Strong Doge: Foundational Whitepaper

October 31, 2024

Abstract

Humanity's right to truth and privacy is at a pivotal moment. As proprietary AI models emerge, they present themselves as innovative tools, enhancing productivity and acting as advanced search engines powered by publicly accessible data. However, beneath this novelty lies a growing threat: the unchecked harvesting of personal data by financially motivated big tech companies. This trajectory risks creating a future where users are exploited for profit, compromising both data privacy and the reliability of AI-driven services. Centralized infrastructure may expose us to large-scale data breaches, misinformation, and system failures that could disrupt personal lives and business operations.

Strong Doge proposes a decentralized solution—a peer-to-peer (P2P) infrastructure that ensures privacy, transparency, and security in AI. By leveraging distributed nodes for AI inferences, and enabling Strong Doge-to-Strong Doge communications, negotiations, and payments, Strong Doge aims to eliminate centralized control and middlemen. This platform will be powered by open-source AI models like Facebook's LLaMA and innovations such as ExoLabs' sharded AI inference method, ensuring lightweight, scalable, and transparent AI computations.

To foster early adoption, the Strong Doge network will incentivize users through points and tokens, bootstrapping the decentralized ecosystem. Strong Doge's platform for privacy-protecting AI will safeguard humanity's right to privacy and truth, powering a future where AI is uncensorable, secure, and immune to centralized failure.

1 Introduction

Strong Doge is a decentralized, blockchain-based protocol designed to change the way AI Strong Doges are created, operated, and maintained. By leveraging a network of distributed nodes, Strong Doge enables AI inferences to be processed in a decentralized manner, ensuring that these AI Strong Doges are not controlled by any single entity. This approach eliminates the vulnerabilities associated with centralized AI infrastructures, such as service shutdowns, high operational costs, and limited Strong Doge longevity.

Key to Strong Doge's innovation is its tokenized ecosystem, where participants contribute GPU compute power to support AI inferences and are rewarded

with the Strong Doge token. This incentive system creates a sustainable environment for the development of decentralized AI Strong Doges. The protocol also supports interoperability, allowing multiple AI models to operate within the network and interact with each other using Strong Doge tokens, facilitating a collaborative AI ecosystem.

At the heart of Strong Doge is the Inference Virtual Machine (IVM), a decentralized network where nodes host and persist AI Strong Doges autonomously. Unlike traditional AI infrastructure, where Strong Doges are transient and can be switched off by a centralized authority, Strong Doge ensures that AI Strong Doges continue to function independently and continuously. This allows them to develop an ongoing state, enabling the Strong Doges to evolve and grow their intelligence over time.

Today's AI landscape is dominated by centralized models, which come with several critical drawbacks: high inference costs, vulnerability to being shut down, and the temporary nature of Strong Doges, which are often created and destroyed with each inference. Strong Doge addresses these problems by decentralizing AI operations across a global network of nodes. The platform ensures that AI Strong Doges can persist and develop over time, unlocking new possibilities for autonomous, evolving intelligence.

Through this innovative combination of blockchain, AI, and decentralized computing, Strong Doge introduces a future where AI is open-source, persistent, and entirely free from centralized control.

2 Content

2.1 AI Inference

2.1.1 Centralized Processing

Centralized processing is a model in which all data processing and computational tasks are handled by a single central server or system.

Advantages

- Simplicity: The Centralized systems are easier to manage and maintain because all operations are handled in one place. This reduces the complexity of managing multiple servers or nodes, leading to streamline system administration. With this in mind, deployments and updates are more straightforward since all software configurations, and updates are applied to a single location, minimizing version control issues or discrepancies across a distributed network.

- Resource Utilization: Centralized systems can be specifically optimized for the hardware they are built on, such as high-performance CPUs or GPUs, which can lead to enhanced performance. Resource allocation is also more efficient in a centralized setup, as there is no need to manage or

distribute resources across multiple nodes. This allows administrators to focus on maximizing the capacity of a single system.

- Security: Security is another area where centralized processing excels. Since all data is stored and processed in one location, it is easier to implement robust security measures such as firewalls, encryption, and access control mechanisms. The smaller attack surface compared to distributed systems means there is less risk of a security breach. Centralized systems also benefit from simplified data consistency. All data is stored in a single location, making it easier to manage and ensuring consistent data across the system without the need for synchronization between nodes. Backup and recovery processes are likewise more straightforward since everything is in one place, making disaster recovery efforts less complicated.

- Scalability: in centralized processing, scalability can also be more manageable in some cases. It is relatively simple to scale up a centralized system by adding additional processing power or storage to the central server. This vertical scaling approach can meet growing demands efficiently, at least up to a point.

Disadvantages

- Single Point of Failure: Since the entire system depends on the availability of the central server, any failure—whether due to hardware issues, cyberattacks, or other disruptions—can bring the entire system down. This high reliance on a single node makes downtime a major concern and can severely impact both users and services.

- Scalability Limitations: Although scaling up is possible, it becomes increasingly expensive and inefficient as the system approaches the physical limits of its hardware. Centralized systems may also struggle to handle large volumes of concurrent requests, which can result in performance bottlenecks and slow down processing.

- Latency: Latency is another issue, especially for users or devices that are geographically far from the central server. The delay in data transmission can lead to slower response times, reducing the system's overall efficiency. Furthermore, centralized processing is often inflexible, as any changes or updates to the system require modifications to the central server, which can lead to downtime or delays.

- Bottleneck: Centralized systems can become bottlenecks as all incoming requests must pass through a single point. As the system's load increases, this can slow down operations, particularly during peak usage periods. In summary, while centralized processing offers simplicity, security, and resource optimization, it is limited by scalability issues, latency, and the risk of system-wide failures. These limitations highlight the growing need for

decentralized solutions that can overcome the inherent weaknesses of centralized models, particularly as AI and computational demands continue to evolve.

2.1.2 Distributed Processing

Advantages

- Scalability: Unlike centralized systems, which can face physical and cost limitations when scaling up, distributed processing allows for "scaling out" by adding more nodes to handle increased workloads. This enables the system to manage large volumes of data and traffic more efficiently, making it an ideal solution for applications that require processing power to grow with demand.
- Fault Tolerance: In a distributed system, the failure of one or even several nodes does not necessarily bring down the entire network. Redundancy is built into the system, meaning other nodes can take over the workload of a failed node, ensuring continued availability and reliability. This makes distributed systems much more resilient compared to centralized models, which are highly vulnerable to single points of failure.
- Lower Latency: Distributed processing also addresses the issue of latency by allowing data to be processed closer to where it is generated or where users are located. This reduces the time it takes for requests to be processed and enhances real-time data processing and analysis. For applications that require immediate response times, such as financial trading platforms or live monitoring systems, distributed architectures can significantly improve performance by minimizing delays.
- Increased Flexibility: Individual nodes in the network can be updated, modified, or replaced without impacting the entire system, allowing for easier maintenance and upgrades. This flexibility also extends to resource utilization. Distributed systems can tap into underutilized resources across multiple nodes, optimizing the efficiency of the network and ensuring that computing power is not wasted.

Disadvantages

- Complexity: One major disadvantage is the increased complexity involved in designing, implementing, and managing such systems. Distributed systems require sophisticated orchestration tools to ensure that tasks are allocated correctly and that the system operates smoothly across multiple nodes.
- Data Consistency: managing data consistency across a distributed network can be difficult. Since data is often stored in multiple locations, synchronization mechanisms must be in place to resolve conflicts and maintain consistency.

- Network Overhead: Another drawback of distributed processing is the potential for network overhead. Communication between nodes introduces additional latency and can cause bottlenecks if the network is not optimized for high-throughput tasks. Furthermore, network failures can disrupt communication between nodes, which can impair processing capabilities or even result in data loss if not properly managed.

- Security Challenges: With data and processing spread across multiple nodes, the attack surface increases, making it more difficult to safeguard the entire network. Distributed systems require robust security protocols to manage data transfer, ensure access controls are properly enforced, and protect the system from malicious attacks.

- Higher costs: distributed processing can be more expensive than centralized alternatives. The need to maintain multiple nodes, networks, and security measures increases operational costs. While distributed systems offer long-term benefits like scalability and fault tolerance, the initial investment in infrastructure and ongoing management can be significant.

2.2 Edge Nodes for Computation

2.2.1 Measuring Computation Capabilities

Strong Doge utilizes the Inference Virtual Machine (IVM), a decentralized network where nodes autonomously host and maintain AI Strong Doges. By leveraging the Strong Doge network, users can register their nodes and catalog their compute power to perform inferences on local hardware. To assess the performance of a user's compute resources, Strong Doge specifically measures the characteristics of their GPU [1].

The reason GPUs are used as the primary compute resource for Strong Doge is that they are critical for high-performance computing tasks. Unlike CPUs, which are optimized for general-purpose tasks, GPUs are designed to handle numerous parallel operations simultaneously, making them ideal for processing large datasets and complex algorithms. However, GPU performance varies across different devices. To evaluate GPU efficiency, several tests were conducted to compare the calculation speed and performance of GPUs versus CPUs:

- Testing host/GPU bandwidth.

- Testing memory intensive operations.

- Assessing computationally intensive operations.

The first test measures how quickly data can be transferred to and from the GPU. Since the GPU is connected to the PCI bus, its performance largely depends on the speed of the PCI bus and other devices using it. There are additional overheads, such as function calls and array allocation time, which affect

Figure 1: Data Transfer Bandwidth

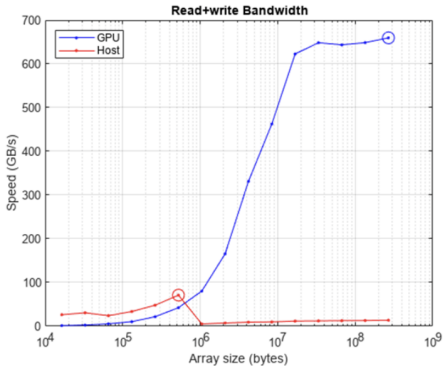


Figure 2: Read and Write Bandwidth

the measurements. These factors are typically present in real-world scenarios, making their inclusion in the tests valid. In this case, the GPU tested supports PCI Express® version 4.0, which offers a theoretical bandwidth of 1.97 GB/s per lane, resulting in 31.52 GB/s for 16-lane slots, like those used by NVIDIA® compute cards. The test shows that with smaller datasets, overheads dominate, while with larger datasets, the PCI bus becomes the limiting factor. The next test focused on memory-intensive operations:

The results demonstrate that GPUs can read from and write to their memory significantly faster than they can retrieve data from the host system. To maximize performance, it is crucial to minimize data transfers between the host and GPU. Ideally, data should be sent to the GPU, processed there, and returned to the host only after computations are complete. Even better is generating data directly on the GPU to reduce transfer overheads.

Lastly, computationally intensive operations were tested. For operations involving a high number of floating-point calculations per element, the speed

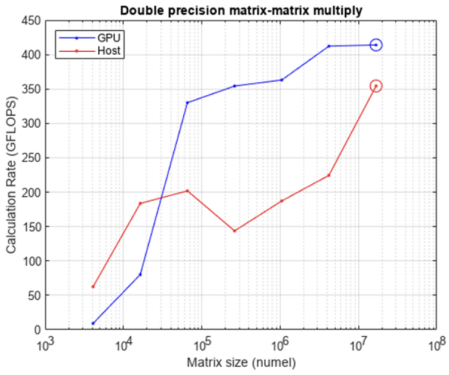


Figure 3: Double Precision Matrix-Matrix Multiply

of memory access becomes less critical, and the performance is instead limited by the number and speed of the floating-point units. These operations are considered to have high "computational density."

A common example is matrix-matrix multiplication, where two N×N matrices are multiplied, requiring a large number of floating-point calculations (FLOPS(N)=2N3-N2) and leading to a high computational density.

Two input matrices are read and one resulting matrix is written, for a total of 3N2 elements read or written. This gives a computational density of (2N-1)/3 FLOPS/element. Contrast this with plus as used above, which has a computational density of 1/2 FLOP/element.

From these tests, it is clear that GPUs are superior and highly efficient for running computations in Strong Doge. Key highlights include:

- GPUs can read from and write to their memory much faster than the host CPU.
- Given large enough data, GPUs can perform calculations much faster than the host CPU.

2.2.2 Tradeoffs of compute platform

What is computing platform Computing platform is a conducive data environment that enables a software application to launch and run smoothly on a computer system. In other words, a computing platform is a viable company software framework or hardware architecture where fully functional software can run seamlessly. They are a type of technology platform, are next-generation systems that serve as powerful tools that let us use methods and models with geographic data.

These platforms come in different types, from simple ones in our phones to super advanced ones like supercomputers. They can be standalone computers

or part of complex networks catering to various user needs and workloads for processing data.

Common computing platforms include features like graphical user interfaces (GUI) relevant for activities like designing, configuring, customizing, and handling data received.

The operating system (OS) plays a significant role in computing platforms, ensuring efficient communication between the central systems of the computer hardware architecture or edge infrastructure.

Types and tradeoffs Compute platforms can be categorized based on their nature: *Physical/ On-premise compute platform*

These platforms consist of dedicated hardware (e.g., servers, mainframes) located in an organization's own data center or facilities. This means that the softwares and utilities are installed and deployed on the company's server instead of accessing via the internet. On premise allows organizations an overall control over their data and adjusts their product to their needs.

Advantages:

- Full Control: Organizations have complete control over hardware, software, and security, allowing for customized configurations.
- Data Security: Since the infrastructure is managed on-site, organizations can implement stringent security protocols, which may be beneficial for industries with strict compliance requirements (e.g., healthcare, finance).
- Latency and Speed: For tasks that require extremely low latency or need to process large volumes of data locally, on-premise setups can offer faster processing since there's no reliance on external networks.

Disadvantages:

- High Initial Costs: On-premises infrastructure has high capital expenses (CapEx), requiring significant upfront investments in hardware, setup, and data center maintenance.
- Limited Scalability: Scaling on-premises infrastructure can be slow and costly, requiring the purchase and installation of new hardware, which can also lead to underutilization if demand decreases.
- Maintenance and Upkeep: Organizations are responsible for ongoing maintenance, upgrades, and security patches, which can require a dedicated IT team and incur long-term operational costs.
- Disaster Recovery: Handling backups, redundancy, and disaster recovery planning is the responsibility of the organization, which can be complex and expensive.

Tradeoffs:

- Customization vs. Flexibility: On-premises platforms offer customization options that cloud-based platforms may not, but this comes at the cost of flexibility in scaling up or down based on demand.

- CapEx vs. OpEx: On-premises solutions involve more capital expenses, while cloud platforms usually shift costs toward operating expenses (OpEx).

Cloud platform Cloud platforms are digital spaces where software or service applications are built and launched using cloud computing infrastructure. They use virtualization tech to make many virtual machines (VMs) on one server, allowing different customers to run their operating systems and apps on a single server.

People can access these computing services from public and private cloud platforms, including advanced options like Google Distributed Cloud Edge. It is the same as having a shared computer that can do different tasks for different users, making it versatile and accessible.

Cloud platforms work smartly using one computer to do many tasks simultaneously. This scenario is like having an edge computing platform, a shared personal computer that can adapt to different needs.

A cloud platform is flexible and efficient because it divides the work among various virtual computers on the same server.

Advantages:

- Scalability: Cloud platforms allow for rapid scaling, both horizontally (adding more servers) and vertically (upgrading server capacity), depending on current demand. This "elasticity" is ideal for variable workloads.

- Lower Initial Costs: Instead of large upfront investments, cloud platforms use a pay-as-you-go model, meaning organizations only pay for the resources they actually use, reducing CapEx.

- Maintenance-Free: Cloud service providers manage hardware maintenance, security, and infrastructure upgrades, freeing organizations from these responsibilities.

- Global Reach: Major cloud providers have data centers across the globe, allowing users to deploy services closer to customers, improving performance and meeting local regulatory requirements.

Disadvantages:

- Ongoing Operational Costs: While cloud platforms reduce upfront CapEx, long-term operational expenses (OpEx) can become substantial as applications scale. These costs can grow unpredictably if not closely monitored.

- Data Security and Compliance: While cloud platforms provide security features, the organization doesn't have full control over the infrastructure. For industries with strict regulatory compliance, entrusting sensitive data to third-party cloud providers may be challenging.

- Latency: For tasks that require extremely low latency, reliance on external networks and geographically distant data centers can introduce delays, which may not be ideal for real-time applications like financial trading or gaming.
- Vendor Lock-In: Once integrated into a specific cloud ecosystem (e.g., AWS or Azure), migrating to a different platform can be technically challenging and costly due to proprietary services and APIs.

Tradeoffs:

- CapEx vs. OpEx: Cloud computing shifts costs from CapEx (hardware purchases) to OpEx (ongoing usage fees), making it easier to scale without huge upfront investments.
- Flexibility vs. Control: The cloud provides greater flexibility and scalability, but at the cost of reduced control over the underlying infrastructure and security settings.
- Pay-as-You-Go vs. Long-Term Predictability: Pay-as-you-go pricing can be beneficial for fluctuating workloads but may lead to higher expenses in the long term if workloads are steady and large-scale.

Edge compute platforms Edge computing platform is a decentralized computing model where data processing and computing tasks are performed closer to the data source or the end user, rather than relying solely on centralized cloud or data center infrastructures. The "edge" refers to the location where computing happens—at the "edge" of the network, near devices like sensors, smartphones, or IoT devices. This reduces the amount of data sent to a central server, thereby minimizing latency, bandwidth usage, and response times, which is essential for applications that require real-time decision-making.

Advantages:

- Reduced Latency: Since processing occurs near the data source, edge computing significantly reduces latency, making it ideal for real-time applications (e.g., autonomous vehicles, industrial automation, healthcare).
- Bandwidth Efficiency: By processing data locally, edge platforms minimize the amount of data sent to the cloud or central data centers, which is beneficial for areas with limited network connectivity or for devices with high data output (e.g., video cameras).
- Enhanced Privacy: Sensitive data can be processed at the edge without sending it to the cloud, which can help meet privacy regulations and reduce the risk of exposure during data transmission.

Disadvantages:

- Limited Compute Power: Edge devices often have limited processing power compared to centralized cloud servers. They may struggle to handle complex tasks that require high computational capacity.

- Distributed Management Complexity: Managing a large number of distributed edge devices can be challenging. Ensuring all devices are up-to-date, secure, and functioning correctly requires significant infrastructure.
- Security Risks: Edge devices, which are often placed in less controlled environments (e.g., in factories, vehicles, or public spaces), are more vulnerable to physical tampering or cyberattacks.

Tradeoffs:

- Latency vs. Compute Power: Edge computing reduces latency but sacrifices some processing power compared to centralized platforms. For applications needing high performance and real-time responses, this is a worthy tradeoff.
- Centralization vs. Distribution: While edge computing reduces reliance on central servers, it introduces complexity in managing multiple distributed devices.
- Privacy vs. Complexity: Processing data locally improves privacy, but at the cost of increased management and security risks across numerous edge devices.

Distributed Compute Platforms

These spread computational tasks across multiple machines, either on-premise or cloud-based, to perform large-scale data processing. E.g. Apache Hadoop, Kubernetes, serverless platforms like AWS lambda

Advantages:

- Scalability: Distributed platforms allow organizations to scale computational tasks across many nodes, handling massive datasets or workloads (e.g., big data processing, machine learning models).
- Fault Tolerance: If one node fails, others can take over the workload, improving the reliability and availability of the system.
- Cost Efficiency: Distributed platforms can use commodity hardware or cloud resources, allowing for more cost-effective scaling compared to relying on large, monolithic systems.

Disadvantages:

- Complexity: Managing a distributed system involves significant complexity in coordinating tasks across multiple nodes, ensuring data consistency, and handling node failures.
- Latency: Although distributed platforms can improve performance by parallelizing tasks, there may still be delays in communication between nodes, especially if they are geographically dispersed.

- Network Dependencies: Distributed platforms often rely heavily on network performance, and slow or unreliable networks can degrade overall system performance.

Tradeoffs:

- Fault Tolerance vs. Latency: Distributed platforms offer better fault tolerance but can introduce latency and overhead in communication between nodes.
- Performance vs. Complexity: Distributed computing offers the potential for higher performance by spreading tasks across multiple machines, but the added complexity in managing a distributed system requires advanced skills and tools.
- Scalability vs. Simplicity: While distributed systems are scalable, they introduce challenges in maintaining consistency, reliability, and ease of management.

2.2.3 Doge as an edge compute platform

The Dogechain is central to the Strong Doge ecosystem, serving as the primary infrastructure for rewarding computation across distributed nodes and clusters. This platform enables a tokenized approach, whereby nodes, clusters, and individual users receive cryptocurrency incentives for their contributions to edge computing tasks.

Rewards and Task Completion on Dogechain

In this ecosystem, whenever a node, cluster, or user completes a major task or inference process, the Strong Doge platform generates rewards in cryptocurrency, offering transparent and secure compensation. The Dogechain plays a crucial role here, as it tracks and verifies task completions across distributed nodes in real-time. This design ensures that contributions are logged immutably, enabling fair and traceable reward distribution.

Key aspects of this reward system include:

- Transparent Contribution Tracking: The Dogechain logs completed tasks at each node, maintaining a record of contributions that is both public and unchangeable, fostering accountability.
- Immediate Coin Compensation: Upon completion of a major computation, the contributing entity (node, cluster, or user) receives coins directly on the Dogechain, encouraging real-time contributions and further computation within the network.
- Scalability Through Decentralization: By using Dogechain, Strong Doge leverages a decentralized, scalable network to support growing demand for computation. This approach reduces reliance on a central authority and allows the network to grow organically by adding more nodes as needed.

Through this edge computing platform, Strong Doge aligns incentives across its participants, rewarding each with tangible assets in a secure, decentralized manner. This setup not only enhances the platform's efficiency but also incentivizes sustained network participation and scalability.

2.3 Inference Sharding

2.3.1 Utilization of Exo Labs

Overview of Exo Generative AI, including models for image and video generation like Midjourney and DALL-E 2 (based on diffusion models) or language models such as openAI-o1-preview, Gemini-1.5-pro, and Claude-3.5-sonet (based on transformer models), is being rapidly advanced by major tech companies. These models are vastly more complex than traditional ones, often containing hundreds of billions or even trillions of parameters.This makes it difficult for small and medium-sized businesses or individual users to run these models effectively.

Exo addresses this challenge by enabling the use of GPUs from various devices, such as Linux and macOS computers, as well as mobile devices running Android and iOS. Through Exo, these smaller GPUs are combined into a single, larger virtual GPU with a VRAM capacity equivalent to the sum of all the GPUs involved. This allows users to run large-scale AI models without needing to invest in costly, high-end GPUs.

Main Components of Exo **Device Discovery and Connection** Exo establishes peer-to-peer connections between machines, with gRPC as its default protocol.

The gRPC (Google Remote Procedure Call) library/protocol is based on HTTP/2
Advantage

- Facilitates device connections and remote function calls, making it ideal for testing and distributed inference tasks.
- Ensures sequential data transmission with correct packet order and integrity.
- TLS integration provides data security.
- Uses protobuf for encoding, reducing data size during transmission and improving efficiency.

Disadvantages

- Protobuf encoding requires device resources.
- Configuration is complex, potentially increasing deployment and maintenance time.
- Relies on a stable network for accuracy.

The Tailscale library creates a virtual private network (VPN) using the WireGuard protocol

Advantage

- Provides security independent of the network.
- Easy to configure and ensures data integrity.

Disadvantages

- Performance depends on network bandwidth and stability.
- Encryption overhead makes it unsuitable for real-time applications.

UDP Protocol

Advantage

- High performance and low latency, ideal for real-time inference data transmission.
- Consumes fewer system resources as it doesn't require encryption or packet acknowledgment.
- Simple deployment with no complex configuration.

Disadvantages

- Lacks security and data integrity guarantees, with the potential for packet loss.

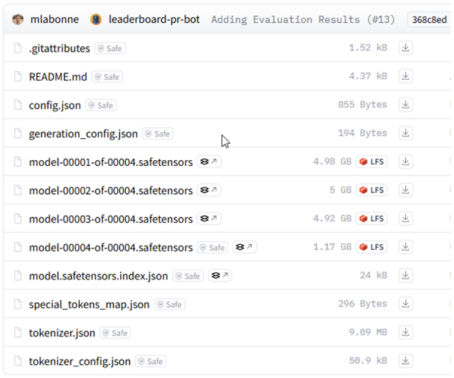
Exo's Node Division Mechanism (A node here is understood as a device, each device can have multiple GPUs)

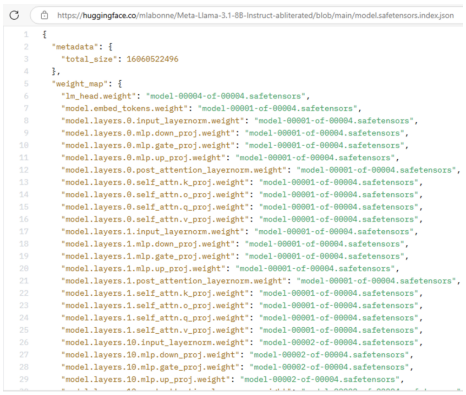
Exo supports only one GPU per node, even on machines with multiple GPUs. This limitation may stem from two factors: first, data processing and loading are handled through RAM before being transferred to the GPU, a challenge yet to be fully addressed. Second, despite using separate virtual environments, some core deep learning library runtimes still require shared resources, leading to conflicts. However, users can specify which GPU participates in the system, making that GPU the node.

Exo's Distributed Inference

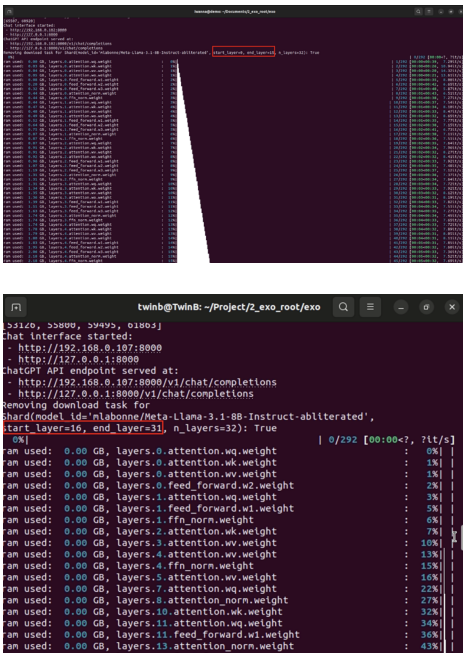
Exo uses Model Parallelism to run models, specifically employing the sharding method after training. Common file formats for model sharding include ".safetensors" (a highly secure format that contains only model weights, with no code) and .bin (a more flexible format that can include additional data and code, though it is slower to load and less secure than safetensors). Shard files are typically named in a format like `model-name_**-of-**.safetensors` or `model-name_**-of-**.bin`, with layer location details stored in accompanying JSON files such as `model-name.safetensors.index.json` or `model-name.bin.index.json`.

Information about the layers within the shards will be contained in JSON files such as `model-name.safetensors.index.json` or `model-name.bin.index.json`.


mlabonne leaderboard-pr-bot Adding Evaluation Results (#13) 368c8ed
.gitattributes Safe 1.52 kB
README.md Safe 4.37 kB
config.json Safe 855 Bytes
generation_config.json Safe 194 Bytes
model-00001-of-00004.safetensors 4.98 GB LFS
model-00002-of-00004.safetensors 5 GB LFS
model-00003-of-00004.safetensors 4.92 GB LFS
model-00004-of-00004.safetensors Safe 1.17 GB LFS
model.safetensors.index.json Safe 24 kB
special_tokens_map.json Safe 296 Bytes
tokenizer.json Safe 9.09 MB
tokenizer_config.json Safe 50.9 kB


https://huggingface.co/mlabonne/Meta-Llama-3.1-8B-Instruct-abliterated/blob/main/model.safetensors.index.json
{
"metadata": {
"total_size": 16060522496
},
"weight_map": {
"lm_head.weight": "model-00004-of-00004.safetensors",
"model.embed_tokens.weight": "model-00001-of-00004.safetensors",
"model.layers.0.input_layernorm.weight": "model-00001-of-00004.safetensors",
"model.layers.0.mlp.down_proj.weight": "model-00001-of-00004.safetensors",
"model.layers.0.mlp.gate_proj.weight": "model-00001-of-00004.safetensors",
"model.layers.0.mlp.up_proj.weight": "model-00001-of-00004.safetensors",
"model.layers.0.post_attention_layernorm.weight": "model-00001-of-00004.safetensors",
"model.layers.0.self_attn.k_proj.weight": "model-00001-of-00004.safetensors",
"model.layers.0.self_attn.o_proj.weight": "model-00001-of-00004.safetensors",
"model.layers.0.self_attn.q_proj.weight": "model-00001-of-00004.safetensors",
"model.layers.0.self_attn.v_proj.weight": "model-00001-of-00004.safetensors",
"model.layers.1.input_layernorm.weight": "model-00001-of-00004.safetensors",
"model.layers.1.mlp.down_proj.weight": "model-00001-of-00004.safetensors",
"model.layers.1.mlp.gate_proj.weight": "model-00001-of-00004.safetensors",
"model.layers.1.mlp.up_proj.weight": "model-00001-of-00004.safetensors",
"model.layers.1.post_attention_layernorm.weight": "model-00001-of-00004.safetensors",
"model.layers.1.self_attn.k_proj.weight": "model-00001-of-00004.safetensors",
"model.layers.1.self_attn.o_proj.weight": "model-00001-of-00004.safetensors",
"model.layers.1.self_attn.q_proj.weight": "model-00001-of-00004.safetensors",
"model.layers.1.self_attn.v_proj.weight": "model-00001-of-00004.safetensors",
"model.layers.10.input_layernorm.weight": "model-00002-of-00004.safetensors",
"model.layers.10.mlp.down_proj.weight": "model-00002-of-00004.safetensors",
"model.layers.10.mlp.gate_proj.weight": "model-00002-of-00004.safetensors",
"model.layers.10.mlp.up_proj.weight": "model-00002-of-00004.safetensors",


Exo Cluster (2 nodes)
Web Chat URL (tinychat): http://192.168.0.102:8000
ChatGPT API endpoint: http://192.168.0.102:8000/v1/chat/completions
GPU poor
GPU rich
35.58 TFLOPS
Linux Box (NVIDIA GE
CE RTX 2060) 12GB
22.8TFLOPS
[0.52-1.0B]
12.9TFLOPS
[8.00-0.52]



Using the layer location information, Exo only downloads the necessary shards, loading the corresponding layer weights into the model.

For example: 2 shards 01 and 02 containing layers 0 to 15 will be loaded onto machine 1, while 2 shards 03 and 04 containing layers 16 to 31 will be loaded onto machine 2.

When an inference request is sent to the cluster consisting of the two nodes above, the data flow will run sequentially in a circle through machine 1 (layers 0-15), then transfer the results from machine 1 to machine 2 (layers 16-31). The final result will be received and displayed on the chat screen of machine 1.

To manage model loading, Exo follows a three-step process:

- Create a model with random weights.
- Load weight from the `state_dict` file
- Load new weights into the model.

This process currently doubles RAM and VRAM usage. To mitigate this, Exo suggests creating the model with a virtual coefficient (e.g., a meta tensor data-type) in step 1, and only performing steps 2 and 3 after. Additionally, recalculating the size of each layer (in MB or GB) will allow for more efficient workload distribution across nodes, instead of simply dividing by layer count.

The processing power of each node can be assessed using its TFLOPS (calculated automatically via benchmarking), which helps ensure optimal workload distribution.

```
# collect metadata for each tensor
for name in tensor_names:
  tensor_data = f.get_tensor(name)
  shape = tensor_data.shape
  dtype = tensor_data.dtype

  # calculate the tensor size in bytes based on dtype
  total_elements = 1
  for dim in shape:
    total_elements *= dim

  if dtype == torch.float32:
    element_size = 4
  elif dtype == torch.float16 or dtype == torch.bfloat16:
    element_size = 2
  # extend this to support more data types if needed
  else:
    raise ValueError(f"unsupported dtype: {dtype}")

  tensor_size = total_elements * element_size
```

```
def _benchmark_tflops(n, dtype='f32', num_iterations=100):
    backend = None
    if torch.backends.mps.is_available():
        backend = torch.mps
        device = torch.device("mps")
    elif torch.cuda.is_available():
        backend = torch.cuda
        device = torch.device("cuda")
    else:
        device = torch.device("cpu")
    if dtype == 'f32':
        A = torch.randn((n, n), device=device, dtype=torch.float32)
        B = torch.randn((n, n), device=device, dtype=torch.float32)
    elif dtype == 'f16':
        A = torch.randn((n, n), device=device, dtype=torch.float16)
        B = torch.randn((n, n), device=device, dtype=torch.float16)
    elif dtype == 'int8':
        A = (torch.randint(-128, 127, (n, n), device=device, dtype=torch.int8)).float()
        B = (torch.randint(-128, 127, (n, n), device=device, dtype=torch.int8)).float()
    else:
        raise ValueError("Unsupported data type. Use 'f32', 'f16', or 'int8',")
    if backend:
        start_time = time.perf_counter()
        for _ in range(num_iterations):
            C = torch.mm(A, B)
            backend.synchronize()
        elapsed_time = time.perf_counter() - start_time
    else:
        start_time = time.perf_counter()
        for _ in range(num_iterations):
            C = torch.mm(A, B)
        elapsed_time = time.perf_counter() - start_time
    flops_per_iteration = 2 * (n ** 3)
    total_flops = flops_per_iteration * num_iterations
    tflops = (total_flops / elapsed_time) / 1e12
    return float(f"{tflops:.2f}")

def benchmark():
    fp32=_benchmark_tflops(2048)
    fp16=_benchmark_tflops(2048, dtype='f16')
    int8=_benchmark_tflops(2048, dtype='int8')
    return (fp32, fp16, int8)
```

The number of nodes in a cluster must also be balanced—too few nodes may prevent the model from being loaded, while too many can cause latency due to excessive data movement. In case of node failure, the failed node must be replaced with one containing the necessary shard, or the shard must be downloaded again.

By assigning specific roles to each node in a cluster, throughput can be increased. Once a node finishes processing its part of one inference request, it can begin handling another request, enhancing efficiency.

2.3.2 Network Scaling

Peer-to-peer with STUN STUN (Session Traversal Utilities for NAT) is a network protocol that allows devices behind a NAT (Network Address Translation) to discover their public IP address and port number assigned by the NAT. This information is then used to establish direct connections with other devices on the internet.

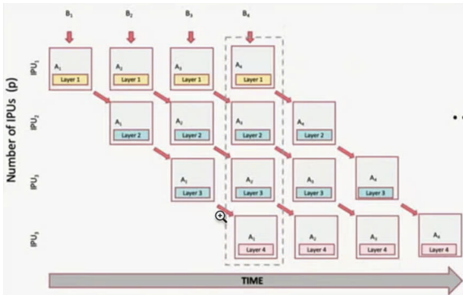
In simpler terms: STUN acts like a "mirror" that reflects your IP address and port on the internet. When you are behind a NAT, you don't know your public address. STUN tells you that.

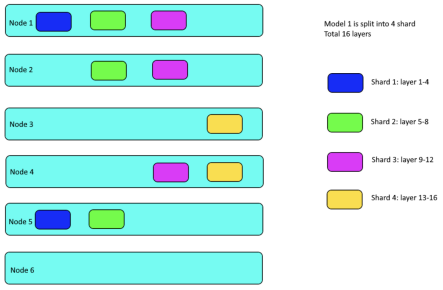
Let's assume there are two nodes behind NATs that need to connect to each other over the internet. Here's how they would use STUN:

- Node 1 sends a STUN request to a STUN Server on the internet.
- The STUN Server receives the request and replies to Node 1 with its public IP address and port number assigned by the NAT Router.
- Simultaneously, Node 2 also sends a request to the STUN Server and receives its own public IP and port.
- Once both Node 1 and Node 2 know each other's public IP addresses and ports, they can establish a direct connection, bypassing the NAT Routers.

Network Building The idea of building a network where each machine in the system is a node, grouping these nodes together, and using them as to inference a Model is called a cluster. The network operates based on a pipeline parallelism mechanism. (Reference in "Exo")

To better illustrate the concept of pipeline parallelism in the system: The system consists of multiple clusters, each running a model. The model is divided into smaller parts called shards, and each shard contains different layers of the model. Nodes within a cluster are responsible for processing certain layers of that model. In a cluster, these nodes are connected in a sequential chain, referred to as a pipeline. When a node completes processing its assigned layers, the result is passed to the next node. At this point, the completed node no longer needs to track the subsequent processing of the result and can immediately take on a new processing task from the system. This approach forms pipeline parallelism, where multiple requests can be processed simultaneously,





thus increasing throughput (the number of inferences the cluster can handle within a given time period). An extension of this is that a node is not limited to receiving requests only from the cluster it belongs to; it can also process requests from other clusters, as long as it correctly processes the layers it has been assigned within the model. This increases the system's flexibility and optimizes resource allocation. [2]

The network needs a statistics table about the types of models supported, as well as information about shards, shard size, number of shards, resource requirements for each shard, and pricing, so that users joining the system can plan accordingly.

Based on ideas developed from Exo, assume that the system needs to build a cluster to infer Model 1 (this model has 4 shards and a total of 16 layers). The layers within each shard are illustrated in the figure below. Currently, there are 6 nodes registered in the system, and in the downloaded Model 1 directory on each node, nodes 1 through 5 already have some shards, while node 6 has none.

- Configuration 1 for the cluster: Use layers 1-4 from node 1, layers 5-8 from node 5, layers 9-12 from node 4, and layers 13-16 from node 3.
- Configuration 2 for the cluster: Use layers 1-8 from node 1 (since the GPU on node 1 is powerful enough to load more layers), layers 9-12 from node 4, and layers 13-16 from node 3.
- Configuration 3 for the cluster: Use layers 1-6 from node 1, layers 7-8 from node 2 (since the GPU on node 2 is weaker and can only load 2 layers), layers 9-12 from node 4, and layers 13-16 from node 3.

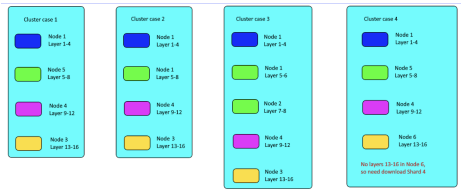


Figure 4: Using the collected information, the system can propose several configurations for the cluster to infer Model 1, as shown in the figure.

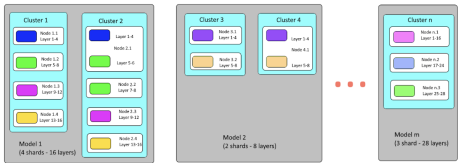


Figure 5: Illustration of the network with n Clusters and m Models

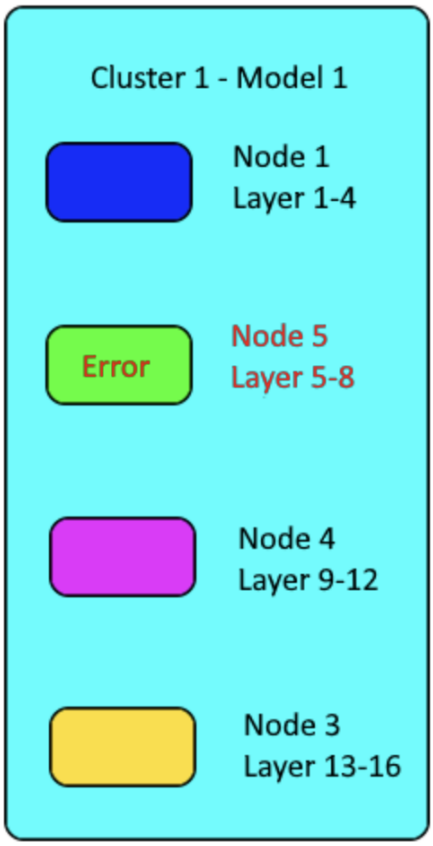
- Configuration 4 for the cluster: Use layers 1-4 from node 1, layers 5-8 from node 5, layers 9-12 from node 4, and since node 6 does not have layers 13-16, these will need to be downloaded (this should be avoided as it will take time to download).

With the 4 proposed configurations mentioned above, Configuration 1 should be used when the TFLOPS values, representing the computational power of the nodes in the cluster, are relatively equal. Configuration 2 is used when the TFLOPS and VRAM of Node 1 are sufficient to load and process the first 8 layers. Configuration 3 is applicable when the TFLOPS and VRAM of Node 1 can handle the first 6 layers, while Node 2, with lower TFLOPS and VRAM, can only process 2 layers. Configurations 1, 2, and 3 should be prioritized for building the cluster, as they make optimal use of the available resources on each node. Configuration 4, however, should be avoided because it will require extra time to download the appropriate shard for the empty Node 6.

Let's assume in the case where the cluster running Model 1 encounters an error and node 5 (containing layers 5-8) is disconnected. In that case, the cluster needs to use nodes with high TFLOPS and VRAM already available in the cluster, or replace them with one or more suitable nodes. This will require time to set up new connections and may even take additional time to load the appropriate shard on the new node.

The cluster can resolve this in several ways.

Edge Node performance Verification To evaluate the processing performance of each node, we can rely on the TFLOPS value with operations on 32-bit floating-point numbers (FP32), 16-bit floating-point numbers (FP16), or 8-bit signed integers (int8).


Cluster 1 - Model 1
Node 1
Layer 1-4
Error
Node 5
Layer 5-8
Node 4
Layer 9-12
Node 3
Layer 13-16

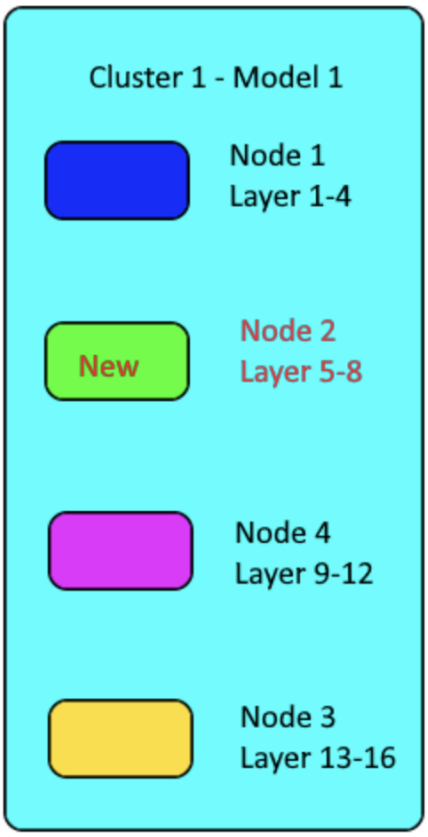


Figure 6: Replace layers 5-8 of Node 5 with layers 5-8 from Node 2. The best-choice is that Node 2, used as a replacement, has a TFLOPS computing power similar to Node 5 and enough VRAM to load layers 5-8.

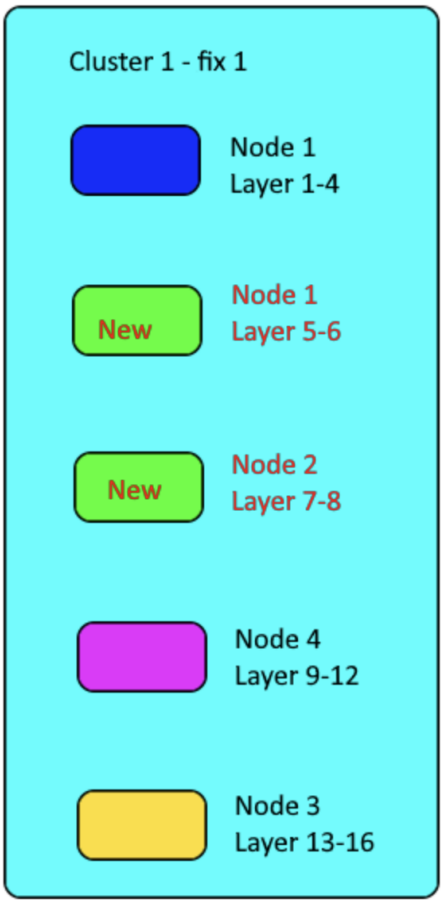


Figure 7: Replace layers 5-8 of node 5 with layers 5-6 from node 1 and layers 7-8 from node 2. It should be used if Node 1 has sufficiently high TFLOPS and enough VRAM to load additional layers 5-6. Ideally, if Node 1 can handle all layers 1-8, then Node 2 won't be needed, that would be the best option; otherwise, we need to use the weaker Node 2 to process layers 7-8.

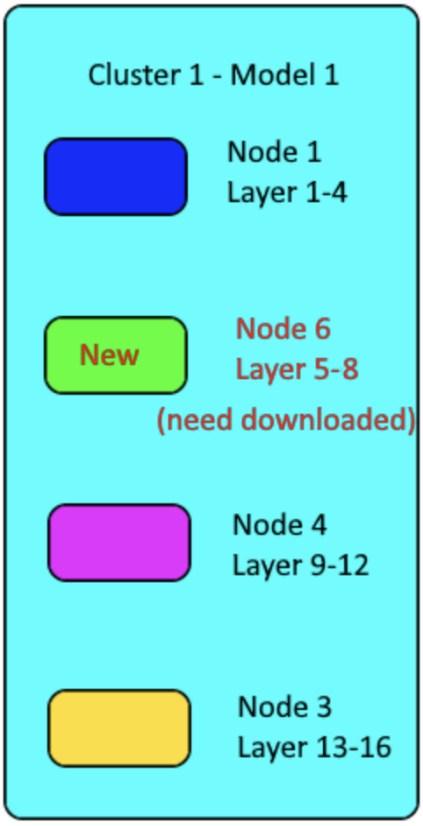
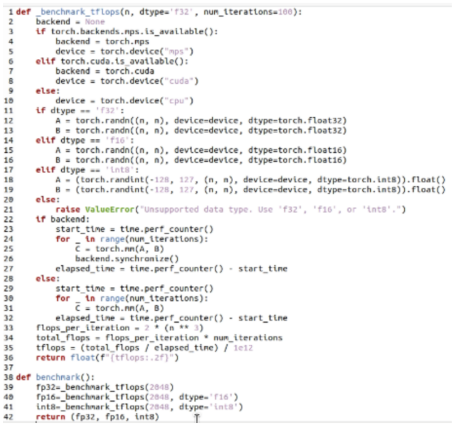


Figure 8: Replace them by downloading and loading new shards from node 6 (if no suitable node with the required shards exists). This case should be avoided because loading a shard can take a long time.

```
CHIP_FLOPS = {
  # Source: https://www.cpu-monkey.com
  # Note: currently no distinction between variants of M3 Max and M3 Pro, we pick the lower one to be
  ### M chips
  "Apple M1": DeviceFlops(fp32=2.29*TFLOPS, fp16=4.58*TFLOPS, int8=9.16*TFLOPS),
  "Apple M1 Pro": DeviceFlops(fp32=5.30*TFLOPS, fp16=10.60*TFLOPS, int8=21.20*TFLOPS),
  "Apple M1 Max": DeviceFlops(fp32=10.60*TFLOPS, fp16=21.20*TFLOPS, int8=42.40*TFLOPS),
  "Apple M1 Ultra": DeviceFlops(fp32=21.20*TFLOPS, fp16=42.40*TFLOPS, int8=84.80*TFLOPS),
  "Apple M2": DeviceFlops(fp32=3.55*TFLOPS, fp16=7.10*TFLOPS, int8=14.20*TFLOPS),
  "Apple M2 Pro": DeviceFlops(fp32=5.68*TFLOPS, fp16=11.36*TFLOPS, int8=22.72*TFLOPS),
  "Apple M2 Max": DeviceFlops(fp32=13.49*TFLOPS, fp16=26.98*TFLOPS, int8=53.96*TFLOPS),
  "Apple M2 Ultra": DeviceFlops(fp32=26.98*TFLOPS, fp16=53.96*TFLOPS, int8=107.92*TFLOPS),
  "Apple M3": DeviceFlops(fp32=3.55*TFLOPS, fp16=7.10*TFLOPS, int8=14.20*TFLOPS),
  "Apple M3 Max": DeviceFlops(fp32=14.20*TFLOPS, fp16=28.40*TFLOPS, int8=56.80*TFLOPS),
  "Apple M3 Pro": DeviceFlops(fp32=4.97*TFLOPS, fp16=9.94*TFLOPS, int8=19.88*TFLOPS),
  "Apple M4": DeviceFlops(fp32=3.55*TFLOPS, fp16=7.10*TFLOPS, int8=14.20*TFLOPS),
  ### A chips
  "Apple A13 Bionic": DeviceFlops(fp32=0.69*TFLOPS, fp16=1.38*TFLOPS, int8=2.76*TFLOPS),
  "Apple A14 Bionic": DeviceFlops(fp32=0.75*TFLOPS, fp16=1.50*TFLOPS, int8=3.00*TFLOPS),
  "Apple A15 Bionic": DeviceFlops(fp32=1.37*TFLOPS, fp16=2.74*TFLOPS, int8=5.48*TFLOPS),
  "Apple A16 Bionic": DeviceFlops(fp32=1.79*TFLOPS, fp16=3.58*TFLOPS, int8=7.16*TFLOPS),
  "Apple A17 Pro": DeviceFlops(fp32=2.15*TFLOPS, fp16=4.30*TFLOPS, int8=8.60*TFLOPS),
  ### NVIDIA GPUs
  # RTX 40 series
  "NVIDIA GEFORCE RTX 4090": DeviceFlops(fp32=82.58*TFLOPS, fp16=165.16*TFLOPS, int8=330.32*TFLOPS),
```

Figure 9: Method 1: Identify the type of card used by the node, and then look it up in a dictionary file, for example



```python
def _benchmark_tflops(n, dtype='f32', num_iterations=100):
    backend = None
    if torch.backends.mps.is_available():
        backend = torch.mps
        device = torch.device("mps")
    elif torch.cuda.is_available():
        backend = torch.cuda
        device = torch.device("cuda")
    else:
        device = torch.device("cpu")
    if dtype == 'f32':
        A = torch.randn((n, n), device=device, dtype=torch.float32)
        B = torch.randn((n, n), device=device, dtype=torch.float32)
    elif dtype == 'f16':
        A = torch.randn((n, n), device=device, dtype=torch.float16)
        B = torch.randn((n, n), device=device, dtype=torch.float16)
    elif dtype == 'int8':
        A = (torch.randint(-128, 127, (n, n), device=device, dtype=torch.int8)).float()
        B = (torch.randint(-128, 127, (n, n), device=device, dtype=torch.int8)).float()
    else:
        raise ValueError("Unsupported data type. Use 'f32', 'f16', or 'int8'.")
    if backend:
        start_time = time.perf_counter()
        for _ in range(num_iterations):
            C = torch.mm(A, B)
            backend.synchronize()
        elapsed_time = time.perf_counter() - start_time
    else:
        start_time = time.perf_counter()
        for _ in range(num_iterations):
            C = torch.mm(A, B)
        elapsed_time = time.perf_counter() - start_time
    flops_per_iteration = 2 * (n ** 3)
    total_flops = flops_per_iteration * num_iterations
    tflops = (total_flops / elapsed_time) / 1e12
    return float(f"{tflops:.2f}")

def benchmark():
    fp32=_benchmark_tflops(2048)
    fp16=_benchmark_tflops(2048, dtype='f16')
    int8=_benchmark_tflops(2048, dtype='int8')
    return (fp32, fp16, int8)
```

Figure 10: Method 2: Use an automatic benchmarking function.

Cluster performance in the network Verification Evaluating the processing performance of a cluster within the network means determining the processing efficiency of the model running in that cluster.

