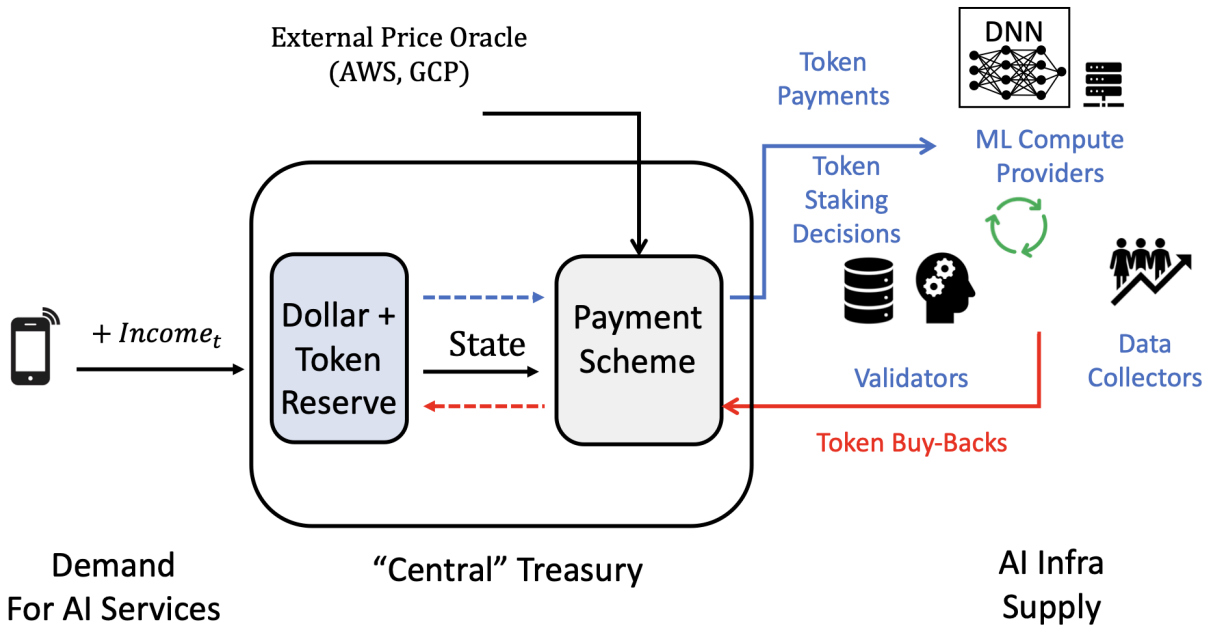
ChainOpera Burn-and-Mint Model

ChainOpera, Inc

Table Of Contents

Overview

This document discusses the Burn and Mint Equilibrium Model, commonly known as BME, for Federated Machine Learning. In the ChainOpera ecosystem, consumers demand AI services (such as training a large language model) provided by a decentralized set of ML infrastructure suppliers. Consumers pay in fiat currency and burn a variable amount of tokens to get access to this service, and suppliers receive a mixture of fiat currency and tokens. The token's purpose is to have its price reflect the overall value of the network. The number of tokens consumers must burn will depend on the current token price and job price. Likewise, supplier rewards are distributed based on a dynamic emission schedule and their individual contribution to ML services.



Consumer Demand for AI Services

Each customer pays cash in USD, which is stored in a central dollar reserve (described next). In exchange for cash, consumers get “service credits”, which give them the right to obtain a certain service from the ChainOpera community (data, train a model, serve a model etc.)

Each consumer class is modeled by a demand time series, with the following standardized fields:

1. Service Type Vector:
    a. This is a list of services each consumer wants, corresponding to a specific type of supplier.
2. Price Vector:
    a. This is a list of prices paid by consumers per unit of service provided.
3. Quantity Vector:
    a. This is a vector of total units of services demanded.
4. Incentive Vector:
    a. Each consumer can bid extra beyond the hourly rate to make their jobs go faster or be picked with a higher demand by service providers. This is similar to network fees in Ethereum smart contracts.

We now provide an example list of consumers, which can be flexibly extended as the ChainOpera ecosystem grows.

Model Training Seekers

1. Service Type Vector
    a. This is a vector of the different types of training machines consumers want.
    b. Example: [TPU, GPU 1X, GPU 2X, …]
    c. You can get this list from AWS, GCP, or ChainOpera.
2. Price Vector
    a. Price for each service in USD per hour.
    b. Ideally, these should be cheaper than AWS or GCP for consumers to use ChainOpera.
3. Quantity Vector
    a. This is the number of hours demanded by different consumers per service, such as training on a TPU.
4. Incentive Vector
    a. These are set by individual consumers to get their jobs picked with high priority during peak congestion times.

