies the correctness of the proof. If at any point there is a discrepancy between the challenged inference and the recomputed inference, the challenger and service providers are rewarded and the challenged service provider is punished. Punishments result in nefarious providers being removed from the network with prejudice and the invalid transaction being reversed. The user's task will then be returned to the auction with higher priority.

References

[1] H. Touvron *et al.*, "LLaMA: Open and Efficient Foundation Language Models." 2023.

[2] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-Resolution Image Synthesis with Latent Diffusion Models." 2022.

[3] D. Jackson, "Next-generation 3D Graphics on the Web — webkit.org." 2017.

[4] C. Xu and W. Song, "Intelligent Task Allocation for Mobile Crowdsensing With Graph Attention Network and Deep Reinforcement Learning," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 2, pp. 1032–1048, 2023, doi: 10.1109/TNSE.2022.3226422.

[5] Y. Zhai, L. Huang, L. Chen, N. Xiao, and Y. Geng, "COUSTIC: Combinatorial Double auction for Task Assignment in Device-to-Device Clouds." 2018.

[6] K. D. Conway, C. So, X. Yu, and K. Wong, "opML: Optimistic Machine Learning on Blockchain." Accessed: Mar. 02, 2024. [Online]. Available: http://arxiv.org/abs/2401.17555

[7] S. Bhat *et al.*, "SAKSHI: Decentralized AI Platforms." 2023.

[8] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD: USENIX Association, Aug. 2018, pp. 1353–1370. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner