Introduction

In the rapidly evolving world of modern technology, the significance of distributed computing has become clear. This field, which had modest origins as the focus of a small subset of computer science research and development, is now poised to impact humanity more in the next few years than in all of computing history combined.

We are at a pivotal moment in history. The emergence of generative AI marks a paradigm shift, reshaping how we utilize technology and interact with our everyday realities. The extraordinary becomes ordinary with each passing day, and we stand at the brink of a new digital renaissance.

Data is the new oil, and as with all resources, it is a double-edged sword. On the one hand, imagine a world where the collective power of individuals, organizations, and their computers across the globe is harnessed to fuel generative AI systems, crafting realities beyond our wildest imaginations, freeing us from mundane labor, increasing agricultural and industrial output while simultaneously reducing environmental footprints, curing diseases, avoiding catastrophes, and a plethora of other seemingly unattainable goals.

However, the future is not necessarily so bright. Humanity is faced with vested interests, some of them companies, some of them governments, and some of them individuals, who would rather concentrate this power themselves. And this only if AI does not become cognizant and takes matter into its own hands.

For this reason, it is necessary to have resilient infrastructures for coordinating distributed computing resources. The base of this is the actual distribution of computing work, which continues to be a developing field, and has resulted in the construction of novel and innovative distributed computing platforms like Bacalhau, a key technology in Lilypad's stack. However, until now, most distributed computing technologies assumed that the entity submitting jobs also controlled the machines on which those jobs were being run. This excluded the possibility of utilizing idle computing resources owned by others, which are a massive, mostly untapped resource, just waiting to be unleashed. Furthermore, since these computing resources are owned by many different individuals and entities, they require a substrate on which they can coordinate and collaborate.

The nature of this problem calls out for the construction of a computational resource marketplace as its solution. Lilypad is precisely this. In its simplest description, it is a two-sided marketplace for compute that is robust, extensible, and capable of coordinating complex computational tasks.

However, Lilypad's applications go far beyond this limited description. Artificial intelligence, zero-knowledge proofs, blockchain bridges, sensor and user data, digital twins, supply chain management, hardware profiling, privacy, compute sharing, and public goods computing are just some of the fields where Lilypad can dramatically shift the economic and computational foundations of our world. At the core of Lilypad lies deep thinking about incentives around verifiability of computational workloads, and cooperation within, and across, multi-stakeholder networks. Lilypad is not just about harnessing idle computational resources; it is about empowering a global community to contribute to and benefit from its collective computational power.

Distributed Computing Background

Distributed computing has become a critical backbone of our technological landscape, driven by the need for scalability, efficiency, and the harnessing of vast computational resources. This necessity arises from the limitations of vertically scaled monolithic systems. The demand for distributed computing is further fueled by the increasing complexity and scale of modern workloads, as well as the distributed nature of a large part of data generation.

Why is distributed computing necessary?

The pivotal reason for embracing distributed computing lies in its inherent scalability and cost-effectiveness. Vertically scaling individual machines provides some gains, but costs can quickly spiral out of control; distributed execution was explored because monolithic systems were hitting real-world limitations.

Distributed execution not only offers superior scalability, but also proves to be more economically viable. Parallelism across distributed systems provides efficiency and redundancy at scale. As data is increasingly generated and stored across networks of edge devices and cloud servers, distributed computing aligns with the natural architecture of modern data distribution.

Further, parallelism across distributed systems provides efficiency and redundancy at scale. Workloads like training machine learning models benefit from parallel execution across clusters of commodity machines. By sharing the memory, disk, and networking throughput across many physical devices, it is possible to get significantly higher aggregate performance than all but the most expensive hardware.

Additionally, data is increasingly generated and stored across networks of edge devices and cloud servers. Moving all this data to a single bucket or data lake consumed an enormous amount of time and bandwidth. It makes more sense to take advantage of the distributed architecture already in place, by processing data locally, where it is generated.

A Short History of Distributed Computing

Technology has been in pursuit of distributed workloads since the invention of the ENIAC in the 1940's – one computer in one room is about as centralized as you can get. Mainframes became minicomputers, minicomputers became microcomputers, and microcomputers became microprocessors wired together. The ARPANet and the advent of a network created the possibility of one job, many machines.

Phase 0: The Move to Commodity Compute

The first moves towards broadly used distributed compute took off in the late 1990s, with the canonical example being the launch of commodity distributed computing with Google.

Previously, the majority of machines and deployments required vertical scaling, where increasing the amount of compute required increasingly higher costs. Beyond cost, vertical scaling often created

single points of failure. If a single machine had a hardware outage (it happens!), or needed rebooting for OS upgrades, or a datacenter went offline, the entire system could fail.

Google, and other commodity compute users, recognized the advantages in cost, performance, and uptime when they incorporated an expectation of failure into their system design. Servers would always fail, so by spreading their services across more servers, there was a greater chance they could stay operational during an outage.

Phase 1 - Virtualization Takes Hold

If adding a single piece of hardware could provide improved uptime and performance, perhaps virtualizing that machine and simulating many machines would yield even further benefits. Enter virtual machines (VMs).

Virtualizing machines enabled separation between the hardware platform and the operating system, and a new level of resource utilization. Applications could now be deployed in a defined and repeatable way to clusters of machines. Due to the virtualization of the machine, each physical machine could be much more tightly packed with applications (including redundant ones to reduce downtime). While virtualization has been around for more than half a century, its mainstream adoption really took off around the early 2000s with the introduction of VMWare ESX and GSX.

Virtualization was a step up from the isolated systems of the previous phase, offering better resource utilization, isolation, and load balancing. A single bad application was less likely to bring down an entire cluster, and, by abstracting away hardware, cleaner portability and configuration was possible.

There were also groundbreaking new technologies such as live migration, which significantly reduced downtime due to server issues and planned upgrades. However, these VMs were still localized, limiting the true potential of distributed computing architectures. Worse, applications were often still restricted to a single machine (virtual or otherwise), limiting the ability to take advantage of the distributed topology.

Phase 2 - Distribution Via Platforms

The MapReduce paper was seminal in rethinking what an application platform could provide to simplify the concepts and utilization of distributed machines. Behind a single interface, developers could now schedule and build distributed workloads at the application level, with a limited need to dive deeper than top level structures. MapReduce inspired popular frameworks like Hadoop and Spark, and enabled massive amounts of data to be processed in parallel across a network of machines.

Platforms like this represented a huge leap in scalability and performance, but even with this level of distribution, computing tasks were still confined to a single data center. They also required adopting a new architecture, new SDKs, and often, entire rewrites of applications, in order to take advantage of the technology.

Phase 3 - It’s In The Clouds

Containerized workloads, popularized by Docker, revolutionized the way applications are developed, deployed, and managed. This revolution has been bolstered by sophisticated orchestration systems

like Kubernetes and Mesosphere, enhancing orchestration and scaling to unprecedented levels. This wave of technologies, traditionally called Cloud Native Compute and rolled up into the Cloud Native Computing Foundation, has profoundly reshaped the landscape of cloud computing.

These cloud-native platforms make the adoption of distributed computing technologies much more straightforward because encapsulation of applications and their dependencies have been built into the packaging. Additionally, with the declarative nature of both containers and cloud-native orchestrators, the ability to abstract jobs from underlying APIs and systems has never been easier. This allows for significantly improved utilization by dynamically moving the application to the environments that are best suited for the application and have the capacity to run it.

However, despite their remarkable capabilities, most distributed computing platforms still operate primarily within a single zone or region. Although they do offer some cross-region support, being based on a centralized platform will always require significant work to support distributed and unreliable networks.

Modern Challenges of Distributed Computing

Despite the significant strides made in distributed computing, modern challenges persist:

- **Reliability:** Loosely coupled components fail unpredictably. Lack of redundancy means few or single points of failure.
- **Scaling Issues:** Optimally placing computations near relevant data is difficult. Adding nodes creates bottlenecks.
- **Data Locality:** Jobs and queries require expensive cross-network data shuffling.
- **Fault Tolerance:** Partial failures can cascade, causing widespread outages.
- **Slow processing:** Distributed data dependencies stall workflows. Lack of parallelism wastes resources.
- **Compliance risks:** Sensitive data spread across systems multiplies security concerns.
- **Latency:** Synchronous calls between distributed modules incur delays.
- **Orchestration:** Deploying updates across disparate systems is complex and risky.

Being truly distributed means re-thinking how an application and platform interacts with the network. Reviewing the Fallacies of Distributed Computing, the fallacies listed then, three decades ago, are as true today as they have ever been:

1. The network is reliable;
2. Latency is zero;
3. Bandwidth is infinite;
4. The network is secure;
5. Topology doesn't change;
6. There is one administrator;
7. Transport cost is zero;
8. The network is homogeneous.

Worker nodes needing to check in with a centralized task manager, unreliable networks, widely varying latency, and challenging networking can exacerbate scheduling problems very quickly. In order to realize the dream of truly distributed computing, a robust platform needs to tackle these challenges head-on.

Bacalhau

Bacalhau is a distributed computing platform designed to address these modern challenges. Bacalhau's tenets are the following:

- **Familiar** - Should feel very similar to existing applications and platforms. If you use Kubernetes, Spark, Docker, Python, etc., then Bacalhau should feel like a familiar friend.
- **Resilient** - Supporting distributed networks means a very different approach to resilience - particularly a first-class support for a network that could disappear at any time. Bacalhau must support these kinds of topologies natively.
- **Efficient** - Applications can take advantage of idle compute, move jobs to where there are optimization opportunities, and observe and respond to dynamic network topologies, all without the user having to take any actions beyond detailing the requirements for their job.
- **Distributed-Native** - The platform should feel both familiar, and represent distributed concepts as first class elements. Targeting datasets, managing permissions, navigating multi-location shards, and many more features should be as easy as building locally.

Bacalhau, builds in the expectation that machines, applications, and networks are unreliable, and as a result yields better scaling and reliability. In the same way that:

- Commodity compute allowed people to go beyond a single large machine;
- VMs allowed people to go beyond a physical cluster;
- Data clusters allowed people to go beyond clusters of VMs;
- And containers allowed people to go beyond a cluster of VMs;

Bacalhau allows scaling beyond a single cloud zone to accommodate where the data and compute are located, and make truly global deployments a reality. In doing so, Bacalhau helps make cross-region compute possible.

Bacalhau is poised to propel distributed computing to new heights by seamlessly integrating into existing workflows, enhancing resilience, optimizing efficiency, ensuring verifiability, and embracing the inherently distributed nature of modern computational networks. As the technological landscape continues to evolve, Bacalhau stands at the forefront, offering a robust answer to the challenges that have long hindered the full realization of distributed computing's potential.

Incentivization with Lilypad

Bacalhau is a robust distributed computing platform, but alone lacks incentives that match those who want to run jobs, and those who are willing to run those jobs for the right price. Lilypad is the incentivization layer built on top of Bacalhau. At its core, the incentivization layer is a marketplace that

connects buyers and sellers, enabling the matching of client requests and compute node offers, clients paying compute nodes, and compute nodes getting paid by clients.

Distributed Computing Marketplaces

Marketplaces play a crucial role in incentivizing the use of latent computing resources by creating a dynamic ecosystem where supply and demand for computational power can meet efficiently. The need for such marketplaces arises from the nature of distributed computing and the challenges associated with harnessing latent resources effectively. The benefits of using marketplaces include:

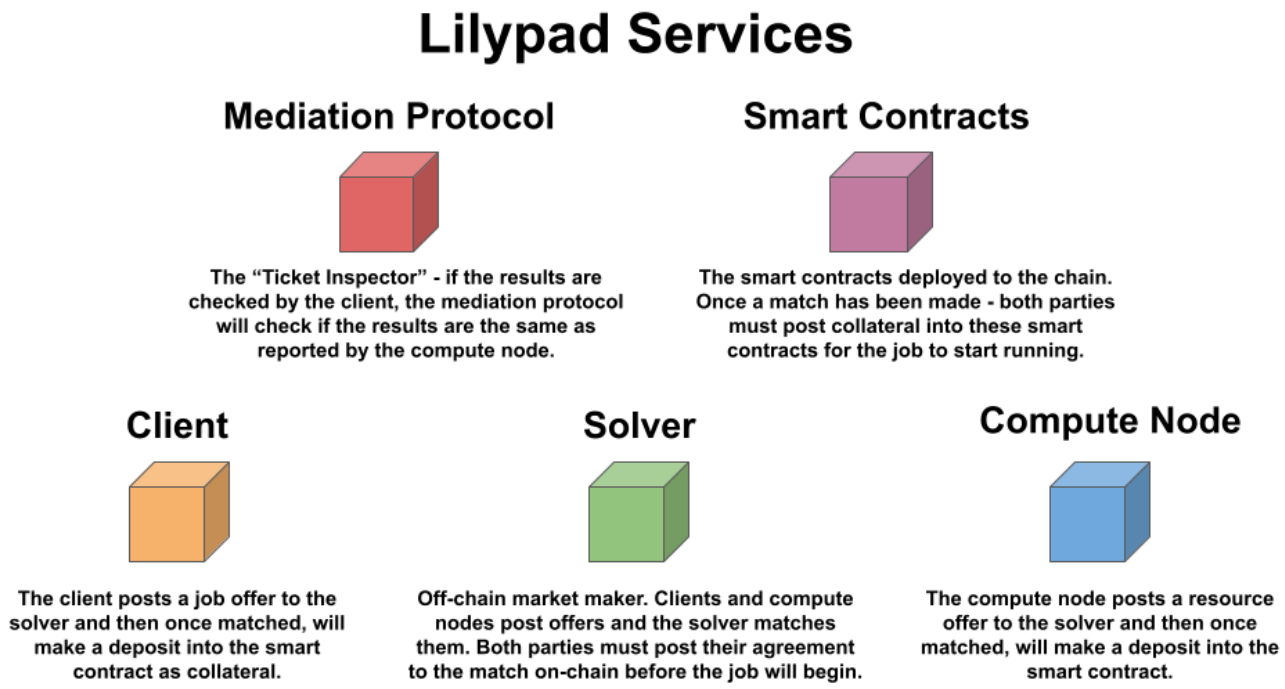
1. **Idle Computing Resource Utilization**
    a. **Market Dynamics:** Individuals or organizations with latent computing resources, such as idle servers, or unused consumer hardware (e.g. when their owners are sleeping), can offer these resources to potential clients.
    b. **Incentives:** Marketplaces provide a mechanism for compute node owners to monetize their idle hardware, transitioning wasted resources into valuable assets.
2. **Cost Savings**
    a. **Cost Optimization:** Clients looking for computational resources benefit by gaining access to affordable computing power. This is particularly useful for workloads with varying resource demands.
    b. **Competitive Pricing:** Markets foster competition among compute nodes, leading to competitive pricing that reflects the actual supply and demand dynamics of the compute network.
3. **Flexibility and Scalability**
    a. **On-Demand Access:** Clients can access on-demand computing resources without the need for long-term commitments or significant upfront investments.
    b. **Scalability:** Clients can scale their computational needs up or down based on real-time requirements, matching the flexibility of distributed computing architectures.
4. **Incentivizing Participation**
    a. **Blockchain and Tokens:** By recording job deals, blockchains provide a verifiable and secure way of ensuring that work is performed correctly. Tokens can be used as incentives or rewards for participants, further driving engagement.
5. **Addressing Challenges**
    a. **Reliability and Trust:** Markets provide a framework for establishing relationships between clients and compute nodes, which is enhanced by blockchains and their transparent, tamper-proof record of transactions.
    b. **Security:** Blockchain-based markets inherently have identity verification and reputation mechanisms, which help to alleviate security concerns associated with decentralized and untrusted networks.
6. **Efficient Resource Allocation**
    a. **Market Signals:** Markets generate signals based on supply and demand, guiding compute nodes to areas where their capacities are most needed. This efficient allocation of resources contributes to overall network optimization.

High Level Overview of Lilypad

Clients and compute nodes send job and resource offers, respectively, to an entity known as a *solver*, which acts as a market-maker by proposing matches to participants in the network. If the client and compute node both agree to the deal, they send transactions to the smart contract. Since blockchains are based on public key infrastructure (PKI), only the corresponding owners could have signed the transactions (unless their keys were stolen). This market-making process also maintains decentralization, since clients and compute nodes have the final say on whether they agree to a deal or not.

In order to assure compute nodes that they will be paid if they compute correctly, and clients that the results they receive will be correct, both clients and compute nodes must place deposits as part of the process of making a deal and submitting a result. Smart contracts mediate financial interactions between nodes, including when the client wants to check whether a compute node has done the computation correctly.

A record of deals, results, and potentially mediation is recorded on-chain. The deals, results, and mediation consist of data that the smart contract needs to be on-chain (e.g. price per instruction), as well as mostly IPFS CID hashes. This way, all necessary data and metadata is stored on-chain, with all data that does not need to be stored on-chain being stored off-chain. Additionally, this makes the protocol much easier to upgrade incrementally, as very few changes will need to be made directly to the smart contract architecture, compared to if all data was being stored on-chain.



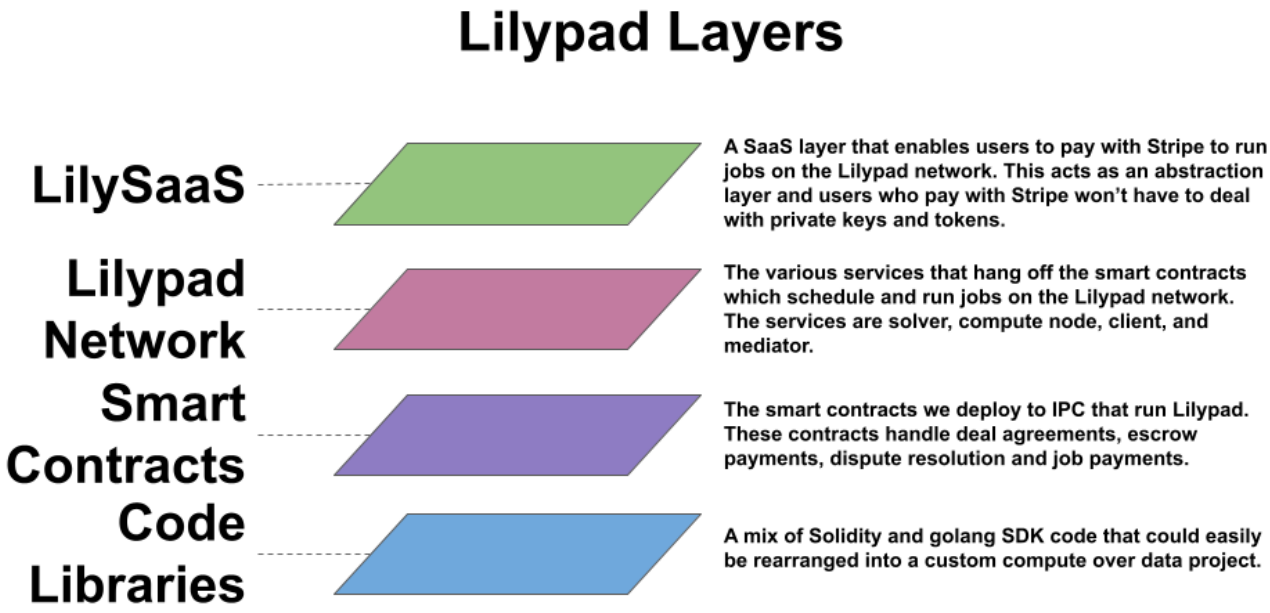
Lilypad Layers

At the core of Lilypad is an SDK that allows anybody to create a token for a distributed computing network.

One layer above are the smart contracts that handle all the financial flows of the economy – payments, staking and slashing, the storing of deals, results, and mediation, etc.

One layer above that are the services that comprise the components of the network. Clients (job creators) and compute nodes (resource provider), mediators (which eventually can comprise any node in the network), and solvers, which act as market-makers.

At the highest layer is the Value Services Layer, exemplified by Lilypad AI Studio, a user-facing SaaS that allows users to pay for jobs on the network with Stripe. The goal is to have this layer abstract away the complexities of using Web3, providing consumers with a streamlined Web 2 experience



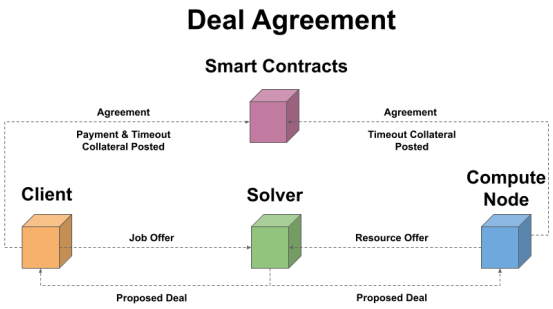
Architecture

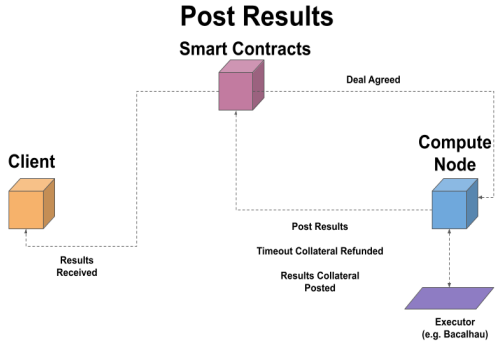
Job Lifecycle

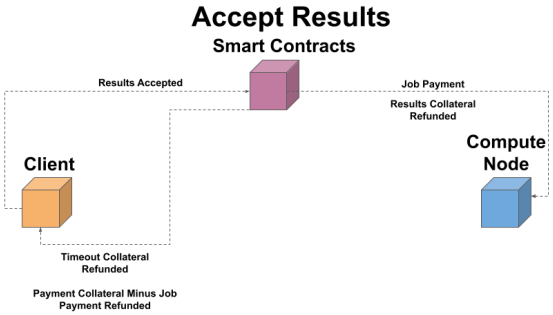
The lifecycle of a job looks like the following:

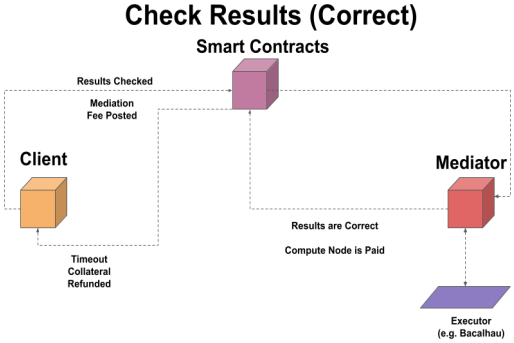
1. Job and/or resource offers are posted to the orderbook
2. The solver finds a potential match of a job and a resource offer
3. The solver proposes a deal to the client and compute node
4. The client and compute node agree to the deal by sending transactions to the smart contract
    a. As part of agreeing to the deal, they also post the relevant collateral
        i. The client deposits payment and timeout collateral
        ii. The compute node deposits timeout collateral
5. If the compute node does not post the result within the agreed-upon time frame, it loses its timeout collateral, and the process stops here

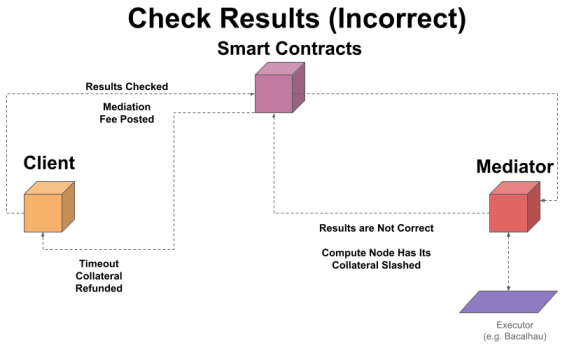
6. The compute node computes the result of the job using an executor like Bacalhau
7. The compute node posts the CID of the result on-chain
    a. The compute node is refunded its timeout collateral, and deposits its cheating collateral
8. The client can either accept the result, or request mediation
    a. If the client accepts the result
        i. The compute node is refunded its cheating collateral
        ii. The compute node is paid
        iii. The client is refunded it timeout collateral
        iv. The client is refunded its payment collateral minus the cost of the job
    b. If the client requests mediation, then a mediation protocol is invoked according to the agreed-upon terms of the deal
        i. If the mediation protocol determines that the compute node computed the result correctly
            1. The same steps occur as if the client accepted the result, except the client also pays a mediation fee
        ii. If the mediation protocol determines that the compute node did not compute the result correctly
            1. The compute node is not paid, and has its cheating collateral slashed
            2. The client is refunded their timeout and payment collateral
            3. The compute node's cheating collateral is allocated to the mediators
        iii. The result of the mediation is posted on-chain











Structure of Offers

Both clients and compute nodes can add job and resource offers to the orderbook, respectively. Since offers exist in a two-sided marketplace, in principle they can be described in any way that their submitters choose. However, in order to match offers, data fields need to be identical across job and resource offers. Additionally, in order to match, a job offer must be a subset of a resource offer - that is, a resource offer alone must be able to accommodate the requirements of a job offer.

Job and resource offers are distributed via IPFS, and their contents can be found in IPFS CIDs. An offer consists of the components that will constitute a job that runs. For example, the type of CPU and number of cores available, the amount of RAM, type(s) of GPU(s) and the number of them available, disk space, and similar data, as well as any additional data that the offer submitters provide - for example, computational benchmarks as part of a resource offer.

If only a part of a resource offer is required to match a job offer, then the solver can propose a match to the corresponding compute node that only uses a subset of its resource offer. The compute node can then either choose to accept or reject the match. If it accepts, then it can use the remaining resources to create a new resource offer. Extending this concept further, the compute node should eventually be able to arbitrarily split and combine its resources in order to accommodate the demand side of the market.

Market Making

When a solver matches a job and resource offer, it proposes the match to the relevant client and compute node. If the client and compute node both accept the match, they agree to the deal by sending transactions to the smart contract, and the deal is then posted on-chain. A deal consists of many of the same details as job and resource offers. The solver does not exist on-chain – it performs all operations off-chain, and merely suggests matches to clients and compute nodes.

Clients and compute nodes can choose the solver that they want to do market-making on their behalf. This allows for a marketplace of solvers, where nodes can choose the solvers that offer them the best services, and avoid using those which are either bad or malicious in their actions. Solvers can bring in revenue for their services by competing in this market. They must weigh the financial and time costs of finding good matches with the revenue that they will bring in for their services. This is similar to companies in the DeFi space that make money by being block proposers (these companies often run some kind of expensive computation that seeks to maximize MEV for block builders).

Market-making algorithms have been studied for many decades, and there is a robust literature from which solvers can draw to design and improve their services. The success of Lilypad may spur further innovation in this field, just like blockchains did for distributed consensus, cryptography, and economics.

Collateralization

To ensure compute nodes that they will be paid if they executed jobs correctly, and ensure clients that compute nodes have an incentive to return correct results, both parties must place collateral in escrow. There are a number of different types of collateral:

1. Client payment collateral – this ensures that the client has deposited enough money for the compute node to be paid
    a. This is put up as part of the deal agreement - that is, when the deal is posted on-chain
2. Compute node cheating collateral – this ensures that if the compute node does not return the correct result, it has its cheating collateral slashed
    a. The way that the compute node will convey the amount of collateral they will deposit to indicate that they will not cheat is via a collateral multiplier. The compute node commits to a multiple of whatever they will charge the client ahead of time as part of the deal agreement. The actual collateral is put up after the result is computed and sent to the client. This eliminates the problem of determining how much collateral to deposit for arbitrary computations in advance, a major shortcoming of most, if not all, prior research on this topic. Additionally, this method improves capital efficiency, since cheating collateral in practice will likely constitute the largest fraction of overall collateral placed into the smart contract, and this approach minimizes the amount of time it needs to be held in escrow
3. Compute node timeout collateral – this ensures that if the compute node does not return a result within the agreed-upon time, it has its timeout collateral slashed
    a. Ideally, this is a way of enforcing deadlines on jobs
    b. It is not necessary to make this collateral slashing binary - for example, a late result can be still be rewarded
    c. It is enforceable if, for example, the compute node says that they will do X WASM instructions/time. However, technical limitations may make this unrealistic, and it needs to be tested

Collateralization Example

Consider a single client, single compute node repeated interaction, where the client consistently sends jobs to the compute node. The jobs have a fixed cost of c, and the client checks results with probability p with a trusted third party. In this case, in order to make the expected payoff from cheating less than the expected payoff from being honest, the compute node must put up collateral of c * (1/p – 1) (see Section 5 of this paper for reference). The logic behind this generalizes to jobs with different costs.

However, the amount of collateral for a particular job is not determined by the protocol, but rather the details of the deal that the nodes agreed upon. As long as both the client and compute node agree, these deposits can be set to any values, including 0. For example, if the client and compute node have a trusted relationship, it may be that they both want to set the cheating collateral to 0.

The issue of collusion in mediation changes the logic applied above. For example, if there is only one mediator, the client could collude with the mediator in order to steal the compute node's collateral, and the compute node could collude with the mediator to return a false result. A similar, but less pernicious, version of this problem applies if the mediation protocol consists of a consortium of nodes. This topic is explored further in the sections on verification.

Capital Efficiency

Collateral pooling could lower capital requirements for collateralization, which may become a concern as the network scales.

Mediation

Upon completion of a task, the compute node submits a result on-chain in the form of a CID, so the actual result is stored on IPFS. Should the client request mediation, a mediation protocol is invoked, with the result of the mediation protocol posted on-chain. Payments are made according to the output of the mediation protocol.

If the mediation protocol finds that the compute node computed the result correctly, then the compute node is paid what was agreed upon in the deal, and the mediators are paid for their work as well. If the compute node is found to be at fault, then its cheating collateral is slashed, and paid out to some combination of the client and the mediators.

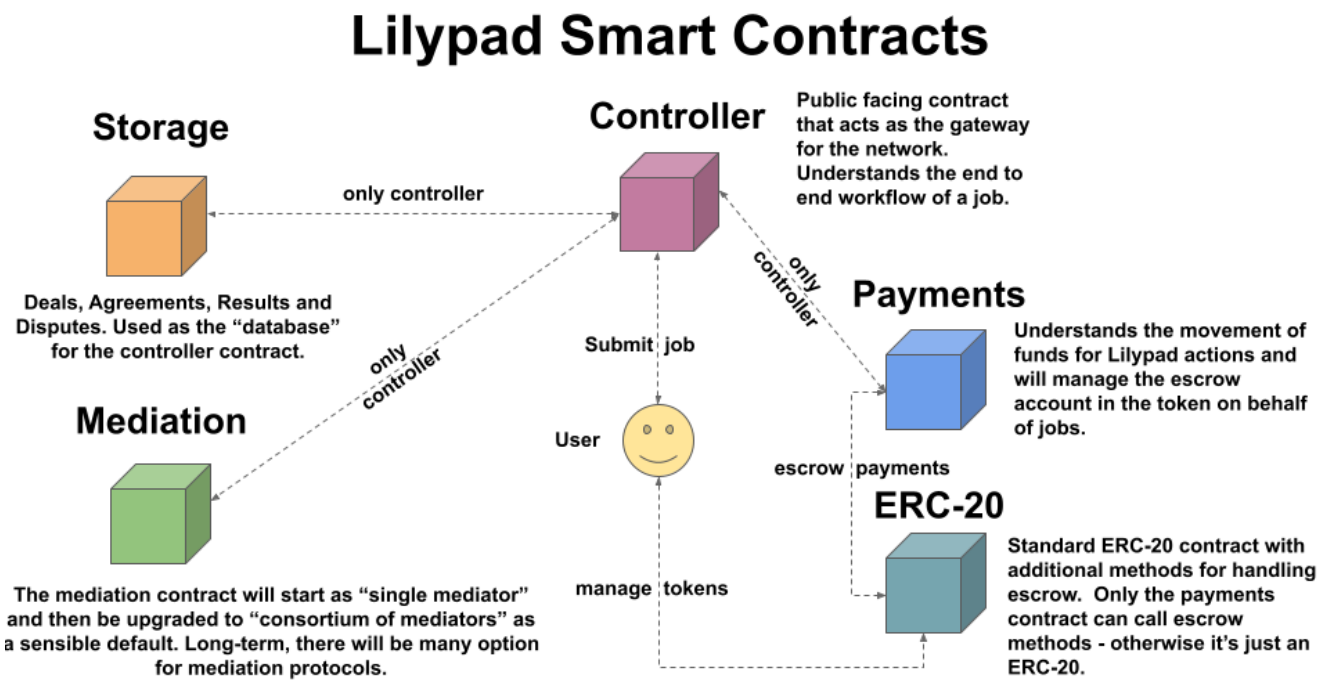
It is possible that further research determines that collateral be sent to addresses other than those involved in the mediation process (e.g. other nodes in the network, or a burn address). The reasons for this are explored in the sections on verification.

As will be explored in the sections on verification, there are a number of different ways to do mediation. To provide maximum flexibility for users of the protocol, Lilypad has pluggable mediation protocols, meaning that nodes can choose from among a number of different options. The default functionality will be for mediation to be conducted by some subset of nodes that the client and compute node agreed upon at the time of the deal agreement. Using a consortium of mediators in a mediation protocol is a simple but reasonable heuristic for achieving trustlessness and decentralization, as will be explored in the sections on verification.

Smart Contract Infrastructure

There are several interacting components of the smart contract infrastructure, which is designed with extensibility in mind.

At the top level is the Controller, which is a public-facing contract that acts as a gateway to the network. The Controller is responsible for overseeing the posting of deals, results, and mediations on-chain, which trigger the flow of payments into and out of escrow. It orchestrates all of the other contracts, ensuring that the correct data is posted on chain, and that the flow of tokens follows the rules that were agreed upon by the nodes.



The token is a standard ERC-20 contract, with additional logic implemented for handling payments into and out of escrow.

The payments contract facilitates the flow of tokens between clients and compute nodes, using escrows to hold tokens until they are ready to be disbursed according to the rules of the contract. Only the payments contract can call escrow methods.

The storage contract acts as the database of the protocol. It keeps track of all the deals made, the status of the agreements (i.e. what stage they are in – negotiating, processing, mediating, etc.), the results provided by the compute node, and the results of mediation (if it happened).

The mediation contract is used when a party wants to check the results of a computation. There can be more than one mediation protocol to choose from, meaning that nodes have options, and can in fact use custom mediation protocols if they so choose.

Blockchains and Off-Chain Compute

Verifiable Computing

The issue of verifiability in such a trustless, permissionless environment is crucial. How do clients know that they are getting the correct results returned to them? This is the subject matter of a large field of computer science known as verifiable computing.

Verifiable computing is dedicated to ensuring that outsourced computations (that is, computations that are not done locally) are done correctly. In some scenarios, it cannot be assumed that the node to which a job is being outsourced will compute the result honestly, or that it is not faulty. Moreover, verifying the result should have less overhead than computing the result in the first place.

There are three main categories of verifiable computing: cryptographic methods, hardware/software methods, and replications methods. However, since we are discussing verifiable computing in the context of blockchains, it is pertinent to first discuss how the verifiability requirements of Lilypad differ from those of blockchains.

Global and Local Consensus

While blockchains provide safety and liveness, the massive replication of computation becomes too costly when that level of security is not needed. There is a difference between global consensus, which is necessary in blockchain environments, and local consensus, which is more suited for two-sided marketplaces. In global consensus, all nodes need to be convinced that every computation was done correctly. In contrast, in local consensus, only a small number of nodes - potentially only one node, the client - needs to be convinced that a computation was done correctly.

Ostensibly, for a two-sided marketplace, this implies that only a client really needs to be convinced that a computation was done correctly. However, these computations are not done in isolation, and the interrelation between a client choosing one node repeatedly versus many different nodes, as well as the need to create a protocol that any client can come along to with no prior experience and trust that cheating is disincentivized, implies the creation of a global game that, while not requiring global consensus in the traditional sense, emulates it in some manner.

Cryptographic Methods

Cryptographic methods rely on cryptography for their security. These methods include interactive proofs (IP), zero-knowledge proofs (ZKP, a type of IP), and multi-party computation (MPC).

In interactive proofs, verification of a statement is modeled as an interaction between a prover and a verifier. The goal of the prover is to convince the verifier that the statement is true, even when the verifier does not have the computation resources to do the computation itself. The protocol must satisfy completeness (if the statement is true, an honest verifier will be convinced) and soundness (if the statement is false, the prover cannot convince the verifier except with some negligible probability).

Zero-knowledge proofs are a type of interactive proof where the verifier learns nothing about private inputs of the computation, other than that the outputs were computed correctly from all the inputs (some of which may be public/known to the verifier). A ZKP can be made non-interactive, in which case it is called a Non-Interactive Zero-Knowledge Proof (NIZK). Two common variants of NIZKs are zk-SNARKs (zero-knowledge Succinct Non-interactive Argument of Knowledge) and zk-STARKs (zero-knowledge Scalable Transparent Argument of Knowledge). Like IPs, ZKPs must also satisfy the requirements of completeness and soundness.

Multi-party computation allows multiple parties to jointly compute a function over their individual inputs without any party revealing its input to other parties. The main objectives of MPC are privacy (parties

should learn known about each others' inputs), security (some level of anti-collusion preventing malicious attempts to learn information), functionality (the ability to compute functions over data), and robustness (the protocol should work correctly even in the presence of malicious behavior or faults).

Trusted Execution Environments

Trusted Execution Environments are secure and isolated enclaves, where code and data inside of the enclave are insulated from the rest of the system, including the operating system, applications, and other enclaves. The goal is to maintain both the confidentiality and the integrity of the code and data.

Verification-via-Replication

Verification-via-replication - often described using the adjective "optimistic" in the blockchain world - relies on recomputing the computation to check whether the end result is the same. The benefits of this method are that it is the easiest to understand, and in some sense, the easiest to implement.

In contrast to the other approaches, verification-via-replication often requires game-theoretic mechanisms such as collateral slashing and reputation layers to counter cheating and collusion, the latter of which is the key challenge for this class of mechanisms.

One of the downsides of this approach is, of course, the extra effort expended on recomputing computations. However, with proper incentives, the overhead of recomputing can be reduced dramatically, while simultaneously enabling network scaling. It is also important to keep in mind that the overhead of this approach is much lower than cryptographic methods, which usually have much higher overhead.

Verification-via-replication is the first method that Lilypad will support. However, support for cryptographic methods and TEEs is necessary for the creation of a truly robust distributed computing platform, and so support for these alternatives is part of Lilypad’s mission.

Pure Verification-Via-Replication

There are a number of ways to implement a verification-via-replication protocol. Lilypad opts for starting with a “pure” version, wherein checking a result requires re-running the entire computation and comparing results. This method is used for a few reasons:

1. A precondition for alternative verification-via-replication protocols is that computations are deterministic and reproducible. This means that pure verification-via-replication should always allow for cross-result comparisons
2. Alternative approaches to verification-via-replication require additional overhead, such as storing intermediate results/computation traces, splitting computations, etc. This makes it more difficult to support arbitrary computations
3. Lilypad’s pluggable mediation protocols allow developers to develop their own rules around consensus of results, meaning that they can recover any alternative methods

Mixed Approaches

Some approaches, like those of Truebit (described below) or Arbitrum (which uses an IP), work by breaking down the computation to a point of single point of disagreement between a prover and a challenger, and re-executing that step in order to determine the prover's veracity. While this is a good approach, these methods have so far only been implemented for on-chain transactions and smart contracts. Other approaches could be used to prove the computational step in question (e.g. with ZKPs), but the issue again lies with the domain-specific application limitation.

The benefit of Lilypad is that developers have the freedom to choose which verification protocol they want to use - if they even want to use one at all.

Determinism

There is a natural question that arises from requiring the computations to be deterministic - what if they aren't? This leaves a few options, which depend on the application. One option is to use cryptographic methods (which may require some source of verifiable randomness), or TEEs. Another is to use approximate agreement - if the verification yields a result close enough to the result being checked, then the latter is considered valid. Approximate agreement requires an application-specific distance measure for results, which introduces developer overhead, but may be worth it. The final option is to simply trust the compute node, and not verify the result. The downside to this approach is the requirement of trust, and that it, by definition, does not fall under verifiable computing. The upside to this approach is that there has been for many years now a thorough literature developing on reputation systems, which provides a wide space of options for implementing such protocols.

Prior Verification-via-Replication Protocols

There have been many decentralized computing platforms constructed over the years. Among those pre-dating blockchains and cryptocurrencies are BOINC and HTCondor. Since the blockchain revolution, there have been many new attempts, including Primecoin, Gridcoin, Curecoin, Pinkcoin, Banano, Truebit, Golem, Akash, Fluence, Gensyn, Koii, Blockless, and Nexus – many more are popping up as this paper is being written.

Verification-via-replication for deterministic workloads is among the most common anti-cheating mechanisms employed by existing protocols. We will briefly go over three such protocols here - Modicum, Truebit, and Smart-Contract Counter-Collusion - to provide a concise introduction, and then explain our approach.

Modicum

The original version of Modicum had five key components: Job Creators (JC), Resource Providers (RP), Solvers (market makers), Mediators (agreed-upon third parties for mediation), and Directories (file systems, which Lilypad has replaced with IPFS and Docker registries). Job Creators are clients, the ones who have computations that need to be done and are willing to pay. Resource Providers are those with computational resources that they are willing to rent out for the right price. Solvers are

market-makers; they match the offers from JCs and RPs. Mediators are third parties trusted by both JCs and RPs to arbitrate disagreements. The Directories are network storage services available to both JCs and RPs.

Job Creators are only allowed to submit deterministic jobs to the protocol. The Mediator exists to check for non-deterministic tasks submitted by the Job Creator (which can be used by the Job Creator as an attack vector to get free results), and fake results returned by the Resource Provider. The method for determining whether a job is deterministic or not is for the Mediator to run a job n times and check to see whether it receives different answers.

Modicum combines two separate ideas: checking the result from a Resource Provider to see if it is correct, and checking a result from a job submitted by a Job Creator to see if the job was deterministic or not. This implies that there is no capability for a client to simply check whether a result is correct or not, without the possibility of its collateral being slashed.

The issue of the Resource Provider and Mediator colluding to return a fake result is not addressed by this protocol. The authors allow for a Job Creator or Resource Provider to remove a Mediator from their list of trusted Mediators if they no longer trust it. However, that still leaves room to cheat at least once, and ideally this should be disincentivized from the outset.

There is also the issue of collateralization. The Modicum protocol, as well as a number of other protocols, assume (approximate) guesses as to the cost of jobs in advance, so that nodes can deposit the correct amount of collateral. However, doing so is fraught with complications; we provide an alternative approach in the Mechanisms to Explore section.

Challenges

- The Mediator is basically a trusted third party
- The client cannot simply check a result without its collateral being slashed if the resource provider was honest, which is a consequence of the client attack model
- No accounting for repeated games
- Collusion model not fully explored

Takeaways

- Potential client attack, though one that can be mitigated by technical means by ensuring that the computation is deterministic
- Useful prototype for a two-sided marketplace, as the authors provided an open-source implementation of their protocol, on top of which the first version of Lilypad was built (the authors' follow-up paper for stream processing applications is also useful)

Truebit

Truebit is a protocol for outsourcing computation from blockchains, built using smart contracts on top of Ethereum. It was one of the first attempts to create a blockchain-based, real-world distributed

computing platform. As such, Truebit introduced many important concepts and ways of thinking about the field, and has been very influential in its development.

The original potential use cases were trustless mining pools, trustless bridges, scaling transaction throughput, and scalable “on-chain” storage. Since its goal is to scale on-chain computation, it aims for global consensus: "Since there exist no trusted parties on Ethereum’s network, by symmetry we must allow any party to be hired to solve any computational task, and similarly anyone should be able to challenge a Solver’s outcome of a computational task. The latter requirement ensures that TrueBit operates by **unanimous consensus**." [emphasis added]

The components of Truebit are Miners, Task Givers, Solvers, Verifiers, and Judges. In order to incentivize checking results, random errors are forced into computations, with jackpots awarded to those who find them. These jackpots are funded by taxes on computations.

The verification game consists of a series of rounds, where in each round, a smaller and smaller subset of the computation is checked. Eventually, only one instruction is used to determine whether a Solver or Verifier is correct: "In fact, only one instruction line is used in a verification game. There will be a part of the program code where there is a discrepancy between the Solver and the Verifier. The instruction of that discrepancy point is used to verify who is right."

The authors claim that Sybil attacks are mitigated by pairwise Sybil-resistance between the parties of Task Givers, Solvers, and Verifiers, with Judges and Referees, whose roles are played by Miners, assumed to function as intended. Likewise, they claim that attacks to get bogus solutions on-chain by scaring off Verifiers are mitigated by the economics of deposits, taxes, and jackpot rewards. Additionally, a cartel of Solvers who absorb losses until they find a task with a forced error, upon which time they will receive the jackpot, will lose money in the long-term, since the expected Solver deposit per task is higher than the expected jackpot per task. Addressing another attack, the authors claim that an attack involving a flood of Challengers who try to take as much of the jackpot reward resulting from a forced error as possible is mitigated by having the total jackpot reward decrease as the number of Challengers increases.

Challenges

- Does not scale to large/complicated/arbitrary computations
- No formal theorems or proofs, no simulations, many plausible but unsubstantiated claims, especially regarding collusion
- Everything is done on-chain
- This model does not work well with two-sided marketplaces, because
    - It aims for global consensus, where any node is allowed to do the computation, whereas in two-sided marketplaces, clients need to be able to choose which nodes they are paying to do the computation
    - Clients may have time restrictions on their computations, and cannot wait for cases where their computations were injected with forced errors
- No accounting for repeated games

Takeaways

- Taxes and jackpots are a valuable tool to create a global game that affects local outcomes
- Provides a list of potential client attacks
- No TTP required for mediation

Smart Contract Counter-Collusion

The authors reason that cryptographic methods for verifiable computation are too computationally expensive for real-world scenarios. For that reason, they rely on verification-via-replication. The scenario is one in which a client *simultaneously* outsources computation to two clouds, where those two clouds deposit collateral into smart contracts in such a way to create a game between them, where the game incentivizes doing the computation honestly. The central challenge that the authors tackle is the issue of collusion - that is, what if the two clouds collude on an incorrect answer?

In contrast to Modicum, the client is assumed to be honest, and in contrast to Truebit, a trusted third party (TTP) is used to handle disputes.

Three Contracts

The authors use a series of three contracts to counter collusion.

The first game is an induced Prisoner's Dilemma - to avoid the two clouds colluding, one cloud can be rewarded the other cloud's deposit (minus payment to the TTP for resolving the dispute) if the former returned the correct result and the latter did not. Thus, each cloud is better off giving the other cloud fake results while computing the correct result itself. This contract is called the **Prisoner's contract**. It is analogous to the equilibrium in the classic prisoner's dilemma being defection <> computing the correct result and giving the other node a fake result if offered to collude.

However, the clouds can agree to collude via a smart contract as well. They could do this by both depositing another amount into a contract, where the leader of the collusion must add a bribe (less than its cost of computing) to the contract as well (disregarding the bribe, both clouds deposit the same amount of collateral). The deposit is such that the clouds have more of an incentive to follow the collusion strategy than to deviate from it. This contract is called the **Colluder's contract**.

In order to counteract the Colluder's contract, a **Traitor's contract** is used to avoid this scenario by incentivizing the clouds to report the Colluder's contract. The basic concept is that the traitor cloud indeed reports the agreed-upon collusion result to the client in order to avoid the punishment in the Colluder's contract, but also honestly computes and returns the result to the client in order to avoid the punishment of the Prisoner's contract. The client must also put down a deposit in the Traitor's contract. Only the first cloud to report the Colluder's contract gets rewarded. The signing and reporting of the contracts must happen in a particular order in order for this process to work.

The authors prove that these games individually and together lead to a sequential equilibrium (which is stronger than a Nash equilibrium), meaning that it is optimal not only in terms of the whole game, but at every information set (basically the set of options each player has at every turn).

Challenges

- A Colluder's contract can be signed on different chains (or even off-chain). In order to mitigate this, the Traitor's contracts would have to become cross-chain (which is a major technical challenge), not to mention the possibility of cryptographically secure contracts (e.g. MPC contracts) where one of the parties alone would not be able to prove the existence of this contract
- Relies on TTP to resolve disputes
- Every task is replicated (that is, two copies of each job are always computed)
- Assumes client is honest
- Assumes amount of collateral known beforehand
- No accounting for repeated games
    - It is well known that in the repeated Prisoner's dilemma, depending on the assumptions, cooperation becomes the equilibrium

Takeaways

- The contracts and the payoffs that they induce offer a valuable toolbox to think about the problem of collusion
- The contracts offer, in a restricted setting, an ironclad way (assuming the proofs are correct) of preventing collusion

Lilypad's Approach

While many verification-via-replication protocols have been implemented, game-theoretic verifiable computing is still a developing field, with no mechanisms believed to be fully secure. For this reason, Lilypad aims to improve prior protocols by building its own verification-by-replication protocol.

Basic Ingredients

There are a few basic ingredients to Lilypad's approach to game-theoretic verifiable computing.

The first is the capacity for modular mediation protocols. The ability to choose which mediation protocol to be used, if any, allows developers to implement any mediation protocol that they believe is most appropriate for the problem. Additionally, it allows researchers to improve existing protocols and test new ones.

The second is to not assume that any particular approach is correct. The importance of maintaining a trustworthy network necessitates rigorously testing the variety of approaches available.

The third is the approach towards "utility-maximizing agents" as being the nodes against which we are trying to prevent cheating, which will be explored below.

Game Theory Background

Most protocols underpinned by game theory assume that agents are rational, and are aware of how to act in their own self-interest. However, nodes are operated by humans, who are fallible. Furthermore, it is difficult for humans to predict how truly rational, self-interested actors would behave. What then, can we do? Either we can try to account for the irrational behavior of humans, or we can try to emulate the behavior of utility-maximizing agents. While there is a large amount of game-theoretic literature dedicated to the former, we opt for the latter for reasons that will become clear below.

While this problem setting - verifiable computation by way of game theory - is different from many game theoretic settings, we can draw inspiration from commonly used concepts like the revelation principle and strategyproofness. Both strategyproofness and the revelation principle are centered around the idea of incentivizing agents to truthfully report their preferences. Most approaches in the literature rely on analytic methods to determine what rational agents will do by analyzing their payoffs as a function of their preferences, the behaviors of other agents, and the mechanism under analysis. Ultimately, we are also aiming to find (a) mechanism(s) that lead(s) to an equilibrium where all agents choose to not cheat and not collude.

Using this approach, the utility-maximizing agents that are baked into the assumptions are constructed first. In this manner, we use an adversary-first approach to this problem. With proper construction of the game and training of the agents, these agents may very well behave in manners not predictable by humans. Any protocol that is robust to cheating by such agents should be sufficient to prevent cheating by humans, as the best that humans could do is to train such agents themselves.

One major shortcoming of this approach is that it is not known whether the desired equilibria even exist, or whether we can even find them, whether through analytic or computational means. Rather, this approach should be thought of as game-theoretic penetration testing. Absence of evidence (of ways to break the protocol) is not evidence of absence, so there is no guarantee that this approach will be able to model and help mitigate all possible attacks. Just like with penetration testing, there is no guarantee that there are no holes, but rather that very large efforts were made to find and close holes discovered by the testing.

Autonomous Agents

Note that the actual environment of a two-sided marketplace for distributed computation is extremely complicated (e.g. the heterogeneity of hardware, types of computation, latencies and bandwidths, etc.). Any theoretical/analytic approach to the problem that is actually correct should also work in simulation, so we opt for a simulation-driven approach.

The way that we can emulate perfectly rational behavior is by training autonomous agents to act on behalf of their human owners in a utility-maximizing manner. At that point, the challenge is to design the global game to drive the probability of cheating to zero - ideally, to make it be equal to zero - which is no small feat in a trustless and permissionless environment. However, the simplifying assumption that we are in fact operating with utility-maximizing agents conceptually simplifies the problem immensely.

The process begins by creating a digital twin of a two-sided marketplace. In this environment, autonomous agents acting on behalf of client and compute nodes will be trained to maximize returns based on data gathered in simulation. For now, we will elide maximizing returns by optimizing scheduling, though this is a future topic of interest. We will use techniques primarily from the field of multi-agent reinforcement learning in order to train the agents. The precise methods we will use (e.g. modes of training and execution, homogeneous vs. heterogeneous agents, choice of equilibrium, self-play vs. mixed-play, value-based vs. policy-based learning, etc.) will be determined in the course of building the simulation. See the pre-print by Albrecht, Christianos, and Schäfer for our reference text.

At a minimum, the action space for an autonomous agent representing a compute node should be to cheat or not to cheat, and to collude or not collude within a mediation protocol. The observable environment for nodes on the network should include all data stored on the blockchain - that is, the sequence of deals, results, and mediations - as well as the information in the orderbook. While the orderbook will be off-chain, we model in the digital twin the orderbook acting as a centralized, single source of truth that all agents have access to. In the long-term, nodes will have (potentially non-identical) local information regarding other job and resource offers on the network.

Further work may explore agents forming beliefs about the hardware and strategies of other agents, but that is beyond the scope of the current work.

First Principles Approach

We conclude with two "axioms" upon which we will base our simulations:

1. Every agent attempts to maximize its utility, including cheating and/or colluding if necessary
2. All other components of the game should lead to a "good" solution

"Good" solutions can take a number of forms:

1. Nodes never have an incentive to be dishonest
2. Nodes have an incentive to be dishonest that goes to zero as a function of the parameters of the protocol
3. (1) or (2), but under some simplifying assumptions, such as there being some fraction of honest nodes within every mediation protocol

A good solution would achieve any of these goals. Alternatively, another valuable outcome of this research would be to discover under what assumptions these goals can or cannot be met, or if the goals are even possible to achieve at all.

Mediation Protocols

There are a number of possibilities for different mediation protocols, some of which can be combined with each other. The following is a non-exhaustive list of options. Note that some of these mechanisms clearly would not be able to deter cheating and collusion alone.

Within a mediation

There are several variations of the structure of the mediation protocol; the following parameters can be varied:

1. The number of nodes in the mediation protocol
2. If there are more than two nodes in the mediation consortium, the consensus threshold that determines which result is the one considered to be correct
3. How the nodes are chosen
    a. For example, the baseline in Modicum - only mediators that both the client and the compute node mutually trust can be used for mediation
        i. Even with this baseline, there is still a question of how to choose the node(s) - it can be random, be determined by an auction, or any other method
4. Recursive mediation - that is, if there is no consensus in the consortium, do another mediation
    a. There is a large space of possibilities regarding how to execute this
    b. There needs to be a limit to the number of nodes this recursive process can use. For example, the set of potential nodes can be the same as the set of mutually trusted mediators, as described above

Taxes and Jackpots (inspiration from Truebit)

Taking inspiration from the taxes and jackpots scheme used in Truebit, deals can be taxed, with those taxes going to a jackpot that is then used to reward nodes via some distribution protocol determined by the mediation process. For this, we want to be able to take any fraction of the jackpot(s) and distribute it arbitrarily to arbitrary nodes (perhaps even those not involved in the mediation process).

This is a particularly interesting approach because the taxation + jackpots mechanism inherently creates a global game that impacts local outcomes. While it may lead to potential collusion attacks, the tool alone is very useful, especially in conjunction with the other methods discussed in this section.

This method may also be useful in creating a robust platform where some clients do not care to check their results. That is, if some clients consistently do not check results, it may be difficult to assert that the network is secure. Taxes and jackpots may be a way to address this, as they can reward checking calculations even if the client did not choose to do so themselves.