Inference Seekers (Model as a Service, Community Model API)

This is very similar to model training, but with different hourly rates. Depending on the project, we can pay per inference query, such as a single question to ChatGPT. Or we can serve a model for a predefined time or contract.

Labeled Data Seekers

1. Service Type Vector
    a. This is a vector of the different types of labels consumers want.
    b. Example: [Bounding Boxes, Segmentation Mask, Classification]
    c. You can get this list from Scale API or Google Cloud Labeling Service.
2. Price Vector
    a. Price for each service in USD per unit (per bounding box etc.)
3. Quantity Vector
    a. This is the number of labeled images demanded by different consumers per service.
4. Incentive Vector
    a. Same as defined earlier.

Pre-Trained Model Buyers (ML NFT)

Customers might also buy a pre-trained ML model for a certain business vertical.

1. Service Type Vector
    a. This is a vector of the different types of pre-trained models we have.
    b. Example: [Medical, NLP, Automotive]
    c. ChainOpera advertises this list on a public dashboard.
2. Price Vector
    a. Price for unlimited access to each model (SaaS Fee).
    b. It can be fixed or be set by price discovery or an auction.
3. Quantity Vector
    a. This is the number of models per domain demanded by each consumer class.
4. Incentive Vector
    a. Not readily applicable since this is not a computational task.

Raw Compute Seeker

This is similar to model training, but customers want to rent a GPU, TPU, or storage unit for a predefined amount of time.

AI Infrastructure Suppliers

Our key step is to (a) delineate the suppliers, and (b) determine how much they are paid in proportion to their differentiated services.

The ChainOpera ecosystem will have the following (non-exhaustive) list of suppliers:

1. AI Infrastructure Suppliers:
    a. Examples include GPU, TPU, CPU, and storage nodes.

2. Labeled Data Suppliers:
    a. These are data annotators who provide a labeled training and test dataset.
3. Model Trainers:
    a. For example, agents who host a GPU and train a specific model to a predetermined accuracy on a given dataset.
4. Model Servers for Real-Time Inference:
    a. For example, an agent who hosts a community-trained LLM on a specific TPU instance.
5. Model Validators
    a. For example, agents who validate that a model achieves a pre-defined accuracy on a publicly available validation dataset.

Consumers to Suppliers

Our next step is to proportionally pay suppliers based on the amount of consumer demand they attract and service. To understand the payment scheme, let us take a few examples.

For model servers, the payment is simple. Suppose a specific server S ran their TPU for 5 hours and served a specific customer C for a total price of $20 (including the gas fee). Then, server S should be directly paid $20, either in fiat or tokens or a mixture there-of.

However, payments are much more subtle for model trainers. If they contributed to a model of $5 for the SaaS Fee, and 5 people bought models, they should get the number of SaaS payments times the number of purchases, which is $25. However, a fraction of this $25 should also be paid to the labelers and validators who also contributed to the process. This will be based on the "value of data for an AI model", which can be assessed using a Data Shapley Model and ChainOpera's Proof-of-Contribution paper.

We formalize the above two examples by defining a payment matrix. Each row is a different consumer type. Each column is a different supplier. For $1 of revenue from a specific consumer, we split that dollar based on the marginal contributions of each supplier, which form the matrix entries. This payment allocation scheme is publicly advertised from ChainOpera and will be dynamically changed to be competitive compared to cloud service providers.

The following is an example payment matrix, where each entry is denoted by $PriceMatrix[i, k]$. Notice that for the last row (an end-to-end AI pipeline), customer income is split among data providers, annotators, model trainers, and inference providers.

	Dedicated Server Provider	Raw Data Provider	Data Annotators	Model Training Providers	Model Inference Providers
Model Training	_____	_____	_____	100%	_____
AI Foundry (dedicated model serving)	100%	_____	_____	_____	_____
Inference seekers	_____	_____	_____	_____	100%
Data annotator Seekers	_____	_____	100%	_____	_____
Training to **serving** pipeline	_____	_____	_____	85%	15%
Annotation to **serving** pipeline	_____	_____	50%	40%	10%
End to **End** AI pipeline	_____	15%	15%	60%	10%

Token Supply

We have a fixed supply of 1 billion tokens. The native token $ChainOpera is a utility token.

Token Funds

We use a standard split between investors, advisors, and the ChainOpera operations fund. The ChainOpera operations fund will have 60% of all tokens reserved for it. This is used for token payments to reward the community for providing decentralized ML services. The token allocation is given below.