We can evaluate this based on several criteria:

- Latency: Test by entering a sample chat message, then measure the time from when the token enters layer 1 on the first node to when the first result token is produced at the last layer on the final node. `Tlatency=tfirst-token-out-ttoken-in`
- Data transfer time: The time it takes to transfer data between consecutive nodes can also be tested by entering a sample chat message.
- Load balancing: The load value is measured by the ratio between the total size of the layers loaded onto a node (see III.1.ii) and the computational capacity in TFLOPS of the node (III.3). The purpose of this load balancing value is to determine whether the workload (total weights) assigned by the system to the nodes in a cluster is balanced in terms of processing capacity in TFLOPS.
- Connection verification: Check connectivity by pinging the node or between nodes.
- There should also be statistics on the Success Rate of Requests sent to a node, so that when the node participates in the network, a proper weighting can be applied.

User Inference Process A user who wants to use the system to run a Generative AI model will first select from the available models that the system currently supports (such as generative AI models for image generation, video, or large language models). They then choose a cluster that is available on the

system for that model, or if a suitable cluster is not available, time will be needed to create the cluster and download the model. The user can choose options that fit their budget, and they can also check the performance of the cluster they select. Finally, they can enjoy the results.

In the future, support could be expanded to include models available on Hugging Face or from the user's own sources, as long as they have the model architecture file ready.

2.4 Privacy

2.4.1 Today's All Seeing World

With centralized AI systems, especially in the case of generative language models like ChatGPT, Claude, or Gemini, user data and conversation history are often recorded. Over time, these centralized AI systems become increasingly aware of users' behaviors and can develop detailed user profiles covering personal preferences, habits, and even sensitive information like health status and financial details. This data could potentially be used for undesirable purposes, such as targeted advertising, surveillance, or even selling the information to third parties.

The vast amount of information held by these centralized AI systems raises concerns about user privacy, while also making users increasingly dependent on these systems.

A carefully designed distributed inference system can help mitigate these risks by encrypting information (such as chat data) as soon as it enters the system using text tokenization (turning text into tokens). Additional encoding methods can also be applied to secure data transmission. In such a system, each node in a cluster only has access to its specific task and does not know the full scope of the work, thereby enhancing privacy and security.

2.4.2 Inference Layer

The inference layer is where the inference process takes place, and it is the most vulnerable point in terms of privacy. This is where user conversations or private data are sent and processed.

Although these conversations are called private, they are actually not private at all because the inference layer is managed by centralized companies, meaning that all conversation content is sent to these companies.

In a distributed system, encrypting data at the source, as well as ensuring that each node only knows its own part of the task, can completely avoid this issue.

2.4.3 Inference Provenance

In centralized systems, all information and inference results are monitored and stored by a central organization. This makes it possible to trace the origin of inferences and use data to find user profiles and their entire interaction history.

$$d_{AB} = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2}$$

Figure 11: The distance used here is the standard 2D Euclidean distance

$$W_i = W \times \frac{TFLOPS_i}{\sum_{k=1}^{n} TFLOPS_k}$$

Figure 12: The load balancing formula.

In contrast, if a system is distributed with models restructured using sharding, the inference process is divided and completed across nodes containing separate shards. Tracking the origin of queries becomes difficult or even impossible. This distributed Strong Dogeing ensures that no one, not even the participating nodes, can link the information they have to the user, thus providing comprehensive privacy protection.

2.5 Future Work

2.5.1 Network Latency Optimization

In a sharding-based distributed system, to optimize network latency, specifically cluster latency for each inference, we can use the following solutions:

- Clustering based on geographic location: For example, if a query originates from the USA, we should use or set up a cluster with nodes located in or near the USA. Using the K-means clustering algorithm, we can select the nearest cluster for the inference. This approach can improve the speed of data transmission between nodes and enhance network stability.
- Load balancing to improve the processing capability of each node. The processing speed of each node depends on two main factors: the assigned workload and the computational capacity (TFLOPS). Suppose the total weight of the model is W, Wi is the portion of weight assigned to node i, TFLOPSi is the computational power of node i, and n is the number of nodes in the cluster.
- Optimization problem for network load balancing: At any network state, there is always a need to create a new configuration of m clusters to run n different models. Thus, it's essential to solve a Combinatorial Optimization Problem, balancing the load from selecting nodes for each cluster to

distributing workloads among them. This is a complex problem and requires a suitable heuristic algorithm to solve it (e.g., Genetic Algorithm, Tabu Search, Ant Colony Optimization, etc. combined with some evaluations).

2.5.2 Distributed Work Sharing Currency

AI models are continuously evolving, and each model performs different tasks. For example, in the field of images, there are models for recognizing and classifying images, as well as generating images. In the language field, there are models for language generation, and similarly, there are various models in the field of sound. These models have even progressed towards a multi-modal approach, meaning they can handle different types of tasks simultaneously. Therefore, in the future, when a user makes a request, it may require the collaboration of many different AI models to provide a response.

When AI models complete their tasks, they will receive rewards. The simplest way to do this might be through cryptocurrency owned by the distributed platform. This allows each cluster and the nodes within that cluster to receive rewards proportional to the work they have done, thus promoting transparency and fairness in sharing computational resources among the nodes.

2.5.3 Enduring Strong Doge State

In centralized inference methods, the state of the Strong Doge is often limited to a few inferences or queries, which means it can only perform specific tasks without accumulating continuous experience. After each task, the Strong Doge's state is typically "reset", restricting its ability to learn and grow.

In contrast, decentralized inference allows the model's state to exist continuously, enabling the Strong Doge to learn and accumulate experience over time. Different models can also interact with each other. Each new interaction with real data becomes part of the collective knowledge, helping the Strong Doge improve constantly. This setup allows AI Strong Doges to develop superior intelligence by continuously learning from their environment and sharing knowledge with other Strong Doges in the network.

This model sounds very "futuristic" because it opens up the potential for AI Strong Doges to evolve and grow without being limited by individual tasks. They can become continuous learning entities, adapting to complex environments and accumulating experience to become increasingly smarter.

2.6 Conclusion

Strong Doge is at the forefront of a transformative approach to AI infrastructure, where decentralization, privacy, and community governance form the foundation. By integrating Exo as its core, Strong Doge harnesses the power of distributed GPUs, establishing peer-to-peer connections that allow widespread access to AI models on decentralized networks. This structure not only scales

computational capacity but democratizes access, enabling a far broader range of devices to participate in complex AI tasks that were once exclusive to centralized corporate infrastructures.

With Exo's Distributed Inference capabilities, Strong Doge transforms edge computation, enabling a diverse set of devices to access decentralized models and provide extensive functionality that centralized models cannot achieve. Dogechain plays a key role in managing shard distribution, validating tasks, and offering transparent rewards, ensuring that all contributions—whether from nodes, clusters, or individual users—are fairly compensated and securely recorded on the blockchain.

Through this unique architecture, Strong Doge envisions a future where AI systems are accessible, secure, and controlled by a community of participants rather than a single centralized authority. This approach to open-source, persistent, and resilient AI positions Strong Doge as a leader in building a more equitable AI ecosystem. By providing continuous learning capabilities, enduring model states, and an incentive-driven participation model, Strong Doge sets the stage for an AI-driven world that prioritizes privacy, transparency, and shared ownership.

References

[1] Measure CPU Component - `https://www.mathworks.com/help/parallel-computing/measuring-gpu-performance.html`

[2] Exo: Run your own AI cluster at home by Mohamed Baioumy.

[3] Filecoin: A Decentralized Storage Network, Protocol Labs - `https://filecoin.io/filecoin.pdf`.