Prediction/Replication Markets

Prediction markets have been well-studied in a variety of different fields. More recently, a type of prediction market called a replication market has been explored in the context of replicability in science. With this inspiration, it may be possible that allowing nodes to make bets regarding the replicability of the computations of nodes may be useful in mitigating cheating. For example, nodes with a low prediction for replicability may act as a signal for that node's reputation and encourage it to behave honestly.

It is possible to overlay this mechanism on top of taxes, allowing nodes to choose where their taxes go in the prediction market.

Additionally, since Automated Market Makers are closely related to prediction markets, it may be possible to leverage DeFi tools in this context.

Staking behind nodes

Allow users to stake behind nodes. This is similar to prediction markets, but with slightly different economics. Like with prediction markets, it may be possible to tax users and then allow them to choose which nodes they stake behind. Overall, this approach is similar to delegated Proof-of-Stake.

Announcing successful cheating

It may be possible that a node announcing that it successfully cheated (and thereby receiving a reward) benefits the robustness of the protocol. This mechanism would have to be investigated thoroughly, but may provide some surprising insights.

Frequency of checks

How often should a client check results? Clearly it is related to the amount of collateral that the other node deposits, how much they value getting true/false results, reputation, and so on. This is a parameter that the client would need to learn to maximize its own utility.

Reputation

The ledger can maintain a record, for each compute node, of the number of jobs the compute node has completed, the number of times its results were checked, and the number of times those results were replicated successfully. It can also keep a record of how nodes rated the performance of other nodes. All other nodes (client and compute) could locally run some arbitrary function over these numbers to determine how reputable they find that node. There is a deep literature dedicated to reputation systems in computing environments which can be leveraged to make Lilypad's reputation system robust.

Storing Inputs/Outputs

Results can only be replicated for as long as the inputs are stored somewhere. The client, compute node, or some other entity can pay for storing the inputs/outputs of jobs. The longer they are stored, the more time there is to check the results, which affects things like collateralization, the frequency of checks, etc.

This is related to, but not totally overlapping with, the amount of time that a node might have to wait before getting paid, which is the same time interval that a client has to check a result. However,

checking the result after the node gets paid and receives back its collateral may still be possible, with other penalty schemes (or reward schemes, for example, coming from jackpots).

Anti-Collusion via Obfuscation

Colluding requires the following knowledge in order to enforce the parameters of the collusion:

1. The public keys of the nodes participating in collusion
2. The results that were posted by those public keys
3. The payouts to the public keys

In order to sign a collusion contract to begin with, the public keys must be known. However, in a mediation protocol with enough nodes, it may be possible to obscure (2) and (3) by

1. Having nodes submit results to the mediation protocol in an obscured/anonymous way
2. Have nodes be paid out according to the results of the mediation protocol in an obscured/anonymous way

If these two criteria can be met, then a mediation protocol based on them might be capable of imitating the game-theoretic outcomes seen in the Smart Contract Counter-Collusion paper.

There have been many decades of cryptography and security research focusing on similar problems to these. It may be the case that it is already possible to do this; otherwise, there is a large amount of ongoing research on the topics of privacy-preserving transactions, and much prior work in the flavor of secret-sharing/MPC/Tor/Monero/ZKPs that could enable this.

Applications

Artificial Intelligence

Inference

Artificial Intelligence, especially generative AI, has exploded in the past year, and interest in combining blockchain technology and AI has surged. Lilypad offers a convenient and flexible way for developers to incorporate AI inference into their workflows, as well as a user-friendly interface for the general public to access low-cost inference. However, the possibilities go far beyond AI inference.

Cooperative AI

Lilypad enables the development of trustless, cooperative AI systems by enabling multiple autonomous agents to collaborate on complex tasks. Lilypad's shared ledger can facilitate coordination and communication among distributed AI nodes, enhancing the overall efficiency and reliability of cooperative AI. The emerging field of cooperative AI exemplifies the possibilities enabled by having a shared data structure that enables and can enforce credible commitments.

Federated Learning

Federated learning is a process by which training or inference of AI models is split across many different nodes, with potentially different owners and heterogeneous hardware. While the research field dedicated to federated learning is relatively new, the industry implementations are even newer, yet have enormous potential in the coming decades. Many applications of federated learning require consensus over outputs of models and aggregation of information of those outputs - a task well suited for blockchains, and a protocol like Lilypad.

Model Verification

Lilypad can be used to validate the integrity and provenance of AI models and ensure they have not been compromised.

Zero Knowledge Proofs

The explosive growth of blockchains in the past decade has prompted and massively accelerated the research and development of zero-knowledge proofs (ZKP). ZKPs are a cryptographic way of proving that a computation was done correctly without revealing some private input called a *witness*. Much of the focus of ZKP development for blockchains has been focused on zk-rollups for L2s on Ethereum, privacy-preserving transactions, and the like. However, most blockchain zk-coprocessors are still in their infancy. Bacalhau already supports Filecoin data onboarding, which is the single largest use of ZKPs in the world in terms of computational power and financial payments, and has recently become a major provider of data onboarding services.

Developer Workflows

Lilypad intends to support arbitrary deterministic compute, and would allow developers to more easily incorporate ZK computations into their workflows, without having to make the entirety of their computations be zk-compatible, which would incur substantial overhead. The ability to develop sophisticated computational workflows is critical to the long-term success of blockchains as a technology, and just like with AI compute, having the ability to incorporate zk-computations into computational workflows enables Web3 developers to create much more complex programs than they were able to previously.

Marketplace Uses

There are many applications for ZKPs within Lilypad itself. For example, compute nodes could prove their hardware capabilities in zero-knowledge without revealing sensitive details. Once privacy-preserving transactions can be easily integrated into smart contracts, payments and collateralization details can be used to keep transaction data private (although it is worth mentioning that many of the benefits here can also be accomplished with payment channels). Mediation protocols can be made privacy-preserving by having the mediating node(s) remain private, and using MPC to disassociate the outcome of the mediation protocol from the inputs of the mediating nodes.

Zero-Knowledge Machine Learning

Zero-knowledge machine learning (ZKML) allows a compute node to prove that they ran a specific AI model without revealing the model weights and/or the inputs.

Bridges

Bridges have been the topic of much attention in the blockchain world for a number of years. Cross-chain interoperability is a challenging task, as exemplified by the many hacks of such protocols. As a distributed computing standard, Bacalhau can provide a unified codebase upon which to build bridge protocols. Likewise, since Lilypad contracts are written in Solidity, they can be deployed on any EVM-compatible chain, further enabling cross-chain interoperability. Bridges built on top of Bacalhau and Lilypad can complement existing bridge protocols, including ambitious ones like zkBridge, as well as provide a platform for developers to build their own.

Streamlining Off-Chain Storage

A number of blockchains and scaling solutions do not have native storage protocols for off-chain data. Bacalhau has the native ability to take IPFS CIDs as inputs, meaning it can provide arbitrary compute over arbitrary data in a totally distributed manner.

Oracles

Lilypad compute nodes can act as oracles, accessing real-world data from other blockchains and triggering cross-chain actions based on the data that they read. This would allow developers to create their own custom oracle protocols, and more easily incorporate them into their software workflows.

Sensor Data

Environmental Data

Practitioners who use diverse sensor data, such as weather, air quality, and satellite imagery, and compute over it locally, can leverage Lilypad's ability to tap into idle computing resources and GPUs. This makes it feasible to perform complex calculations swiftly, with the added assurance of blockchain security for automatic cross-checking of results for accuracy and anomalies.

Drone Data

For instance, environmental data gathered by drones, which is critical for tracking climate change, mitigating natural disasters, and coordinating search and rescue operations, can be processed on the fly. Lilypad makes it practical to perform machine learning inference or training that was previously too costly or slow directly on the blockchain. Now, with sufficient processing power, drones can compute data locally and then send only what is needed to centralized servers or distributed ledgers. In the context of distributed ledgers, which are particularly relevant when drones have different owners, Lilypad's platform facilitates consensus on the state of observations, ensuring trust and verifiability.

Carbon Credits

In a carbon credit application, data stored on the blockchain or decentralized storage systems could be employed to quantify the results of carbon sequestration projects, subsequently translating them into verifiable carbon credits. This computational capacity could also empower community currency initiatives, using Lilypad to validate land conditions through satellite imagery or monitor environmental health via IoT sensors. The resulting land cover change maps could then yield valuable insights into the success of community development efforts, thereby contributing to both climate action and community development by providing the computational backbone for Web3-based solutions.

Wildlife Monitoring

Sensors capturing data on wildlife movement, habitat conditions, and biodiversity can be valuable for environmental research. Lilypad can automatically coordinate and reward environmental signals related to wildlife conservation, habitat preservation, and ecological modeling, enabling the tokenization of environmental assets and progressing the incorporation of environmental damage into traditional economic flows.

Agriculture and Water Resource Management

Like with carbon credits and wildlife monitoring, Lilypad can contribute to agriculture and water resource management by analyzing data from sensors, drones, and satellites. It can help provide consensus over, and reward, different cultivation strategies, irrigation schedules, soil management strategies, reforestation initiatives, and water use as measured from water distribution systems, reservoirs, and treatment plants.

Weather Forecasting

Consensus over weather data gathered from sensors, like temperature, atmospheric pressure, and wind patterns can be fed into cryptoeconomic systems that adjust their functionalities based on weather conditions.

Energy Management

Data from energy meters, smart grids, and renewable energy sources can support decentralized decision-making for energy optimization, demand forecasting, and the efficient management of power resources. In fact, as a two-sided marketplace, a modified and generalized Lilypad could also support peer-to-peer energy trading.