Token Fund	**Percentage of Total Token Supply**	**Purpose**

Investors	10%	To attract venture capital investment.
ChainOpera Team	20%	Equity for ChainOpera's core development team.
Advisors	5%	External advisers
ChainOpera Ops Fund	50%	Token rewards to the community to train, serve, and validate ML models etc. Stored in a token reserve on the blockchain.
External Business Development	15%	Incentives for open-source projects or other Web3 companies that ChainOpera partners with, such as Polygon, Theta TV etc.

Economy Phases

ChainOpera's token economy is divided into two phases per project. A project is defined as a large-scale engagement with a specific company or AI endeavor, such as decentralized training of a large language model (LLM). As a running example, we will consider a community-driven LLM training project.

Phase 1 is inflationary, where we reward ("mint") tokens to the community for setting up valuable infrastructure. Once we attract substantial consumer demand, we switch to Phase 2, where suppliers are paid proportionally to the amount of consumer demand that they service.

Notation:

We use the following notation. A specific project is indexed by p. A specific job completed by a supplier is indexed by j. A specific customer is defined by i. A time step, such as a day, is indexed by t. A specific supplier is indexed by k. Note that a specific supplier k can naturally work on several jobs j.

Phase 1: Incentivizing The Growth of AI Infrastructure Supply

Phase 1 is inflationary, where we pay tokens to service providers to incentivize setting up infrastructure. For each project, ChainOpera will advertise the required/target number of GPUs, TPUs, data annotators, storage units etc. required to complete the project. Each supplier (e.g., GPU) from the community will first need to pass a certification test to run ChainOpera's stack, such as running a small ML training job. Once certified, they will be rewarded a certification NFT tied to their specific resource type, such as an

NFT indicating they are a ChainOpera certified GPU. Once they stake this certification NFT, they will be eligible to be rewarded tokens.

In Phase 1, token payments will come from the ChainOpera Ops fund. ChainOpera can select a maximum of $N_{project}$ projects to incentivize in Phase 1, since we have a fixed (limited) token supply. We also want only $Frac.\ PhaseOneTokens\ =\ 60\%$ of the ChainOpera Ops Fund to be paid out in Phase 1. The remainder is saved for Phase 2 token rewards. Thus, the number of tokens per project is given by:

$$PhaseOneTokensForProject[p]\ =\ \frac{Frac.\ PhaseOneTokens \times 0.60 \times MaxTokenSupply}{N_{project}}$$

For any given project, p, the tokens can be equally allocated between the various target suppliers. For example, training an LLM might require 10 TPUs, 5 GPUs, and 5 data labelers. Each certified supplier will need to continually stake its token rewards for the duration of Phase 1 to receive cash flows in Phase 2. Once a project reaches the target number of suppliers and/or begins to attract consumer demand, the project automatically moves into Phase 2.

Phase 2: Paying Suppliers from Organic Demand for AI Services

Once the project attracts consumer demand in Phase 2, each supplier is paid in tokens proportionally to the number of consumers it serviced. The supplier is also paid additional token rewards from the 40% of the ChainOpera Ops Fund dedicated to Phase 2. The specific token payments are detailed later in the whitepaper. To receive token rewards and cash payments, each supplier must pass validation tests from the community to ensure a high quality of service.

First, we delineate *emission rewards* in Phase 2. These are extra incentives from the ChainOpera Ops fund to continue growing the ecosystem. Afterwards, we have a pure burn and mint.

Phase 2: Job Rewards

Job Rewards is a point-based system for suppliers that determines how many tokens a supplier earns on any particular job from Phase 2 rewards. To calculate the reward for a consumer job, we take the price of that job and divide that by the current token price. This takes the job price and denominates it in ChainOpera tokens. From there we divide that by $JobRewardModulation$, an on-chain variable. By default, $JobRewardModulation$ will be set to 10. This will set up the Phase 2 emissions so that suppliers are initially earning roughly 10% extra on the first completed jobs. As time goes on, that number will diminish as you will see in the graph below. The Job Reward for a specific supplier j for servicing customer i is given by:

