GenLayer: The Intelligent Contract Network

Albert Castellana
GenLayer
albert@genlayer.com

José María Lago
GenLayer
jm@genlayer.com

Edgars Nemše
GenLayer
edgars@genlayer.com

Abstract

In this paper, we introduce GenLayer - a new kind of decentralized ledger platform for executing programs called "Intelligent Contracts", which retain all the benefits of smart contracts but additionally are capable of processing Natural Language Instructions and accessing the Web. At the core of it is a Python-based Execution Environment titled GenVM capable of executing the Intelligent Contracts within a secure sandbox. We also detail a consensus mechanism called Optimistic Democracy, which enables a set of validators that leverage Large Language Models (LLMs) to efficiently come to a decision on GenVM transactions by aggregating the votes of randomly selected validators. Finally, we detail a broad set of use cases enabled by this platform, from Intelligent Oracles and a Trustless World Database to a truly Decentralized Autonomous Organization.

1 Introduction

In this paper we propose a novel system called **GenLayer** that addresses the limitations of the current generation of smart contract platforms and enables the creation of the next generation of AI-powered smart contracts which we term **"Intelligent Contracts"**. An Intelligent Contract can process natural language instructions and read data from the World Wide Web, in addition to retaining all the capabilities of current generation smart contracts.

Intelligent Contracts are executed by a specialized restricted deterministic virtual machine called GenVM, which is based on Python. It handles the deployment and interaction of Intelligent Contracts and accounts, and persists their state. Intelligent Contracts are invoked through transactions submitted by users and specify the method and parameters to interact with. It also enables Intelligent Contracts to perform non-deterministic operations like executing a call to an abstracted LLM and reading information over the http protocol. We detail a mechanism called the "Equivalence Principle" which allows a developer to define how a set of different validators can come to a conclusion on whether different answers to the same non-deterministic instruction can be considered equivalent.

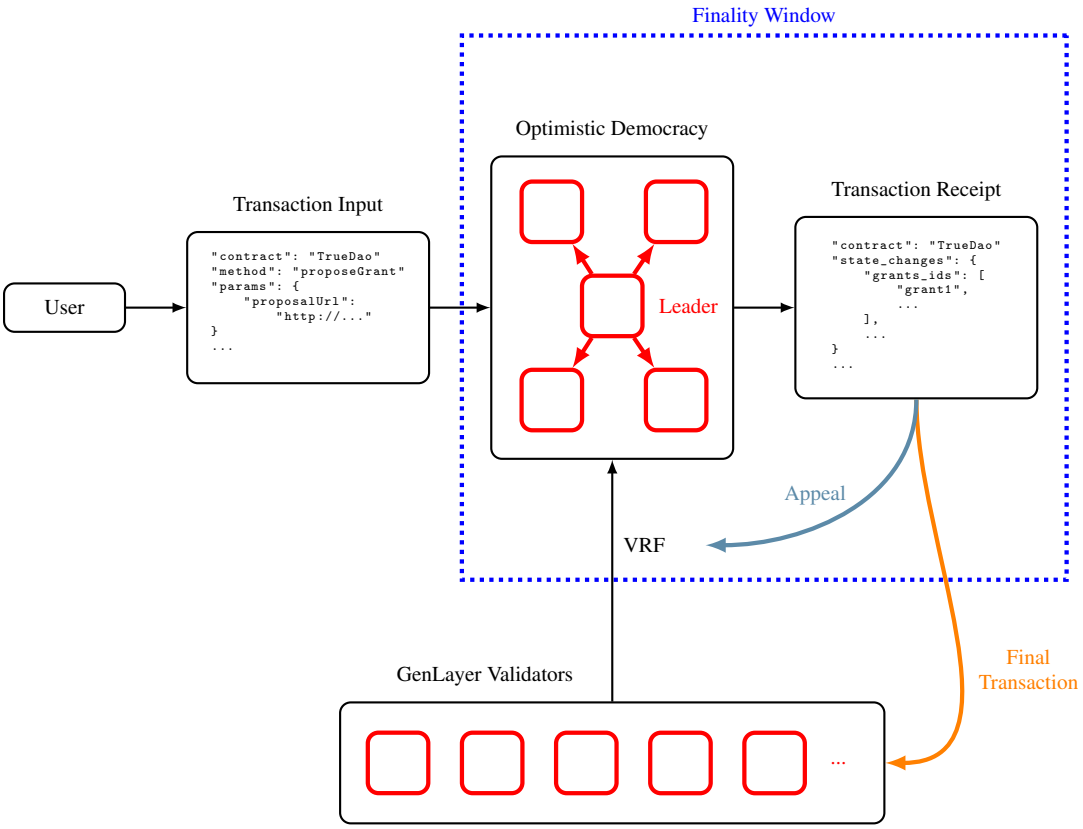
We also propose a consensus mechanism called "Optimistic Democracy" which allows these Intelligent Contracts to be self-executing and self-enforcing just like Smart Contracts, providing a seamless deployment experience for developers of decentralized applications. GenLayer provides a shared environment where multiple Intelligent Contracts can interact and rely on each other, enabling a vibrant ecosystem of applications.

This is achieved in a performant fashion by randomly assigning a small subset of all validators to each transaction. One of the validators is selected as a leader to propose a transaction execution, and the others use the Equivalence Principle mechanism to determine whether the leader's proposal is of

sufficient quality. Their result is provisionally accepted as correct, but can be appealed to a larger set of validators for reconsideration during a limited window of time after the transaction is first processed.

We also discuss security considerations of this system, and propose a mechanism called Grayboxing which enables the network to protect itself against common LLM security vulnerabilities such as Prompt Injections and Adversarial Inputs.

Finally we outline several use cases made possible by Intelligent Contracts, such as an Intelligent Oracle that can answer any question with data publicly available on the web, a Trustless World Database which can aggregate a trusted and verified database of all news pertaining to a set of subjects, and a True DAO which is a decentralized organization capable of making and executing decisions independently with respect to its constitution.



2 Background

In 2014, Vitalik Buterin introduced Ethereum [1], a groundbreaking blockchain platform, bringing into reality the concept of Smart Contracts. First proposed by Nick Szabo in 1994 [2], a smart contract is an autonomous, self-executing contract where the terms of the transaction are written directly into lines of code, enabling the execution of credible transactions without the need for trusted intermediaries or the Legal System to enforce the rules of an agreement.

Smart Contracts provide an alternative to the expensive and slow legal system. However, the current capabilities of smart contracts are limited in certain key respects, compared to the kinds of contracts humans and corporations enter into with each other.

First, smart contracts are written purely in computer code, which is very powerful for encoding exact rules and conditions, but falls short of capturing the subtlety and flexibility inherent in natural language. This flexibility is necessary for the complex agreements that enable the world's commerce.

Secondly, smart contracts are completely isolated from the outside world. They must instead rely on trusted (from the point of view of the protocol) parties called Oracles to bring information on-chain. This has meant that the scope of smart contracts to substitute the legal system has remained limited to certain areas of human activity that are easily captured in code, such as Decentralized Finance.

The advent of Large Language Models (LLMs) in recent years represents a significant leap in the natural language processing capabilities of computers. Rooted in the breakthroughs of the late 2010s [15], these sophisticated models have evolved through exponential advancements in machine learning and computational power. LLMs, such as OpenAI's GPT series [5], have the remarkable ability to understand, generate, and manipulate human language with a level of nuance and complexity previously unattainable. In particular, these language models are capable of interpreting complex instructions written in natural language, extracting data from unstructured documents and making nuanced and adaptable decisions.

There have been previous attempts to combine Artificial Intelligence and Blockchains. One such platform is Bittensor, a A Peer-to-Peer Intelligence Market [3]. In the Bittensor system, developers create subnets with arbitrary objective functions, which are used by the validators to score the outputs of miners. The miners compete for rewards by attempting to produce the best output according to the objective function for any given subnet. In this way, a subnet can be set up to perform tasks such as text generation, machine translation, information gathering and a wide array of other aims. The miners however are ranked over a number of submissions, which makes the platform unsuitable for smart contracts, as the consensus mechanism cannot reject an individual incorrect transaction. Furthermore, each subnet consists of an independent set of miners and validators, requiring a new subnet to attract and organize these parties.