User Data

Most consumers today do not directly profit from their data - at best, they receive free services in return for giving up their data. While this is slowly changing due to regulation, businesses have been slow to design their profit models to enable their users to profit from their own data.

Collaborative Filtering

Related to federated learning, Lilypad can orchestrate the use of mobile user data to construct collaborative filtering for recommendation algorithms. If done in a privacy-preserving way, this would allow users to benefit from their data without giving it up, and allow developers to construct recommendation products that return a portion of the revenue back to consumers, a form of data cooperatives. A natural application is for social media analytics, which can be valuable for businesses and marketers aiming to understand and engage with their audience.

Location-Based Services

Similarly to sensor data, verifiably crowd-sourcing information in real-time opens up many possibilities in traffic control, public transport congestion, mesh networks, and many other areas. For example, there has been much research in cooperatively owned and blockchain-governed self-driving cars that substantially reduce overall costs for consumers. Combining location-based user data with sensor data, such as data coming from traffic cameras, can help significantly improve vehicle routing.

Developers who want to build privacy-respecting apps related to navigation, local business recommendations, and geofencing for personalized alerts can utilize GPS and other location data from mobile devices. Likewise, they can utilize mobile network data to optimize the use of mesh networks. This includes network performance analysis, predictive maintenance, and resource allocation for improved connectivity.

Public Health

Aggregated and anonymized mobile user data can be valuable, especially during public health emergencies. Lilypad can enable developers to construct trustworthy, yet privacy-preserving contact tracing, contributing to tasks such as monitoring the spread of diseases and predicting outbreaks, in a verifiable and traceable manner.

Digital Twins

Digital twins are virtual representations of physical objects, processes, or systems, and are projected to become much more prevalent in the coming years, scaling to an industry in the tens of billions of dollars. Lilypad offers a unique environment for the construction and simulation of digital twins, since especially for cyber-physical systems, the network itself can be comprised of real-life counterparts of elements of the\digital twins (e.g. compute nodes, sensors, video cameras, mobile devices, cars, etc.). This integration can provide feedback loops for self-improving and self-reinforcing systems.

Supply Chain

Supply chain management has long been considered a valuable use-case for blockchains. With Lilypad, the tracking of goods along the supply chain can be augmented with more sophisticated computations, such as data analytics, which brings much more value to blockchain-based tracking of goods.

Smart contracts enable automated and secure execution of agreements within the supply chain by facilitating trustless and transparent transactions, automating payment processes, and enforcing contractual terms.

End-to-End Traceability

The blockchain-based tamper-proof record of transactions enables end-to-end traceability of each transaction and movement of goods, ensuring provenance. This facilitates real-time inventory management, cold-chain (environmental) monitoring, and monitoring factors such as lead times, quality, and reliability. This can help users optimize supplier selection, identify potential risks, and enhance collaboration with each other to improve their supply chain resilience.

Risk Management

Since Lilypad jobs can act like oracles, they can provide automated analysis of data related to geopolitical events, market fluctuations, and other risk factors that impact the supply chain, allowing stakeholders to implement strategies for resilience. Additionally, the constant updating of the record facilitates automatic negotiation of prices and movement of goods, where and when it is desirable.

Coordination

The tamper-proof record, plus real-time data sharing, synchronization of plans, and collaborative decision-making facilitates collaborative forecasting and planning among supply chain partners. Similarly to sensor and mobile user data, processing real-time data on traffic, weather, and vehicle conditions helps to optimize delivery routes, thereby reducing transportation costs and improving on-time deliveries.

Hardware Profiling

Hardware profiling plays an important role in the tech and software engineering industries. Lilypad enables the creation of the largest and most robust hardware profiling database in the world, since it adds an incentivization to participate, rather than relying on volunteer contributions for the data. Such a database can be used by scientists, researchers, consumers, hardware enthusiasts, and tech companies, large and small, to better enable their purchasing choices, use their hardware more efficiently, and make more informed choices when choosing which compute nodes to select for jobs.

Performance Optimization

By analyzing hardware profiles across distributed nodes, developers can identify potential compatibility issues and optimize performance. This is beneficial for industries that require specialized hardware setups, such as scientific research, where specific configurations are needed. This information is also valuable for optimizing software applications and ensuring they meet performance expectations for embedded systems or mobile devices, where it can aid in resource management, allowing developers to optimize apps for different devices, screen sizes, and processing power.

Energy Efficiency

By analyzing the energy consumption patterns of different hardware configurations, organizations can design and implement energy-efficient computing solutions. This is particularly relevant in data centers and large-scale computing environments.

Gaming

In the gaming industry, an extensive hardware profiling database can help to optimize game performance and hardware purchases. Hardware data collected from highly heterogeneous gaming setups can be used to guide the development of games that deliver optimal performance on various platforms.

Privacy

The issue of private computations is especially tricky in permissionless and trustless distributed computing environments. While TEEs and homomorphic encryption provide solutions to this problem, they are not fully mature technologies capable of providing the same level of efficiency and trust as centralized cloud providers. For the time being, Lilypad will optionally use symmetric/asymmetric encryption in order to assure clients that only compute providers can see their data, unless the latter decides to leak it. For coordination problems that can use local computation over local data, ZKPs can be used, as described earlier. However, Lilypad still has a number of privacy-oriented applications.

Secure Multi-Party Computation (SMPC)

Lilypad's compute-sharing capabilities can facilitate secure multi-party computation, where parties jointly compute a function over their inputs while keeping those inputs private. MPC was discussed earlier in the context of verifiable computing, but since Lilypad can execute arbitrary computations, it can also be used for MPC.

Private Data Markets and Data DAOs

Like with federated learning, users can share access to their data for specific computations without revealing the raw data itself. This decentralized approach ensures privacy and gives users control over their data.

An oft-cited example application is healthcare data. Patients can come together to form data DAOs, where they can collectively sell access to their private data without revealing it. This allows individuals and healthcare institutions to jointly analyze anonymized patient data while keeping individual records secure. Likewise, researchers working with private datasets, such as social scientists and economists, can also benefit by running computations over users' private data without revealing anything about it.

Decentralized Private Messaging

Lilypad can augment protocols that support decentralized private messaging platforms with arbitrary compute, including AI inference.

Compute Sharing and Cooperatives

Following up on hardware profiling, it is clear that some machines are better suited for some tasks than others. Machines can cooperate by pooling their resources in order to derive higher revenue for all, and in the case of public good computing, can balance their preferences over projects with their machines' capabilities.

Applications include scientific research collaborations, where projects around the world can lend each other their spare computing resources, effectively forming a compute sharing cooperative. Likewise, individuals or institutions requiring large amounts of compute for rendering and animation can share their spare capacity, keeping track of their contributions on the blockchain, perhaps even creating a community token.

Public Goods Computing

Prior distributed computing projects such as BOINC demonstrated the popularity and feasibility of volunteer computing for scientific projects, which are effectively a public good. Early cryptocurrencies such as Gridcoin attempted to crypto-economically incentivize participation in BOINC, and while achieving many successes (for example, having among the most nodes and being the most decentralized cryptocurrencies, as well as having an organic and passionate community), did not achieve their long-term goals, including increasing participation in volunteer scientific computing. However, with modifications and improvements, the pitfalls of prior approaches can be avoided, and their original missions driven forward.

The space of scientific projects that can benefit from Lilypad's marketplace and coordination mechanisms is vast. To give a few examples, climate modeling and environmental simulations, drug discovery, genomic data analysis, disaster response (e.g. helping discover the structure of Covid-19's spike protein), and astronomical data processing. To explore some of the possibilities, see BOINC's past accomplishments.

Revenue and Profit

Fee-based Model

The simplest way to generate revenue is to take a fee for every job executed by the network (which can optionally be waived for public goods projects). This has a number of advantages over launching a network token, namely that it maintains the Lilypad's chain-agnosticism, makes it clear to each L1/L2 that the project only increases demand for its native coin/token, and avoids complicated regulatory issues.

Liquidity Providers in DeFi Protocols

If the revenues from fees aren't sold or invested in some other way, an immediate way to put them to use is by providing liquidity to DeFi protocols.

Value Services Layers

As with any platform that provides compute, it is possible to build revenue-generating businesses on top of Lilypad.

Lilypad AI Studio

The first example of a value service layer is Lilypad AI Studio, which is a user-facing SaaS that allows customers to pay for generative AI inference with their credit card. The idea is to abstract away from users who would otherwise not use Lilypad the process of using private keys, purchasing tokens, and using those tokens to pay for services. For the time being, this is achieved by the Lilypad team operating a custodial wallet on behalf of users. However, Web3 plugins and account abstraction can circumvent this necessity.

Investing in Distributed Computing Projects

As the network matures, the network will accrue reserves of many different cryptocurrencies and tokens. Due to prior efforts to bootstrap widespread adoption and make Bacalhau/Lilypad the standard distributed computing platform, the capital reserves and deep in-house knowledge will enable the network to begin investing in distributed computing projects built on top of the protocol. Thus, in a manner similar to how Amazon and Google invest in, and give free computing credits to, startups which use their platforms, Bacalhau/Lilypad can provide similar investments and services to Web3 startups.

Community and User Ownership

A truly decentralized protocol would be owned by its stakeholders - the communities and users who rely on its services. Thus, over the long-term, it would be necessary for ownership over the protocol to become widely dispersed. This still presents a massive financial opportunity during the growth of the protocol, but in order to stay true to its principles, ownership would have to become distributed.