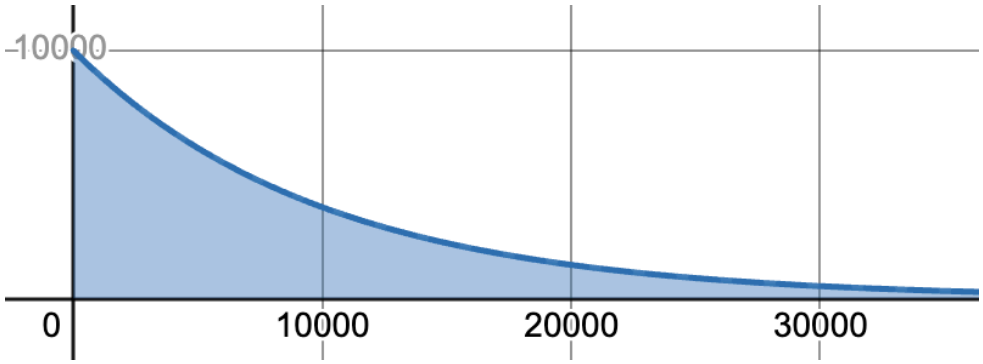
$$JobReward[j]\ =\ \frac{JobPrice[i]\ /\ PriceOfToken[t]}{JobRewardModulation}.$$

Phase 2: Emissions

Now, that we have found the number of Job Rewards for a supplier j, we calculate the total emitted tokens. We note these tokens are an added bonus to the supplier on top of their regular token payments, which are calculated in the next section.
To incentivize suppliers to onboard into the ecosystem, we will reward them with extra tokens in the early phases of ChainOpera. To calculate the supplier token rewards for a particular job, $SupplierReward[j]$, we take the total amount of tokens that will be minted in phase 2, 40% of the ChainOpera Ops Fund, and subtract away the rewards that have already been minted. The below formula has an exponential decay for the emissions. Essentially, initial suppliers get extra "emitted" tokens in Phase 2, while later suppliers have diminishing rewards.

$Phase2Modulation$ is an on-chain variable that will modulate the slope of the supplier reward function. Job rewards will vary depending on the difficulty and length of a job.

$$Phase2SupplierReward[j] = PhaseTwoMintAmount - PhaseTwoMintAmount * e^{Phase2Modulation * \frac{-(JobReward[j] + \sum_t JobRewards[j])}{PhaseTwoMintAmount}} - \sum_t PhaseTwoMintedRewards$$



The graph above shows what the Phase 2 token emissions would look like if the mint amount was 10,000. The y-axis represents the token supply, while the x-axis represents the number of job rewards accumulated on the platform.

Consumer Demand and Total Income

So far, we have calculated extra incentives of job rewards for new suppliers in Phase 2. We now describe their nominal payments, which is directly proportional to the amount of customer demand they service.

First, we calculate the demand for a specific supplier k, such as setting up a GPU. This the sum of how many consumers i we have that require service k as well as the price they pay, given by $PriceMatrix[i, k]$:

$$USDDemandForService[k] = \sum_i NumConsumer[i] \times PriceMatrix[i, k]$$

The total income on a day *t* is the sum of all demand across all suppliers:

$$Income[t] = \sum_k USDDemandForService[k]$$

Total Income

The total USD income per day, $Income[t]$, is a weighted sum of the consumer demand. A fraction $ChainOpera\ Fiat\ Operations\ Fraction = 30\%$ of total income is reserved for ChainOpera to finance and upgrade the ecosystem. This USD income is stored in the dollar reserve, as detailed next. This fraction is an on-chain variable.

Payments

Consumer Payment Mechanism

Anytime a consumer wants to submit a job, they will have to burn tokens. This process is relatively straightforward. We take $BuybackModulation$ of each job payment and use that to buy back tokens. If the consumer doesn't currently hold any tokens, we will take $BuybackModulation$ of their purchase and buy back tokens for them via smart contracts.

The on-chain variable $BuybackModulation$ reflects what fraction of income we want to allocate to token rewards. For example, if it is 100%, we take all net income, buy-back tokens, and pay suppliers proportionally. If it is 0%, we directly pay fiat currency to suppliers (see SEC regulation note below).

To calculate our daily buyback capital, we take $BuybackModulation$ and multiply that by our total income for the day. To get our daily USD income, we get the sum of every job created that day multiplied by the price the consumer paid for that job.

$$USDBuybackCapital[t] = BuybackModulation * \sum_t (JobCreated_t[i] * JobPrice_t[i])$$

This capital goes to token buy-backs and payments.

Next, we calculate how much USD cash flow we have to directly reward suppliers in USD. To calculate our USD payment capital for any given day, we take one minus $BuybackModulation$ and multiply that by our total daily cash flow.

$$USDCapital[t] \ = \ (1 - BuybackModulation) * \sum_{t}(JobsCreated_{t}[i] \ * JobPrice_{t}[i])$$

Due to SEC regulations there is a possibility that cash payments could cause legal issues. In that case, we would increase $BuybackModulation$ to 100% so all of the consumer payments would be strictly in tokens.