Another approach is taken by Autonolas [4] which consists of open-source components that allow building a system based on Solidity smart contracts, whereby a number of validators can come to a consensus and move through a predefined set of states in a Finite State Machine (FSM). While this approach allows for building complex AI-based applications, it is not a platform - each application needs to design its own agents as well as the incentive mechanism for attracting them. It is more akin to building a custom blockchain platform for your application than deploying a smart contract.

Finally, a third approach involves running a specific LLM with the same seed and verifiable outputs, for example on a rollup such as Cartesi[1] or Cortex[2]. This enables a DLT platform with a fully on-chain LLM, the output of which can be deterministically verified by everyone. The limitation of this approach is that since the exact LLM is known and accessible to all participants, and the output has to be deterministic to work with classic consensus mechanisms, they are vulnerable to adversarial inputs, where an attacker can locally try a number of random modifications to their prompt until they find one that produces a desired malicious output, which makes the system unreliable for high-stakes smart contract transactions, where a mistake can have grave consequences.

3 GenVM

To execute Intelligent Contracts, we devise an environment called GenVM based on a restricted and modified Python interpreter. We chose Python because it is the standard language for ML and AI development, and because working with the web and Natural Language Instructions requires advanced string processing capabilities, which Python has good native support for.

The version of Python running inside GenVM will have no access to the underlying operating system, and will be completely isolated from the outside environment, I/O, randomness, networking or any

[1] `https://cartesi.io/`

[2] `https://cortexlabs.ai/`

non-deterministic functions normally available in Python. Instead, GenVM will provide specific operations like calling an LLM or reading data from the web, handled in a sandboxed fashion.

3.1 Accounts

GenVM is an account-based system, like the EVM. Each account has an address associated with it. There are two types of accounts - Externally Owned Accounts(EOAs) and Intelligent Contract Accounts. EOAs are associated with an external private key of the ECDSA type [6], while Intelligent Contract Accounts are created when an Intelligent Contract is deployed.

Both types of accounts hold a balance on the platform's native GEN token. EOAs also have a number associated with them called Nonce, which increases with each transaction submitted to the network by that account.

3.2 Contracts

Like in the EVM and other smart contract VMs, code is deployed and organized into contracts. Each contract is a class with some properties and invocable methods. When a contract is deployed, an instance of it is created and assigned an Intelligent Contract Address based on the creator's address and their account nonce. Intelligent Contracts can deploy new instances of other Intelligent Contracts.

3.3 Transactions

Deployed Intelligent Contracts can be invoked via transactions. Transactions are a bundle of serialized data that identify the address of the contract to be called, the method to be called, any parameters required for a method, the account nonce as well as the invoker's signature and the maximum amounts of gas paid to the validators for its execution and potential appeals (discussed later in the paper).

The creator of a transaction needs to provide a sufficient amount of GEN to cover the gas costs of the transaction execution, including a separate amount of gas for the desired number of potential appeals, detailed in the Consensus section further in this paper. If there isn't sufficient gas to start an appeal process, any party can supply more gas to pay for an appeal during the finality window.

If less gas is used than the maximum amount specified, the remainder is refunded to the supplier's account after transaction finality is achieved.

3.4 State

Intelligent Contracts can store their state between runs. The state of a contract is persisted on the ledger and instantiated automatically by GenVM. Developers can define data they want to store as properties of the contract class and update them during method execution. GenVM will support all serializable native Python data types.

In addition to the basic data structures provided by Python, GenVM will include native support for data types necessary for the contract to operate on a distributed ledger, such as addresses, signatures and contracts. Furthermore, GenVM will also provide native support for data structures that are useful for working with LLMs, such as Vector Encodings and Vector Stores for creating Retrieval Augmented Generation (RAG) systems in Intelligent Contracts. In RAG, the prompt passed to the LLM is augmented with relevant data from a vector store. For example, if the task involves answering questions about a specified topic, it might be useful to have a vector store of news articles about various topics, and retrieve the N most relevant articles by querying the database.

3.5 Natural Language Instructions

GenVM can execute instructions or prompts written in natural language. By this, we mean that you can give it an arbitrary instruction such as "Write a poem about decentralized networks" and "Propose a step-by-step plan to find out who is the current President of the USA". The prompts produce a

text response, which can be in any format desired, as specified in the prompt. The result can then be returned, passed into the next instruction or parsed in the code. It can even write code that can be then executed by GenVM in the same transaction, enabling advanced agentic use-cases. The code is executed in the same restricted and sandboxed GenVM environment.

While the natural language instructions are intended to be processed by LLMs, GenVM makes no assumptions about which specific LLM is being used. A validator node is free to run any model, considering the incentives provided by the consensus mechanism - namely, if a validator doesn't respond in time, or if its response is appealed, it stands to lose part of its rewards and staked tokens.

There are several benefits to this approach. First, this makes the network more resilient to attacks such as prompt injection and adversarial inputs - while a single specific LLM can be easily targeted with datamined prompts, targeting the entire aggregated network consisting of many different and unknown and Greyboxed LLMs is much less likely to succeed.

Furthermore, validators are encouraged to optimize their operations over time, to reduce both their costs and improve the quality of their responses to their odds of being appealed. This will allow the network to get faster, cheaper and smarter over time.

From an Intelligent Contract developer's perspective, any prompt should be written in a general manner, as if an ideal judge might interpret it, and rely on the consensus mechanism to ensure it will be executed correctly.

3.6 Reading data from the Web

One of the features that are seldom found in distributed trustless ledger systems is accessing the outside world, and querying the World Wide Web. Precisely because GenLayer is built around non-deterministic execution and it leverages LLMs, it is possible to access any data available over the http protocol, and digest it, regardless of its structure.

Because hardcoded web parsing is fragile, as websites are constantly updated, even if a consensus mechanism was built around reading web data, it would require constant maintenance. Furthermore, as data on the web changes constantly, it would be a non-trivial problem for multiple validators to agree when data has been retrieved correctly or not.

LLMs solve all of these problems and enable even more useful functionality such as summarization or other processing of the retrieved data. The equivalence principle mechanism described in the next section enables multiple validators to agree on data validity in a flexible manner, and LLM based agents are also capable of data extraction from an ever-changing HTML structure.

In effect this enables Intelligent Contracts to act as a smart, universal oracle, able to read any data in a resilient and reliable fashion.

3.7 Equivalence principle

Because the same input to a non-deterministic call to an LLM or reading from a website can produce different outputs, it is necessary that separate validators can decide when differing outputs can be considered equivalent to achieve consensus. Take the example of an intelligent contract tasked with summarizing any input the user is providing - different LLMs and even the same LLM using different seeds will provide a different output, yet there can be many different summaries that can be considered a valid completion for the task.

To enable multiple validators to come to a consensus on a single final output, GenLayer randomly selects one of them as a leader to propose a version of the output that will be evaluated by the other validators - this mechanism is detailed in the Optimistic Execution section of this paper. The evaluation happens through a mechanism we term "Equivalence Principle", and it gives Intelligent Contract developers control over how the outputs are compared by the validators.

We propose a mechanism whereby the developer of an intelligent contract can specify the equivalence principle E for each non-deterministic call. It works as follows - given equivalence principle E, the proposed output of a Leader O, and a comparable output O', execute an LLM prompt of the following form:

```
Given the equivalence principle {E}, decide whether the
   following two outputs can be considered equivalent.

Output 1: {O}

Output 2: {O'}

Respond with: TRUE or FALSE
```

This decision will be used by each validator in the consensus process to evaluate whether the output proposed by a leader is valid.

It is also possible to disable the equivalence principle check for a given non-deterministic call. This will result in completely trusting leader's output, but can save gas costs in scenarios where the extra validation is not possible, desirable or economically viable.

3.8 Gas Cost Computation

An essential part of any decentralized ledger is a system commonly known as Gas. Gas is a mechanism for rewarding the validators for their work maintaining the network and executing the submitted transactions.

