CamelidCoin

A Blockchain-based Protocol for Trustless Distributed Auto-Regressive LLM Computation and Training.

Dylan Dunn
contact@dylandunn.me

`https://camelcoin.org`

May 5, 2023

1 Abstract

CamelidCoin is a trustless, incentive-based blockchain protocol designed for distributed auto-regressive large language model (LLM) computation and training. The system is built on a peer-to-peer network where lightweight clients can submit input for completion by a large pool of compute nodes. In exchange for their work, the compute nodes are compensated by the client. To ensure the validity of the output from these compute nodes, we propose a new algorithm called Random Autoregressive Subsampling for Token Integrity Checking (RASTiC), which can be completed either by the client or other compute nodes. This algorithm quickly verifies the output's authenticity with near certainty. Additionally, throughout the paper, we will detail the protocol's structure and specifics. Our protocol works off of the original peer-to-peer electronic cash system proposed by Satoshi Nakamoto's Bitcoin and modifies it to fit our use case. We will also discuss current challenges and limitations with our current protocol.

Contents

1 Abstract **1**

2 Introduction **2**
2.1 Related Work 2

3 Technical Overview **3**
3.1 Network 3
3.1.1 Peer Discovery 3
3.1.2 Messaging Protocol 3
3.2 Wallet 3
3.2.1 Keypair Creation 3
3.2.2 Public Address 4
3.3 Transaction 4
3.3.1 Structure 4
3.4 Transaction Structure 4
3.4.1 Proof of Work 5
3.5 Jobs 6
3.5.1 Job Pool 6
3.5.2 Dispute Resolution 6
3.6 RASTiC Random Autoregressive Subsampling for Token Integrity Checking 7
3.6.1 Problem Statement and Motivation 7
3.6.2 Implementation and Methodology 7
3.7 Evaluation 8
3.8 Rationale 8

4 Challenges and Limitations **9**
4.0.1 Homomorphic Encryption 9
4.1 Inefficient Delegation 9

5 Future Aspirations **9**
5.1 Distributed Training 9
5.1.1 Federated Averaging 10
5.2 Proof Of Stake 10
5.3 Improved Privacy 10
5.4 Using job completion as POW 10

6 Conclusion **10**

2 Introduction

2.1 Related Work

The concept of distributed computation is not new, and several existing projects utilize it for various purposes.

One notable project is the Folding@home project, which uses distributed computing to simulate protein folding for research purposes. However, this project requires dedicated software to be installed on users' machines, and the computation is not incentivized beyond the altruistic motives of the participants.

Another project is Golem, which aims to provide a decentralized, global supercomputer. It utilizes the Ethereum blockchain to allow users to rent out computing power or request computation power for their own purposes. However, it does not focus on LLM computation specifically, and its model does not involve any validation of the output.

Finally, several projects in the blockchain space aim to provide decentralized machine learning services, such as SingularityNET and Ocean Protocol. These projects allow users to request machine learning services from a decentralized network of providers. However, their focus is on traditional machine learning tasks, and they do not specifically target auto-regressive LLMs.

In contrast to these existing projects, CamelidCoin is specifically designed for distributed auto-regressive LLM computation and training, with a focus on trustless and incentivized computation. Additionally, our proposed RASTiC algorithm provides a novel approach to output validation that enables the system to operate without relying on trust between the client and compute nodes.

3 Technical Overview

3.1 Network

3.1.1 Peer Discovery

The CamelidCoin network is a peer-to-peer network, where each node connects to other nodes directly. When a node first joins the network, it needs to discover other nodes to connect to. We use a modified version of the Kademlia distributed hash table (DHT) protocol for peer discovery.

In the CamelidCoin network, each node has a unique identifier, represented as a hash. When a node wants to join the network, it sends a message to a bootstrap node (a well-known node in the network) containing its identifier. The bootstrap node responds with the K closest nodes to the new node's identifier. The new node then connects to these nodes and starts exchanging messages with them.

When a node receives a message from a new node, it adds the node to its list of known peers. The node also periodically sends out "ping" messages to its known peers to check if they are still online. If a node does not respond to a ping message for a certain amount of time, it is marked as offline and removed from the list of known peers.

3.1.2 Messaging Protocol

The CamelidCoin messaging protocol is used for communication between nodes in the network. It is a simple protocol based on sending and receiving messages. Each message consists of a header and a payload. The header contains information such as the sender's and receiver's addresses, the message type, and the length of the payload. The payload contains the actual message data.