Supplier Payment Mechanism

$USDBuybackCapital[t]$ of the payments distributed by the aforementioned matrix gets denominated in ChainOpera tokens. The rest gets paid out to suppliers in US dollars. This mint offsets the burn amount and maintains a Burn Mint Equilibrium. The mint function gets triggered simultaneously with USD payments. These payments happen once validation for the provided service is complete. If a validator attempts to check a supplier's work and it turns out to be faulty, the mint function isn't triggered, and the supplier won't receive any payment.

Once there are no more outstanding consumer jobs, the tokens paid out by ChainOpera each day will equal the number of tokens consumers bought back added to the daily emission rewards. As the formula below demonstrates:

$$SupplierTokensPaid[t] \ = \ \frac{BuyBacksUSD[t]}{PriceOfToken[t] + \Delta PriceofToken[t]} + \sum_{j} Phase2SupplierReward\ [j]$$

Validator Token Payments and Periodic Token Buy-Backs

Validators will also be rewarded tokens for every successful validation. These tokens will be paid from the ChainOpera operations fund. Once the ChainOpera ops fund has only $Lower\ Token\ Reserve\ Limit \ = \ 10\%$ of tokens left, ChainOpera will buy back tokens from the market at their market price to increase the reserves back to 20% of the token reserve. This buy-back decision will be automatically triggered based on the number of tokens in the ChainOpera ops fund. To calculate the amount of validator tokens we pay out in any given timeframe t, we get the sum of all the validations we completed in that timeframe and multiply the result by the $ValidationModulator$, an on-chain variable.

$$ValidatorTokensPaid[t] \ = ValidationModulator \ * \ \sum_{v} ValidationsCompleted[v]$$

Calculating Total Tokens Paid

Calculating the number of tokens the ChainOpera protocol will be paying out on a particular day is straightforward. We just take the number of tokens we paid out to suppliers on day t and add it with the number of tokens we paid out validators on day t

$$TokensPaid[t] = SupplierTokensPaid[t] + ValidatorTokensPaid[t]$$

Calculating Token Supply

To calculate tomorrow's token supply, we take the current supply and add the number of tokens we minted that day. We then subtract the number of tokens we are buying back as shown by the formula below:

$$Supply[t+1] = Supply[t] + TokensPaid[t] - \frac{BuyBacksUSD[t]}{PriceOfToken[t] + \Delta PriceOfToken[t]}$$

Token and Dollar Reserve

We have two reserves on the blockchain, the token reserve and dollar reserve, denoted by $TokenReserve[t]$ and $DollarReserve[t]$. The daily income is stored in the Dollar Reserve and the Token Reserve starts out from the ChainOpera Operations Bank.

Calculating Token Reserve

We use a similar formula to predict the future token reserve. To calculate the number of tokens we will have in the reserve tomorrow, we take the number of tokens in the reserve today and subtract the tokens we paid out, and add the number of tokens we bought back.

$$TokenReserve[t+1] = TokenReserve[t] - TokensPaid[t] + \frac{BuyBacksUSD[t]}{PriceOfToken[t] + \Delta PriceOfToken[t]}$$

Calculating Dollar Reserve

Predicting tomorrow’s dollar reserve is also reasonably straightforward. To do this, we take the current number of dollars in reserve and increase it by the platform’s daily income. Then, we subtract the buybacks that occurred during that day. By doing this, we can accurately predict how many dollars will be in the dollar reserve the following day.

$$DollarReserve[t+1] = DollarReserve[t] + Income[t] - BuyBacksUSD[t]$$

Token Price

The ChainOpera token price is partially set by investor demand on an exchange, making it subject to consumer beliefs. However, the primary factors controlling ChainOpera's token price are the sources of demand built into the platform itself. Together, these sources of demand and the current supply of tokens will determine the token price at any given time.

Platform Token Demand

Platform demand for the ChainOpera token comes from four different sources.

The first is a supplier onboarding fee. Any time a supplier wants to start working on the ChainOpera platform, they must buy back a variable amount of tokens before they can begin. This number is determined by taking a fraction, $OnboardingModulation$, of the average supplier income over the timeframe $t - W$. $OnboardingModulation$ is an on-chain variable whose value can be voted on by governance.

The second is the supplier bond feature. Anytime a supplier wants to accept a job on ChainOpera's platform, they must post a bond in tokens. If a supplier decides to act maliciously while completing the consumer's job, their token bond will be confiscated and sent to the ChainOpera token reserve. A supplier bond is determined by taking a fraction, $BondModulation$, of a particular job's price and dividing that by the current token price. Similar to $OnboardingModulation$, $BondModulation$ will also be an on-chain variable controlled by governance.