Unlike a traditional deterministic distributed ledger virtual machine like EVM, where you can know the exact instructions that have been executed and assign a cost to them, each validator can end up executing a different set of instructions based on different non-deterministic outputs, and the cost of producing the outputs will also vary between the validators, hence GenVM needs a different model for arriving at the cost of a transaction.

We propose a model that is based on an auction between the validators that creates an incentive mechanism to prevent validators from artificially inflating their costs. As each validator is processing each transaction, they keep a track of their own costs, consisting of both the cost of executing deterministic instructions and the costs associated with calling a LLM. Since each validator can use a different LLM, their costs can differ, and validators are encouraged to reduce them.

Then, once a validator knows how much it cost them to process the transaction, they commit to their cost on-ledger without revealing the cost, by submitting a hash of it. This is required so that validators cannot see other validators' bids when making a bid. When all validators have committed to their costs, they all post a transaction to reveal the cost.

When all costs have been revealed for any given consensus round, every validator except the one with the highest cost is paid. This creates an incentive mechanism for the validators to optimize their costs and bid honestly, instead of artificially inflating their costs. While a validator can get unlucky on any given transaction and miss out on the reward, if their setup is competitive over many transactions they will on average be paid fairly.

3.8.1 Gas computation mechanism for a User

It is the responsibility of a user to provide sufficient gas to ensure their transaction is executed. They set the amount exceeding that and receive the remaining amount refunded to their account once the transaction is finalized. The wallet software for GenVM will need to assist a user in choosing the right amount of gas to provide - this can be done by either simulating the transaction or using some heuristics to guess the gas usage.

Gas in the GenVM system is a complex topic, and this paper only outlines the mechanism. We will provide a detailed breakdown of the Gas mechanism in a future publication.

4 Network Actors

Now that we have an execution environment for Intelligent Contracts, let's see how a decentralized network of validators can be designed to efficiently execute these contracts in a decentralized fashion. First, let's define the key actors in the model.

4.1 Validators

Validators play a crucial role in the functionality of the GenLayer network. They are responsible for executing the Intelligent Contracts, as well as double-checking each other's work through the consensus mechanism. Validators are rewarded for processing transactions, and risk being slashed for incorrect validations. Validators participate in a delegated proof-of-stake consensus, and a limited number of validators will be active at any given time.

4.2 Users

Users submit transactions to the network, which may include anything from simple transfers to deploying or executing Intelligent Contracts. A user is any controller of an EOA. Users pay transaction fees (in the form of gas) to validators for processing their transactions and to contribute to the upkeep of the network. Users can also become validators by staking GEN, the native utility token, thus actively participating in consensus and earning rewards through transaction processing and staking.

4.2.1 Proposers

While the same as users from the network's point of view, we foresee many applications built around a special type of user called a Proposer. For cost efficiency and performance reasons, in many applications it might be valuable to off-load most of the work off-chain to a set of special actors, whose job is to prepare and submit pre-processed inputs to Intelligent Contracts.

Consider the example of an Intelligent Contract for gathering the summaries of news articles about a certain topic. Rather than tasking GenLayer with the execution of such a complex multi-step task, it would be more efficient to gather and summarize articles off-chain, and only submit the proposed links and their summaries to the Intelligent Contract. The task of the Intelligent Contract would then be the much narrower and easier task of checking whether the provided summary is accurate, non-duplicative and so on.

5 Token

To align the incentives of various network participants, we've introduced a native utility token called GEN. This token serves several key purposes:

- Paying gas for transaction execution
- Validator rewards through staking, and penalties through slashing
- Bonding for appeals
- Spam prevention

More details about the token will be provided in the upcoming GenLayer Tokenomics whitepaper. For the purposes of this paper, it is sufficient to assume a token exists which has value to the participants of the network.

6 Consensus: Optimistic Democracy

Since the execution of an Intelligent Contract call can be non-deterministic, the role of a consensus algorithm in GenLayer is somewhat different from most other DLTs. In a traditional decentralized ledger, the execution environment (e.g. Ethereum's EVM or Solana Runtime) is deterministic. The function of consensus in these systems is to agree on the ordering of transactions to prevent the double-spend attack, but the validity of each transaction is trivially verified and not in question.

In contrast, transaction execution in GenLayer is non-deterministic, because different validators will be running different LLMs, as well as because the execution of transactions may depend on external data from the World Wide Web, which changes over time. Therefore, the role of consensus in the GenLayer system is for a set of LLM-based validators to come to agreement about the very validity of the non-deterministic steps of each transaction, which will be based on judgements of the validators. We facilitate this through an incentive mechanism consisting of staking, optimistic execution, appeals and slashing.

Note that the implementation of Optimistic Democracy assumes an underlying communication layer for recording and ordering of transactions and enforcing the rules of consensus amongst the set of validators, which is addressed by a number of existing solutions and will not be discussed in this paper. Instead we will focus on the design aspects particular to the Optimistic Democracy consensus.

6.1 Staking

Validators that wish to participate in the consensus first need to stake some of the network token behind their node. They are rewarded for doing this with the gas from transaction execution as well as new token emissions. The steps to become a validator are:

1. Set up the validator with a set of keys
2. Delegate the platform token to it

Validators also stand to lose some of their rewards as well as stake as a penalty in the case of appeals, or liveness violations. This is the key motivator that ensures validators must maintain good performance, in terms of execution speed as well as quality of their non-deterministic responses.

To limit the total number of validators and provide all users with equal and fair access to staking yields, the system will provide the ability for users to delegate their stake to validators, a system known as Delegated Proof-of-Stake (dPOS) [8] . The total stake of a validator, combining their own as well as delegated tokens, must be high enough to land them in the set of active validators. Validators get to keep a small percentage of the delegators' rewards, and delegators can participate in securing the network without the complexity of running a validator.

6.2 Optimistic Validation

LLM executions are computationally expensive and time-consuming processes. To achieve a high level of performance with minimized costs, we propose the use of optimistic validation of transactions, where each transaction is validated by a small subset of validators at first.

More specifically, for each transaction, initially N validators are randomly selected from the entire set of the active validators, with one of them randomly designated as the leader. The randomness is based on a Verifiable Random Function(VRF) [9] and the odds of any given validator being selected for a transaction are proportional to their total stake.

The number of validators assigned to a transaction is always odd, so that the system can always come to a majority decision. The default minimum number of validators for a transaction is 5, however a user can select a higher number of initial validators and pay an accordingly higher gas fee if they desire an initial result that is less likely to be appealed, up to the entire validator set if single round finality is desired. The developer can also set a higher minimum validator count for each method individually if desired.

Because each validator might produce different responses to the non-deterministic calls (consider the example task being to summarize a piece of text), it is necessary to select a leader who will be tasked with providing a single proposed final output that the other validators will evaluate for quality. The algorithm for the consensus protocol is defined in Algorithm 1.

Algorithm 1 Consensus Process for Transaction Validation

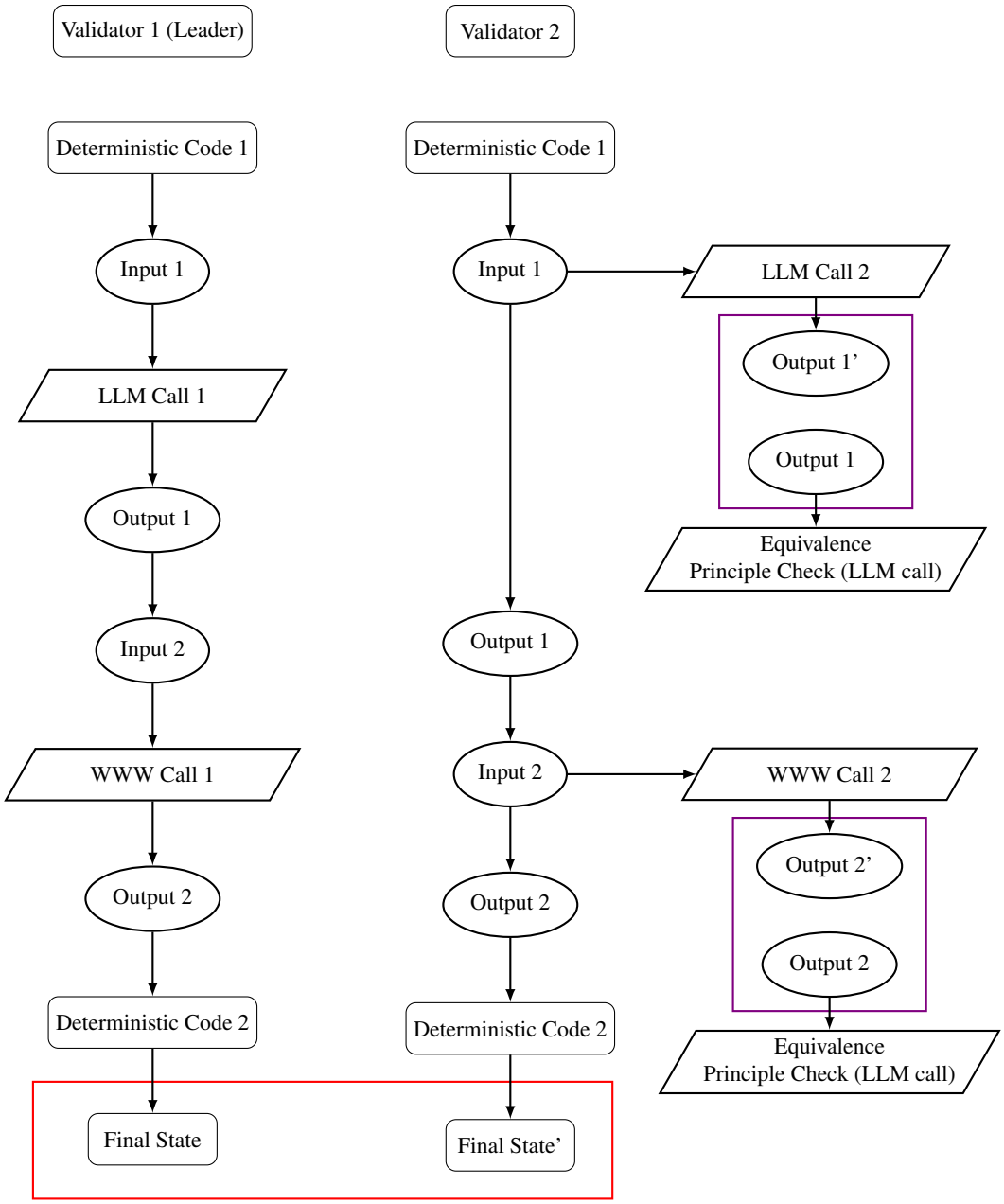
Input: Transaction T, Set of Active Validators V, Set of Active Validator Stakes K, Equivalence Principles E
Output: Transaction Receipt or Undetermined or Deterministic Violation
procedure SELECTVALIDATORS(V, K)
    $N \leftarrow$ Number of Validators (odd, default minimum 5)
    Randomly select N validators from V proportional to their stakes K using a VRF
    return Selected Validators
end procedure
procedure EXECUTETRANSACTION(V, K, T)
    $V_{sel} \leftarrow SelectValidators(V, K)$ ▷ Select Validators for Transaction
    for $V_{sel}[i] \in V_{sel}$ **do** ▷ Iterate through validators, excluding the initial leader
        $L \leftarrow V_{sel}[i]$ ▷ Select current leader
        Leader L executes T and posts proposed transaction receipt containing:
        Final state change ΔS
        M Inputs I_m to non-deterministic calls
        M Outputs O_m to non-deterministic calls
        for $V_{sel}[j] \in V_{sel} - L$ **do** ▷ Iterate through validators, excluding the current Leader
            Execute T using Leader's outputs O_m for non-deterministic calls to produce final state change $\Delta S'$
            if $\Delta S' \neq \Delta S$ **then**
                return Deterministic Violation
            end if
            for Each Non-deterministic Input I_m **do**
                Produce Output O'_m using I_m
                Compare Outputs O_m and O'_m using the Equivalence Principle E_m
            end for
            Commit to and share their vote (Agree/Disagree with Leader)
        end for
        Count votes
        if At least $(N-1)/2$ validators agree with Leader **then**
            return Leader's Transaction Receipt
        else
            Designate the next validator as the new Leader ▷ Rotate to the next validator in sequence
        end if
    end for
    return Undetermined ▷ If consensus is not reached after all validators have attempted
end procedure

Let's look in more detail at how both the leader and the checking validators execute a transaction. The leader executes regular code, until they hit a non-deterministic call - executing a text prompt or reading data from the web. The input I_1 to that call is deterministic - it is entirely derived from the transaction data and the code instructions. The leader's GenVM remembers both the input I_1 and output O_1, and will persist these in the transaction receipt, and so on for any further non-deterministic calls. During the execution the code can modify the internal state of the Intelligent Contract. At the end of the execution, GenVM serializes and persists all the state changes that have happened during the execution.

When a validator wants to check the output of a transaction proposed by a leader, they do the following steps. First, they execute the entire transaction substituting any non-deterministic calls with the leader's proposed outputs O_n. With the same outputs, the entire execution of a transaction is deterministic, and should produce the exact same state delta - if there is a mismatch, either the leader or the validator is running an incorrect version of GenVM. This is a serious violation that should never occur by accident. If a majority determines this is the case, the offending validator is removed from the active validator set.

If the final state change matches, the validator then proceeds to independently validate each non-deterministic execution using the Equivalence Principle defined for each call. More specifically, they use the same inputs used by leader In to produce their own proposed outputs O'_n. Then they compare each leader's output O_n with their output O'_n using the Equivalence Principle E_n defined for that call. Every output must pass the equivalence principle comparison. Since these are independent checks, they can be executed in parallel for higher performance. If and only if all equivalence checks pass, the validator considers the execution of a leader valid and votes accordingly.

See below a diagram illustrating this process. The purple boxes represent the Equivalence Principle checks and the red one represents the final state check.



One can also note that there are many steps of the process which can be executed in parallel for improved performance. First of all, each validator can execute all of the Equivalence checks in parallel, as they are independent of each other and only depend on the Input I_n fixed by the leader. Furthermore, it would also be possible to eagerly validate each of the leader's outputs before the full transaction is executed. The leader would have to post their input I_n and output O_n as soon as it is complete even as they continue to execute the rest of the transaction.

6.3 Liveness

In the context of a DLT, liveness says that all transactions originating from honest account holders will eventually end up included in the ledger. [12, p. 5] This means that the protocol can progress even when some of the validators are malicious and are attempting to disrupt the operation of the network, as long as the honest majority assumption holds.

To ensure liveness, the selected validators have a limited amount of time to submit their response to the network. Validators do not have a choice - if they are in the active validator set and they are selected to validate a transaction, they must participate. If they fail to do so, a new validator is randomly selected for the transaction, and the failing validator is penalized by slashing. If the validator misses several assignments in a row, they will be removed from the validator set.

6.4 Undetermined transactions

Because the execution of GenVM transactions can be non-deterministic – an LLM prompt can be written in a way that produces output that are not strictly equal, and data on websites can change – it is possible to have a transaction for which no consistent result can be determined, as each validator will produce a divergent output.

Consider the case of creating an Intelligent Contract based oracle to extract an exact value from a site which returns a random number every time it is loaded. Each validator will retrieve a different number, and they will be unable to find a leader whose answer they can all agree to.

If during the initial consensus or a later appeal process the validators are unable to find a leader's answer a majority can agree upon, the transaction enters a special state called "Undetermined". This means no state is changed based on the execution of the transaction, as the validators couldn't agree upon an answer.

An undetermined transaction can still be appealed, as it is possible the previous set of validators erroneously failed to arrive at a consensus.

Note that it is possible to write an Intelligent Contract which gets a random value without the transaction being undeterminable, it requires the Equivalence Principle for that non-deterministic call to be disabled, or set to a broad condition e.g. "It has to be a number". In this case, that kind of contract execution becomes effectively unappealable, so the developer is handing full authority to the initially selected leader executing the transaction.

6.5 Appeals

The optimistic execution mechanism is kept in check through a mechanism of escalating appeals. If anyone disagrees with the decision of the initial set of validators, they can submit an appeal request for the transaction during the Finality Window.

In each round, a new leader is selected and the number of non-leader validators is doubled - this way, at each round there are a sufficient number of validators to overturn the decision, and full finality can be reached in a few rounds of appeals.

The process for appellate validators is the same as for the non-leader validators of the initial execution. The votes of the appellate validators are added to those of the initial validators, and a transaction is considered valid if 50% or more validators (majority if you include the leader) approve the execution.

To prevent abuse of the system, the appellant must put a bond behind their appeal request. If the appeal is successful, the appealer recovers their bond plus a reward. This will create an incentive for actors in the network that search for potentially incorrect transactions. If the appeal fails, they lose their bonded tokens.

Meanwhile the validators who end up having voted against the majority at the end of the finality window do not receive any fees and get their stake slashed - this incentivizes validators to use sufficiently powerful LLMs that make good decisions reliably.

When an appeal for a transaction is started, it pushes back the finality window of the original transaction, so that there is an opportunity to further appeal it. As each appeal round doubles the number of validators, it only takes a few rounds - 6 in a network of 100 validators - to involve the entire network, which sets a natural limit on the maximum finality time.

6.6 Finality

To allow external systems to accept a certain output as final, the system has a limited window for submitting any appeals from the initial execution of a transaction. After this window has passed, the result of the transaction is considered final, and systems requiring the highest level of certainty should wait for this window to pass before accepting the result.

Finality Window: The duration after submitting a transaction during which it is possible to submit an appeal request for a transaction.

Alternatively, in scenarios where fast finality is needed - for example, a DAO, which is set up to only execute it's decisions upon finality, needs to make an emergency decision - it is possible to pay the gas cost upfront to have the entire validator set participate in the consensus for a transaction right away. This way it is possible to achieve fast finality while paying the necessary cost.

Fully deterministic transactions, or transactions for which all non-deterministic equivalence principle checks have been disabled, can have a much shorter finality window, as catching an incorrect deterministic execution is trivial compared to validating the non-deterministic outputs.

In this paper, we leave the exact length of the nondeterministic and deterministic finality windows undefined. These will be determined after more extensive simulations and testing, and detailed in a future publication.

6.7 Gas for appeals

The gas costs for an appeal can be covered by the user, the appealer or any other third party. When a user submits a transaction, they provide the initial gas plus an optional tip to cover the cost for potential appeals. If an appeal request is submitted for a transaction without sufficient gas, it will not be processed by the validators, however anyone can supply additional gas to ensure an appeal process is possible.

As the developer of an Intelligent Contract is ultimately responsible for choosing who their users are and what the odds of an appeal are for their contract, they can set the minimum tip required for each public method of their Intelligent Contract to ensure a certain level of appealability. A transaction attempting to execute this method will if the minimum appeal required by the developer is not met. This allows an application to choose the required level of security.

6.8 Interactive Consensus and Appeals

In the approach outlined above, each appellate validator comes to their conclusion independently. Furthermore, the reward system is set up such that only the voters of the winning majority are rewarded and the others are penalized. The combination of these factors incentivises a validator to be set up such that their vote is aligned with the majority of other voters. While this system works to prevents validators from using less intelligent LLMs than the current standard, it also incentivises them to produce an answer that aligns with the majority, even when possibly incorrect - this is a limitation of the design proposed above.

Research shows that roundtable discussion between multiple diverse agents improves the overall quality of answers [10] and that "debating with more persuasive LLMs leads to more truthful answers" and "arguing for the correct answer provides an advantage to debaters, and this advantage increases with persuasiveness." [11]

A possible future improvement on the system would be to allow them to communicate with each other during the consensus and appeal process, by submitting their arguments for consideration and thereby having the ability to persuade other validators of their case. This would enable an injection of additional intelligence into the appeals process and induce upward pressure on the intelligence level of all validators - if a more intelligent validator is capable of convincing other validators, they can succeed in bringing majority to their side. As a result, it becomes profitable to adopt better LLMs

as they can make better decisions and importantly convince the other validators about their case. However this direction introduces complex feedback loops and potential security risks, so in this paper we leave it as an interesting direction for future research.

6.9 The Effect of Appeals on Transactions

Non-deterministic transactions are subject to being appealed and recomputed during the normal consensus procedure, which, while infrequent, is an expected part of the normal functioning of the GenLayer system that allows correcting for errors and aggregating the intelligence of all validators.

It is important to consider how the GenVM interacts with the appeals process, in particular, the implications of a transaction being successfully appealed.

6.9.1 What happens to a transaction when it's appealed

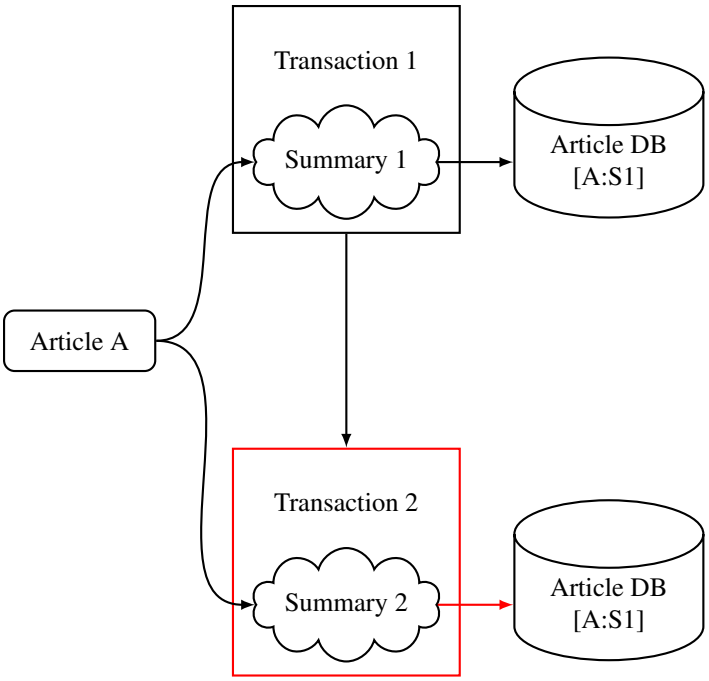
What does it mean, exactly, for a transaction to be appealed? When a transaction passes the first round of consensus, the output is a non-final transaction receipt that contains, among other things, the inputs and outputs of every non-deterministic call (calls to LLMs and reading data from the web) as well as the final changes to the state of the contract that was executed - for example, the value of some persisted variable was changed from 42 to 1337. This execution has been proposed by a leader and accepted by a majority of the initial set of validators.

It is however possible that an outside party observing this transaction might conclude that one or more of the non-deterministic outputs has been produced incorrectly, and the validators have come to the wrong result. This could be due to an honest mistake or malicious act - it is impossible to tell these apart in a foolproof fashion.

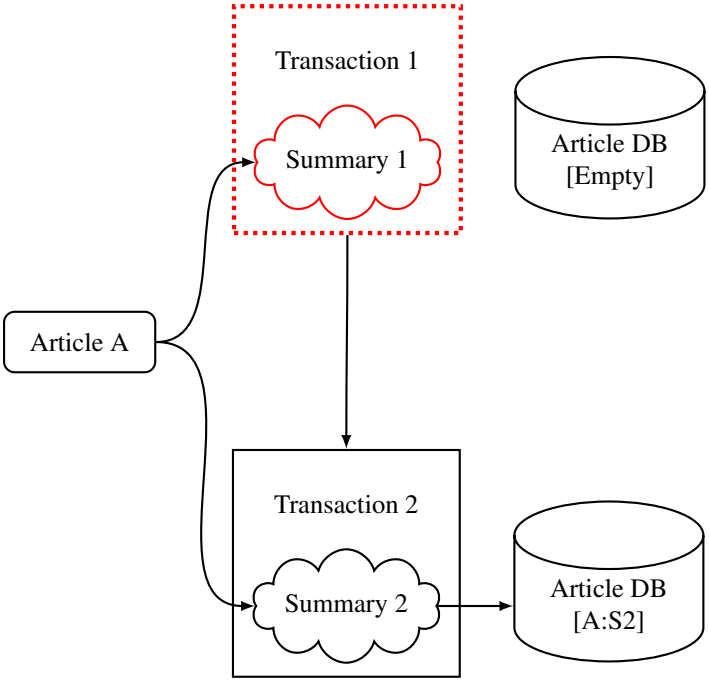
When a transaction is successfully appealed going through the steps described in the Appeals sections of this paper, the system ultimately settles at a new transaction receipt with different final state changes than initially proposed - either none if the transaction fails or a different set of state updates.

Dependent Transactions While simple enough in isolation, this poses a challenge when you introduce dependent transactions. What is a dependent transaction? It is an execution of the Intelligent Contract which depends on a state change in a previous transaction.

Consider the example of an intelligent contract that checks submissions of summarized news articles and only stores those that haven't been submitted before. User 1 initially submits transaction T1 with their summary S1 of article A, and it is accepted. User 2 submits transaction T2 with their summary S2 of the same article A before, but because it is a duplicate the summary will be rejected.



However, let's say afterwards the decision to accept summary S1 is appealed and transaction T1 is recomputed to not accept summary S1 (for example because the summary was determined to not be of sufficiently high quality). In this scenario transaction T2 should also be recomputed, as summary S2 is no longer a duplicate since there is no stored summary for article A anymore, and therefore Transaction 2 should succeed.



Consider that there can be an arbitrarily large number of dependent transactions for any transaction (limited by the throughput of the blockchain and the size of the finality window).

A naive solution might be to not allow the contract to execute a new transaction until there are no pending non-finalized transactions for that contract. While this would surely solve any issues related

to dependent transactions, it would also mean that each contract would only be executable once per finality period, which would severely limit the performance of the network as a whole.

It is preferable to allow the system to optimistically execute transactions, while accepting the risk of recomputation. The transactions would be recomputed in the order they were submitted, one by one optimistically, without waiting for the finality of the previous transaction just like during the normal operation of a network. The recomputed transactions only require the configured minimum number of validators for each transaction, as they have not been appealed themselves.

A user needs to provide sufficient gas to handle a potential recomputation caused by an upstream transaction being appealed - this can be estimated based on the number of preceding non-final transactions. If there isn't sufficient gas, the transaction will fail instead with an "out of gas" error.

There are ways at the system level to make this recomputation more efficient and reduce the gas cost - since the inputs and outputs of non-deterministic calls are stored in the transaction receipt, when the transaction is re-executed by the leader the input hasn't changed, there is no need to recompute it and the same cached value can be used.

It will ultimately be the responsibility of the developer of each Intelligent Contract to design applications that minimize the likelihood and cost of recomputation of downstream transactions for their users.

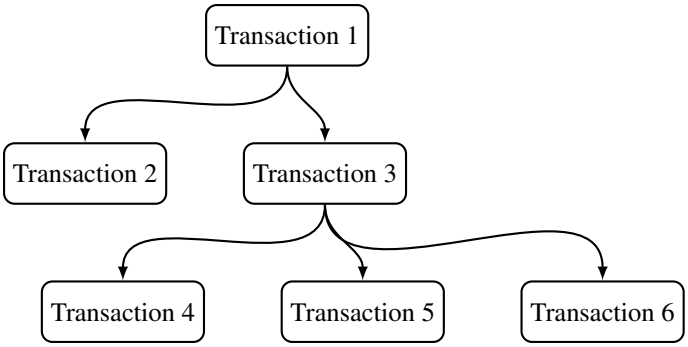
6.9.2 Contract to contract interaction

A similar and highly related set of complexities appear when an Intelligent Contract invokes another Intelligent Contract, either to read its data or execute some action that changes the contract's state.

Reading from another contract First, let's consider the case of reading data from another contract, by calling a read-only method on that contract that cannot change the state of that contract. Take as an example an Intelligent Contract that aggregates the latest news from a website, we'll call it the News Aggregator. This contract might offer a read-only method that returns the most recent 3 articles about a given topic.

Let's work through an example of building a second contract - an agent which will try to make a decision based on the latest news returned from the News Aggregator contract. Consider that the news aggregator will have some news articles that will have been accepted with finality - the transactions that added those have passed their finality window - and some that will still be in their appeals window and might still be reverted.

Naively, we might start by reading the latest non-final state as if it was final. This however means our transaction becomes dependent on all non-final transactions of the News Aggregator - if any of them are successfully appealed, the decision of our contract would have to be reconsidered as well using the changed state. Allowing for these kinds of dependency relationships would lead not only to a linear chain but a Directed Acyclic Graph (DAG) of dependent transactions that need to be recomputed after a transaction is appealed. The amount of computation required to work through this would bring the network to a halt in a scenario of large amounts of dependencies.



Instead, we propose to break the chain of direct dependencies, instead offering the developer two options:

1. Read the latest final state from the News Aggregator. This will only return news articles that have been finally accepted in the contract and cannot be appealed anymore
2. Read the latest non-final state, but accept the results as final at that moment in time. Essentially, allow me to make the decision based on the latest news articles even if they aren't final, accepting that it's the best available information at the moment, and if any of them are appealed, my decision doesn't need to be recomputed

Both options break the direct chain of dependencies and thus the tree of re-computation required if a transaction in the News Aggregator is appealed.

Writing to another contract A very similar problem appears when you consider an Intelligent Contract that wants to call a method on another Intelligent Contract that will change the contract's state, instead of just reading it.

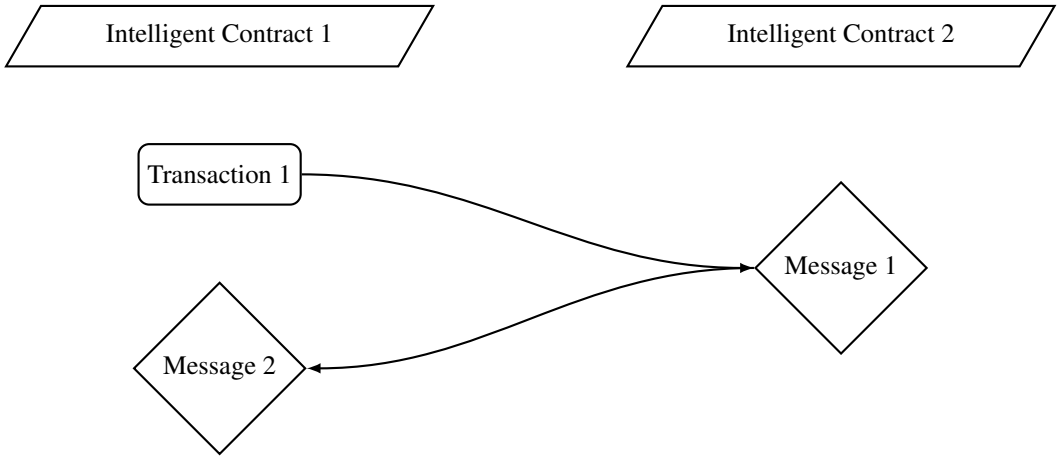
Imagine that there is an agentic Intelligent Contract that holds some tokens. The Intelligent Contract is invoked in a transaction and comes to a decision to sell the tokens on a decentralized exchange on GenLayer, which is a separate Intelligent Contract.

After some time has passed and many other trades have happened, there is a successful appeal of the agent's decision to sell. Should the agent's trade be reverted as well? This would require all subsequent trades that happened on the exchange to be recomputed as well. Given that the likelihood of a successful appeal is always substantial, especially if there is a malicious contract involved, we don't believe this is desirable.

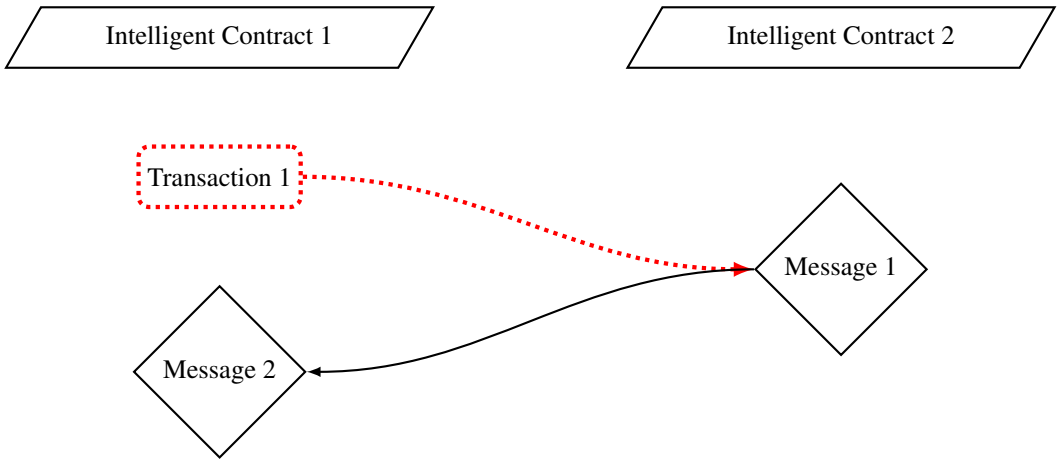
First, note that a transaction that consists of entirely deterministic interactions between contracts can be executed atomically, as the entire execution will be final immediately. Deterministic methods are executed just like they would in the EVM. Deterministic methods can call other deterministic methods atomically, meaning contract A can call contract B can call contract C in a single transaction that is executed all at once.

Regarding non-deterministic contract-to-contract interactions, we propose a system such that a non-deterministic method can only interact with other intelligent contracts through what we term "Messages".

Messages A message is similar to an EOA transaction, except that it is created by the execution of an Intelligent Contract instead of a user. It is created as a side-effect during the execution of a transaction by the leader. It contains details such as the target Intelligent Contract address, the method to invoke as well as the parameters.



A message, once emitted, is non-revertible - if an intelligent contract decides to emit a message that instructs tokens to be sent to the decentralized exchange, even if the decision is later appealed, the tokens will have been irrevocably sent and sold. The validity of the message emission is backed by the emitted transaction receipt.



Depending on the needs of the application, the developer of the non-deterministic contract can choose when a message is emitted:

1. As soon as the initial round of consensus is completed, or
2. After finality for the transaction has been achieved.

This lets the developer choose between fast decisions that run the risk of being appealed and changed, or slow but guaranteed to be final decisions. Like in the case of reading from another contract, this breaks the dependency chain between independent non-deterministic contracts and prevents a cascade of re-computation in the case of an appeal.

6.10 Slashing

Slashing is a mechanism to punish validators for behavior that is detrimental to the network. Validators can be slashed for several different reasons:

1. If they vote against the majority during a transaction execution at the end of the appeals process
2. If they miss their transaction execution window
3. If they miss their appeal execution window

The exact amount slashed for each violation is a tunable parameter and will depend on the economics of the token.

Slashing is applied on a transaction-by-transaction basis, and only becomes final after the finality window closes.

6.11 Unstaking

When a validator wants to stop participating in the consensus, they need to go through an unstaking procedure, which consists of the following steps:

1. Validator submits a transaction to initiate unstaking
2. Validator is removed from the set of active validators which are assigned transactions and appeals to evaluate

3. Validator needs to wait the finality period, to make sure their balance is final and none of their transactions can be appealed
4. Validator and all their delegators are allowed to withdraw their stake

This process is designed to make sure the validator's balance is final after taking into account any potential appeals that have not yet been resolved, and ensures the integrity of the network.

6.12 Scalability

In this section we will discuss the scalability of the Optimistic Democracy consensus mechanism. In the context of distributed ledger technology, scalability refers to the system's ability to handle a growing amount of work, particularly in the number and complexity of transactions it can process. Two key aspects to consider are transaction throughput - the total number of transactions the system can process per second (TPS) - and latency - the time taken for a transaction to be executed and confirmed on the ledger.

6.12.1 Latency

The latency of a transaction in GenLayer is composed of 2 key components:

1. The pickup time - the time from when a transaction is submitted to when it is picked up for execution
2. The execution and validation time of a transaction, until a transaction receipt is proposed and accepted by the first stage of validation

Note that we do not consider the Finality Window as part of latency, as the network can optimistically execute further dependent transactions even while the previous transaction is not final, however for certain use cases this will be the key factor determining latency, in which case it should be added to the calculation.

The pickup time is determined by the general availability of the validators randomly assigned to the transaction, plus the length of time to process any previous transactions the current transaction is dependent on, as such transactions can only be executed in order.

Independent transactions can be executed in parallel with each other, meaning that the latency can remain constant even as the throughput of the network grows, until such time that all validators in the network are busy processing transactions, at which point transactions are processed in the order they have been submitted.

Given current technology, the key component in the execution time of Intelligent Contract transactions will likely be the computation time of LLM calls, by many orders of magnitude. Furthermore, the validation time of a transaction requires at least twice as much time as a single execution - first, for the leader to execute and propose a transaction receipt, then for the additional validators to execute their own versions of the non-deterministic calls to compare with leader's output via the Equivalence Principle.

In conclusion, the key factors determining transaction latency in GenLayer will be the performance of LLMs, as well as the total number of validators in the network determining the average queueing time. As technology improves and they become faster, it will improve the execution and validation time of any individual transaction, as well as the total throughput of the network.

6.12.2 Throughput

Throughput of a network is the total number of transactions it can process in a given unit of time, typically measured in Transactions Per Second (TPS). It is never a completely straightforward measure, as in all but the most simple networks different transactions can be of varying complexity and size, however we can identify the key variables affecting the throughput of GenLayer.

First, consider that before any appeals are involved, the number of independent transactions the network can process in parallel scales linearly with the number of validators, as only a small subset of the validators need to process each transaction. This is limited in practice only by the throughput of the underlying communication layer through which the consensus mechanism is enforced between the various validators - we will discuss this more in an upcoming paper.

Transactions with a single account form a dependency chain and have to be processed in the order they were submitted in, so the throughput for dependent transactions does not scale with the number of validators, instead only the processing time of each transaction determines the throughput.

Beyond this, introducing appeals involves more and more validators at each stage of escalation. This means that the throughput will be limited by the frequency of appeals, as well as the number of appeal stages in an average appeal. This will depend greatly on the type and complexity of the Intelligent Contracts deployed on GenLayer, as well as the capabilities of the LLMs used by the validators.

Further limits on the throughput are introduced by dependent transactions in the case of an appeal. If a transaction is successfully appealed, all downstream transactions of the same contract need to be recomputed as well. Because of the messaging system described in section 6.9, the chain of dependent transactions cannot branch and its size is limited by the linear throughput of the network and the Finality Window. While these transactions are processed, the involved validators are prevented from working on other transactions, however the validator set size is the minimum required for each of the transactions, therefore the network isn't blocked and other validators can work on other transactions in the meantime.

There is another consideration which might require reducing the total number of validators beyond what the communication layer can handle, which is the cost of achieving quick finality. It is desirable that in urgent cases it is possible for a user to pay the cost upfront to have a transaction validated by the entire network, such that the transaction is final after the first stage of consensus, as there would be no additional validators to appeal to. As this involves all validators in the network, the cost of this scales linearly with the total number of validators.

In summary, the throughput of the network will be determined by a few factors such as the number of validators, which will have to be selected to balance the considerations of increasing parallelism versus the cost of full finality and any practical limits of the implementation, as well as the frequency of appeals and the average length of dependency chains.

7 Risks and Security

Large Language Models are a novel technology that developers are only just beginning to incorporate into their applications, and they, along with other neural network (NN) architectures [16], are susceptible to novel kinds of attacks. A decentralized ledger is a highly adversarial environment where the value held by each application effectively becomes a honey-pot for attackers from the entire world, making the security of the system absolutely critical. In this section we'll discuss some of the attack vectors and how they are addressed in the design of GenLayer.