The messaging protocol is used for several purposes in the network. Nodes use it to broadcast transactions and blocks, to request information from other nodes (such as the list of known peers), and to exchange job and output data.

Messages in the CamelidCoin network are sent over TCP/IP connections. Each node maintains a number of connections to other nodes in the network, which it uses for sending and receiving messages. When a node receives a message, it checks the header to determine the type of message and the intended recipient. If the message is intended for the node, it processes the payload and sends a response if necessary. If the message is not intended for the node, it forwards it to the appropriate recipient based on the message header.

3.2 Wallet

3.2.1 Keypair Creation

To create a new keypair, we use the Elliptic Curve Digital Signature Algorithm (ECDSA) with the secp256k1 curve, the same algorithm used by Bitcoin. The process for creating a new keypair is as follows:

2. Generate a new private key k, which is a random 256-bit integer.
3. Calculate the corresponding public key K by multiplying the curve's generator point G by the private key: $K = k \times G$.

3.2.2 Public Address

Let PK be the public key to convert to a CamelidCoin address.
Let $h_1 = \text{SHA-256}(PK)$ be the SHA-256 hash of the public key.
Let $h_2 = \text{RIPEMD-160}(h_1)$ be the RIPEMD-160 hash of the SHA-256 hash.
Let $buf = 0x00$ be a buffer with value 0x00.
Let $h_3 = \text{SHA-256}(buf||h_2)$ be the SHA-256 hash of the concatenated buffer and RIPEMD-160 hash.
Let $h_4 = \text{SHA-256}(h_3)$ be the SHA-256 hash of the previous result.
Let $chksum$ = first 4 bytes of h_4 be the first 4 bytes of the second SHA-256 hash.
Let $addr = \text{Base58Check}(buf||h_2||chksum)$ be the concatenated result encoded in Base58Check encoding.

Note: This formula assumes that PK is a valid public key and that all hash functions return values in their hexadecimal representation. It also assumes that the concatenation operator is denoted by "||".

3.3 Transaction

A CamelCoin transaction is a data structure consisting of several fields:

- **Version**: A 4-byte field indicating the transaction version number.
- **Input Count**: A variable-length integer indicating the number of inputs in the transaction.
- **Inputs**: A list of transaction inputs, where each input contains the following fields:
  - **Transaction ID**: A 32-byte field indicating the transaction ID of the previous transaction output being spent.
  - **Output Index**: A 4-byte field indicating the index of the previous transaction output being spent in the list of outputs.
  - **ScriptSig**: A variable-length field containing a script that satisfies the conditions of the previous transaction output being spent.
  - **Sequence**: A 4-byte field indicating the transaction version number.
- **Output Count**: A variable-length integer indicating the number of outputs in the transaction.
- **Outputs**: A list of transaction outputs, where each output contains the following fields:

  - **Value**: An 8-byte field indicating the amount of CamelCoins being transferred.
  - **ScriptPubKey**: A variable-length field containing a script that defines the conditions under which the output can be spent.

- **Lock Time**: A 4-byte field indicating the earliest time or block height at which the transaction can be added to the blockchain.
- **Job ID**: A 32-byte field indicating the hash of the job that the transaction was used to pay for, if applicable.

3.3.1 Proof of Work

Proof of Work (PoW) is a consensus mechanism that is used to validate transactions and generate new blocks in the CamelidCoin blockchain. The idea behind PoW is to require computational work to be done by a node in order to add a new block to the blockchain. This computational work takes the form of solving a complex mathematical puzzle that is computationally difficult to solve, but easy to verify.

The specific PoW algorithm used in CamelidCoin is SHA-256, which is the same algorithm used in Bitcoin. When a node wants to add a new block to the blockchain, it must first solve a puzzle based on a set of block headers. The header includes the block's timestamp, the hash of the previous block, a hash of the current transactions, and a nonce. The nonce is a 32-bit value that is incremented by the node until the block header's hash meets a certain target difficulty. The difficulty is determined by the current network hashrate, which is the total computational power being used by all nodes on the network.

Nodes compete to solve the puzzle, and the first node to solve the puzzle and find a valid hash can add the new block to the blockchain. This node is rewarded with a block reward, which is currently set at 50 CamelidCoins. The block reward halves every 1,000,000 blocks, similar to Bitcoin. However, we have reduced the frequency of the block reward halving in order to account for a far greater number of transactions per second.