The third source of token demand comes from the consumer payment mechanism. Anytime a consumer buys service credits to get access to ChainOpera's job pool, a fraction of that, $BuybackModulation$, goes towards buying back tokens. $BuybackModulation$ is an on-chain variable that determines the amount of consumer income going toward token buybacks. It's worth noting that cash payments to suppliers could become an issue due to SEC regulations. In that case, $BuybackModulation$ will be increased to 100%, and all supplier payments will be denominated in tokens. The $BuybackModulation$ variable will be controlled by a governance vote.

The final source of demand for ChainOpera tokens comes from platform governance. Any user who holds ChainOpera tokens can vote on important decisions regarding the DAO, such as changing the values of the on-chain variables or implementing a new buyback feature.

Calculating Token Demand

Calculating the demand for the ChainOpera token is done by adding up all of the different sources of demand mentioned above along with $ConsumerSentiment[t]$.

$$Demand[t] = ConsumerSentiment[t] + \sum_k SupplierOnboardingFees[k] + \sum_k SupplierBondPurchases[k] + USDBuybackCapital[t] + GovernanceValue[t]$$

Calculating Token Price

The price of the ChainOpera token at any point in time, t, is calculated by taking all of the current demand for the token, and dividing it by the amount of tokens actively in circulation. Thus, the overall ChainOpera token price is given by:

$$PriceOfToken[t] = \frac{Demand[t]}{Supply[t]}$$

Supplier Onboarding Fee

To create some barriers to entry for suppliers on the ChainOpera platform, we will implement a one-time buyback fee to gain access to consumer jobs. This fee will disincentivize malicious suppliers from trying to join the network. The supplier onboarding fee is calculated by taking the platform's total income over a set period and dividing that by the total number of suppliers. This metric is the average amount each supplier earns over that specified period. We then multiply that result by $OnboardingModulation$. $OnboardingModulation$ is an on-chain variable that will modulate the onboarding fee for suppliers. We then divide that by the current token price to convert that number from USD to ChainOpera tokens.

$$SupplierOnboardingFee_k[t] = OnboardingModulation \times \left(\frac{\sum_{t-W}^{t} PlatformIncome[t]}{\sum_{t-W}^{t} Suppliers[k]}\right) / PriceOfToken[t]$$

Once the supplier gets onboarded, the tokens they purchased will be sent to the ChainOpera treasury.

ChainOpera Job Scheduler and Resource Allocation

ChainOpera will develop a novel real-time scheduler that maps consumer jobs to ML infrastructure suppliers. For example, a customer job originating in Los Angeles to run LLM inference within 500 milliseconds will be routed to nearby cloud GPUs in the Western USA with appropriate compute specifications. In essence, each incoming job will contain metadata about its required compute, deadline,

required model, and gas fees. Then, the ChainOpera scheduler will match it to a filtered list of appropriate compute infrastructure with the required SLA and geography. The filtered list will be sorted by the supplier reputation formulas below

Suppliers work gets validated through a secure Proof-of-Training, Proof-of-Inference, and Proof-of-Contribution mechanism. If the suppliers' service is not validated, they will not receive token rewards or fiat payment for that work.

The nodes with the highest reputation will get priority for incoming jobs. Reputation will depend on the number of jobs completed and the job validation rate described below.

Supplier Reputation

Supplier reputation decides who has priority to incoming jobs. Suppliers with the highest reputation will get more access to jobs, rewarding the best participants on your platform and helping to ensure product quality. To calculate the supplier reputation, we take the number of jobs a supplier k has gotten validated over their last $j - W$ jobs, j is the most recent job the supplier has finished. We then divide that by the number of jobs they have completed in that same window. This will give us the supplier's validation to completion percentage over $j - W$ amount of jobs.

$$SupplierReputation_j[k] = \frac{\sum_{j-W}^{j} NumberOfJobsValidated_j[k]}{\sum_{j-W}^{j} NumberOfJobsCompleted_j[k]}$$

If two suppliers have the same reputation the queue will be ordered by the total amount of jobs a supplier has completed.

$$SupplierOrder = \sum_{t} NumberOfJobsCompleted\ [k]$$

This method will make it easier for new suppliers to onboard onto the platform if they are successfully training models. However, it will still reward consistent early adopters of your platform as the job queue gets ordered by the number of jobs a supplier has completed if two suppliers have the same validation to job completion ratio.

Supplier Bonds

When a supplier takes a consumer job, they must post a bond in tokens. This bond is taken from the supplier and transferred to the ChainOpera treasury if their work fails during the validation phase. This bond acts as another mechanism to disincentivize malicious behavior from suppliers on the ChainOpera

platform. To calculate the supplier bond for a specific consumer job, k, we take $BondModulation$, an on-chain variable, and multiply that by the original job price. We then divide that number by the current token price $PriceOfToken[t]$. This gives us the supplier bond amount for that specific job in tokens.

$$SupplierBond[k] = \frac{BondModulation * JobPrice[i]}{PriceOfToken[t]}$$

Simulations

Simulation Setup

We obtain prices from Google Cloud Platform and Scale API to use as our reference prices. ChainOpera may change these to lower values to undercut the market rate, but at least setting the costs at market level makes them competitive in the market.

JOB COSTS	Image Labeling*	TPU Training**	GPU Training**	Inference **
Job Rates	$0.06 / Image	$18.00 / hour	$3.15 per node hour	$1.25 per node hour
Average job sizes***	1000 Images	3 hours of training	8 hours of training	24 hours inference
Cost per Job	$60 / job	$54 / job	$ 25.2 / job	$30 / job

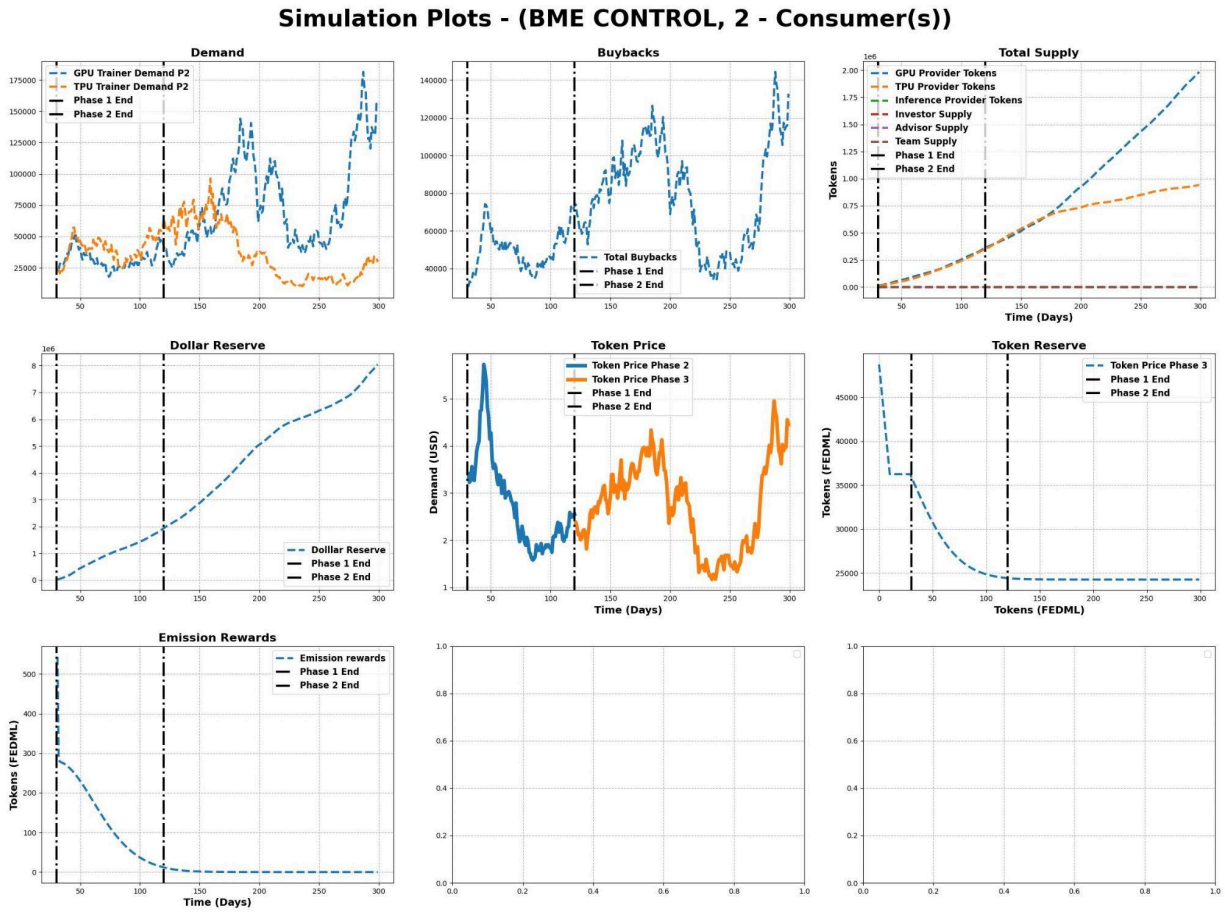
* Taken from scale ai pricing https://scale.com/pricing
** Taken from gcp pricing https://price2meet.com/gcp/docs/vision_automl_pricing.pdf

We used these job costs as the basis and created a demand time series for each of the consumer types. Each class's demand is modeled as a linear time series with gaussian noise added to it to model market variability.