PoW is an energy-intensive process, and as such, it has received criticism for its environmental impact. However, it is a proven and secure consensus mechanism that has been used by Bitcoin for over a decade. In the future, CamelidCoin may explore alternative consensus mechanisms that are more energy-efficient, such as Proof of Stake or Proof of Authority.

3.4 Jobs

A job consists of parameters that must be computed using an auto-regressive LLM, including an input string, seed, number of tokens to generate, timestamp, timelock, client address and port, and the model to use (e.g. ggml-alpaca-7b-q4). The client appends a signed transaction with a reward and fee, where the reward is a transaction into an escrow pool and the fee is paid to the miner who includes the transaction in the block. The client broadcasts a job creation message including the jobs parameters. Compute nodes that are available and capable of completing the job will start computing the output and broadcast that they have accepted the job. When either the end token or token limit is reached, the node appends an output address to receive the job reward. Then, the node encrypts the output with the job creator's public key and signs the hash of the input parameters, output, and reward address. The node also creates a staking transaction that is only valid if the output fails validation, to testify the output validity. The client receives the broadcasted testament transaction and, upon receipt of the signed output hash, creates a new transaction to replace the original one and sends it directly back to the compute node. The compute node replies with the output encrypted with the client's public key. The client verifies the output using the RASTiC algorithm, whose implementation will be covered later, locally if it has the full model or by requesting neighboring nodes to perform the check. While this verification step can be skipped, checking helps identify bad actors among the nodes.

3.4.1 Job Pool

The job pool operates similar to a mempool in traditional blockchains, where each full node maintains a record of every job it observes and updates their status from "created" to "accepted" to "completed." Disputes for jobs can only occur when they are in the job pool. When a miner discovers a new block, it may include some or all of the transactions in the job pool in the blockchain. After a certain block height chosen by the miner, jobs will be discarded as they cannot be retroactively invalidated without mining a fork of the blockchain. Thus, jobs have a time limit for being invalidated unless a malicious miner is involved. Additionally, jobs cannot be included in a block until at least 10 minutes after their creation time, setting a minimum time for them to be invalidated. The parameters, input, and output for jobs are not stored on the blockchain. Instead, only a job ID, a hash of it's parameters, are saved in the transaction properties. Overall, this design establishes a commitment from both parties before information is exchanged without requiring a separate escrow transaction or payment channel.

3.4.2 Dispute Resolution

During the period in which jobs are in the job pool, they may be contested for a duration of approximately 15 minutes. Each contestable situation consists of two components: the proof that can be presented as evidence of a contract violation, and the counter-proof that provides evidence that the accused violation did not occur.

Table 1: Dispute Situations and Their Outcomes

Claim	**Proof**	**Counter**	**Result**
Client doesn't send payment with updated destination.	Non-existance of updated client payment transaction	Existence of client payment message	Compute node forfeits payment, Client doesn't receive output.
Compute node gave invalid output	Output commitment fails RUSTiC test.	Re-test shows otherwise	Compute node forfeits stake, Client commitment to escrow is invalidated.

3.5 RASTiC Random Autoregressive Subsampling for Token Integrity Checking

3.5.1 Problem Statement and Motivation

The current process of outsourcing the computation of auto-regressive models lacks a way to verify that the computation has been performed accurately. In situations where fulfilling requests is compensated, this creates a strong incentive to answer every request, regardless of whether the computation has actually been done.

At present, clients have two options: to perform the computation themselves, which is redundant, or to ask multiple nodes to compute the output, which is also inefficient. Therefore, a new algorithm is needed that employs an asymmetric difficulty, where generating outputs is difficult (currently $O(n^2)$), but checking the validity of these outputs is fast and efficient ($O(1)$).

Thus, we propose the RASTiC algorithm as a solution to this problem. RASTiC allows for efficient verification of the outputs generated by an auto-regressive model, providing a reliable way to verify that the computation has been performed accurately without the need for redundant computations.

3.5.2 Implementation and Methodology

The RASTiC algorithm states that if the following assertions hold true we can be confident that the output is valid.

Let X be a random integer between 2 and $(A+B)-2$.
Let A be the length of the input in tokens.
Let B be the length of the output in tokens.
Let T be the maximum number of tokens to generate.
Let R be the concatenation of A followed by B.
Let F be our auto-regressive model function.
We must ensure that the following condition is true:

$$F(\mathrm{R}_{[0:A]}) = \mathrm{R}_{A+1}$$

We must ensure that the following condition is true:

$$F(\mathrm{R}_X) = \mathrm{R}_{[X+1]}$$

Lastly we must ensure that

$$\begin{cases} F(\mathrm{R}_{[A+B-1]}) = \mathrm{E} & \text{if } \mathrm{R}_{[A+B-1]} \text{ is the } <\text{end}> \text{ token} \\ F(\mathrm{R}_{[A+B-1]}) = F(\mathrm{R}_{[A+B]}) & \text{if R has length T} \end{cases}$$

3.6 Evaluation

The chances of a false positive using RASTiC can be calculated as follows. Suppose we have two sequences of numbers, each with 50,257 tokens (e.g., using GPT-2 as an example). If we choose 3 random indices in each sequence, the probability that the numbers match at all 3 indices is:

$$\text{Probability of a match} = \frac{1}{50,257^3} = 7.78 \times 10^{-15}.$$

Therefore, the chances of a false positive in RASTiC are extremely low, even with a smaller vocabulary size such as with GPT-2.

3.7 Rationale

Sample a subset of the data. Test for continuity, integrity, and truncation:

- Continuity: Ensure that the output is connected to the input. This prevents a case where the output is generated, but is unrelated to the input. This would produce a message where the linkage of tokens is relatively connected, but would be broken at the intersection of the input and output token arrays.
- Integrity: Randomly test the interior tokens. Since any one token can be tested it enforces the content of the message to not only be genuinely generated, but also ensures that it hasn't been tampered with since tampering would break every token after it.

- Truncation: Ensure that the message has not been truncated. Ensure that an ⟨end⟩ token has been reached and that the compute note did not prematurely truncate the completion.

With these tests, we can be fairly sure of the integrity of the entire output.

Potential Attacks and Exploits

There are a couple of exploits that we can foresee being used to bypass the algorithm.

1. **Repetition:** By using very repetitive outputs, we could theoretically force a false positive. For example, if the tokens for 1 2 3 4 5 6 are seen one after each other, the language model may latch onto the pattern and fill the rest of the output with the pattern, allowing the compute node to bypass expensive computation. However, this is why we test for continuity. Theoretically, the only exploitable inputs would be highly repetitive, and even then, we haven't achieved anything since we can't deviate from the pattern, the pattern that is very close to or identical to truth. However, we do see this as a possible path for exploitation in the future.

2. **Trends towards truncation:** While we weren't able to produce such a token, sequences could be forced to quickly generate the ⟨end⟩ token early. However, the node would need to be able to do this more than 51% of the time or would lose money through slashed stakes.

3. **Inference:** Theoretically, because of the limited vocabulary of everyday inputs, lazy prediction algorithms could be used to predict words without computing exploits. But, if the node were able to do this effectively enough, this is but an evolution of the network since it would need to be right at least 51% of the time as not to lose money.

4 Challenges and Limitations

4.0.1 Homomorphic Encryption

4.1 Inefficient Delegation

5 Future Aspirations

5.1 Distributed Training

In the process of creating a self-incentivizing economy, we aim to build an enormous collection of compute power. With this, we can allow for distributed training of the model, and create regular checkpoints to track progress. We believe that nodes will have a strong incentive to contribute to this effort, as the better the model, the more in demand the network will be. This will increase the cost per token, which compute nodes can sell back to clients, creating a self-sustaining ecosystem. Therefore, we will not need to provide monetary incentives for nodes to participate in this process.

5.1.1 Federated Averaging

Federated learning allows for a democratic training process, where the model is shaped by the collective power of the network. This allows any data to be slowly incorporated into the model. In this process, nodes are free to train any dataset they wish to see incorporated into the model. At set blockchain heights, nodes take time to slow average out their weights with their peers, creating a centralized model which can be agreed upon through several stages of proof of work. The specifics of the federated averaging process would need to be explored in detail at a later date.

5.2 Proof Of Stake

In the future, we hope to switch over to a proof of stake model. This is a challenging task for an emerging coin, as without a large market cap, a minority can take over a large portion of the voting. However, once the coin has stabilized and has a sufficient market cap, we plan to follow in the footsteps of Ethereum and make the switch. By doing so, we will be able to eliminate the wasteful compute associated with traditional Proof-Of-Work and move towards a more sustainable and eco-friendly system.

5.3 Improved Privacy

5.4 Using job completion as POW

6 Conclusion