☑

Our phase 1 period is 30 days, and we run the full simulation for 180 days. Phase 1 mints out 60% of its total supply with a token reserve of 2 million.

Simulation Results



In this plot we can see that the token price is rising as the consumer demand is much higher than the token supply put into the system. It is interesting to note that the Dollar Reserve and Token Reserve remain relatively the same in both of the plots. The reason is because BME wants to keep the net change of Token Supply / Dollar reserve at 0. Our key insight is that, even though interest in TPU training (orange, top left) wanes, the token price is robust since the income from GPU training remains increasing.

Tables Of Variables

Modulation Variables

These are all on-chain variables that represent fractions in key formulae.

Variable	Meaning
$BuybackModulation$	An on-chain variable that determines the percentage of income going towards token buybacks

$OnboardingModulation$	An on-chain variable that will modulate the supplier onboarding fee formula
$Phase2Modulation$	An on-chain variable used to modulate the Phase 2 supplier job reward formula
$BondModulation$	An on-chain variable used to modulate the Supplier Bond formula
$JobRewardModulation$	An on-chain variable used to modulate the Job Reward formula

Consumer Variables

Variable	Meaning
$JobCreated_t[i]$	Job's created on day t indexed by consumer i
$JobPrice_t[i]$	Job prices of jobs created on day t indexed by consumer i
$\sum_i NumConsumer\ [i]$	The total amount of consumers on ChainOpera's platform
$PriceMatrix[i,\ k]$	The supplier, k, payment distribution for a particular consumer, i, job

Supplier Variables

Variable	Meaning
$USDDemandForService[k]$	The demand for a particular supplier's service, k, denominated in dollars
$PriceMatrix[i,\ k]$	The supplier, k, payment distribution for a particular consumer, i, job
$SupplierTokensPaid[t]$	The amount of tokens we are paying out to suppliers on day t
$Phase2SupplierReward[j]$	The supplier Phase 2 token reward for a particular job j

$JobReward[j]$	The supplier job reward for a particular job, j
$SupplierOnboardingFee_k[t]$	The onboarding fee in tokens for a particular supplier, k, on day t
$\sum_{t-W}^{t} Suppliers[k]$	The total number of suppliers active on the ChainOpera platform during the last $t - W$ days
$SupplierReputation_j[k]$	The supplier, k, reputation after their most recent job, j, completion
$\sum_{j-W}^{j} NumberOfJobsValidated_j[k]$	The total number of jobs, j, a supplier, k, has gotten validated over their last $j - W$ jobs
$\sum_{j-W}^{j} NumberOfJobsCompleted_j[k]$	The total number of jobs, j, a supplier, k, has completed over their last $j - W$ jobs
$\sum_{t} NumberOfJobsCompleted\ [k]$	The total number of jobs a supplier, k, has completed on the ChainOpera platform
$SupplierBond[k]$	The amount of ChainOpera tokens a supplier, k, will have to post as a bond to accept a particular consumer job

BME Variables

Variable	**Meaning**
$USDBuybackCapital[t]$	The dollar amount from day t's income that is going toward token buybacks
$USDCapital[t]$	Day t USD income that isn't going to token buybacks
$BuyBacksUSD[t]$	The amount of tokens we bought back on day t denominated in dollars
$PriceOfToken[t]$	The price of the ChainOpera token on day t
$\Delta PriceOfToken[t]$	The extra incentive price we pay to buy back tokens from the market

$\sum_{t} Phase2SupplierReward\ [j]$	Tokens rewarded to suppliers on day t from the Phase 2 emission formula
$PhaseTwoMintAmount$	The amount of tokens that will be minted during Phase 2 emissions
$Supply[t]$	The supply of ChainOpera token's on day t
$TokensPaid[t]$	The total amount of ChainOpera token's paid out on day t to suppliers and validators
$TokenReserve[t]$	The amount of tokens in ChainOpera's token reserve on day t
$DollarReserve[t]$	The amount of dollars in ChainOpera's dollar reserve on day t
$Income[t]$	Total income for the ChainOpera platform on day t
$\sum_{t-W}^{t} PlatformIncome[t]$	The total platform income over the last $t - W$ days where t is the starting point and W is how far in the past we are indexing

Project Variables

Variable	**Meaning**
$PhaseOneTokensForProject[p]$	The amount of token's a particular project, p, will receive out of the ChainOpera Ops fund during the Phase 1 token rewards
$N_{project}$	The number of projects ChainOpera is funding through its ops fund.