7.1 Prompt Injection

Prompt injection is a technique used in the context of interacting with AI models, especially those processing natural language inputs, where an external entity (an attacker or user) crafts an input (the "prompt") in a way that attempts to manipulate the model into executing unintended actions or disclosing information it's not supposed to. In the context of GenLayer, this would involve making an Intelligent Contract invocation arrive at a decision unintended by the developer.

A lot of the responsibility of preventing prompt injection attacks will fall on the developer of an Intelligent Contract. First, wherever possible, a developer should avoid direct text input from a user - instead any prompt to the LLM should be crafted by the code of the Intelligent Contract, only taking limited and sanitized inputs from the user. Similarly, when reading data from the web, if possible

reading from websites consisting of user-generated content should be avoided in favor of reputable and secure sources.

Of course it is not always possible to avoid taking direct input from users. In these situations the developer should take great care to sanitize any input from the user, and apply techniques such as post-prompting, user input tagging and more to guard against adversarial inputs that convince the model to come to an incorrect conclusion. Where possible, GenLayer will integrate industry best practices in the validators.

7.2 Universal Attacks

Similar to prompt injection, a universal attack involves an adversary modifying the prompt sent to the LLM to produce undesired (from the point of view of the creator of the Intelligent Contract) outputs, however unlike prompt injection, which is typically crafted by hand and is more of a logic attack on the model, universal attacks involve systematically modifying the prompt with random characters to find a prompt through sheer brute force that produces the required adversarial output. [13]

It is less possible for a developer of an Intelligent Contract to design their application to defend against such an attack, as it is a low level vulnerability of the underlying LLM, however the network architecture of GenLayer which consists of validators running many different LLMs makes it resilient against such attacks. While there is some transferability of these adversarial attacks between different LLMs, they are far less likely to work across different models with potentially different fine-tunes, and this should be more and more the case over time. Furthermore, in the next section we outline an approach to further guard the network against these kinds of attacks.

7.3 Greyboxing

We propose a novel approach for minimizing the risks of adversarial inputs we term "Greyboxing". It is a play on the term “black-box” which in the LLM world is used to refer to models which are not publicly released and can only be used through an API, making the model effectively a black box which cannot be inspected.

Greyboxing consists of passing the direct prompt from an Intelligent Contract through a pre-processing stage intended to clean up, optimize and remove unexpected symbols from the prompt using techniques such as paraphrasing, retokenization, perplexity-based detection and more before passing it to the actual LLM for processing [14]. The exact setup of the parameters of the greyboxing stage should be initialized differently between each validator, effectively further diversifying the validator set and making it exponentially more expensive for a datamining attack to succeed.

Furthermore, GenLayer’s security is enhanced by its validator selection process, which uses a Verifiable Random Function (VRF) to randomly select validators. This randomness makes it difficult for attackers to predict and exploit the system, as the loss function used by validators changes with each transaction, complicating the development of effective universal attacks.

Defending against adversarial inputs is an area of active research and the LLM-agnostic approach enabled by the design of the GenLayer consensus algorithm enables and encourages validators to integrate new techniques into their greyboxing setup as they are discovered, making GenLayer antifragile and adaptable in the face of adversaries.

8 Use Cases

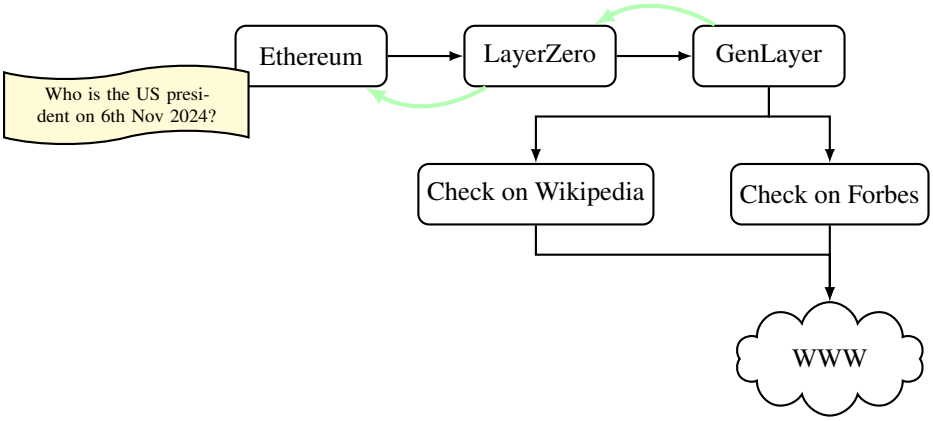
In this section, we will provide some examples of use cases that are enabled by GenLayer and Intelligent Contracts.

8.1 Intelligent Oracles

Using the capabilities of GenLayer, it is possible to create an Intelligent Contract which can answer arbitrary and complex questions about public data on the Web. A few examples of what would be feasible:

- Monitor news sites to answer the question "Who won the 2024 U.S. Presidential election" and resolve a prediction market
- Find out whether there is a drought in a given location to decide if an insurance contract should be paid out
- Look at crypto media and detect if a protocol is experiencing an attack, and if so initiate an emergency shutdown of the protocol
- Monitor the website of the Federal Reserve for the latest interest rate changes and adjust the interest rate accordingly in a decentralized lending protocol
- Check if a user has posted a specific message and read their follower count on a social media platform, to link their real-world identity on-chain

Furthermore, through inter-blockchain communication services such as LayerZero, these results can be transmitted to other smart contract platforms to execute actions such as resolving a prediction market.



8.2 World Database

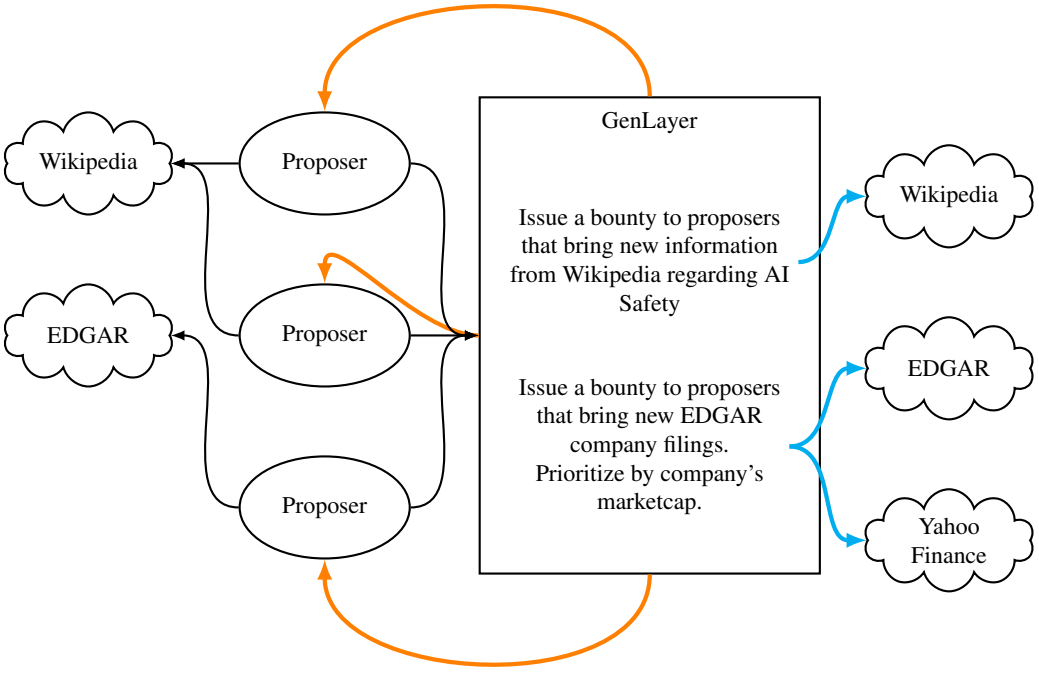
Imagine a freely accessible, trustless data-structure storing relevant world events. Off-chain proposers would be tasked with scouring the web for relevant data and preparing a summarized version for validation and storage on the network.

The proposers would submit their source as well as their summary of a relevant piece of information to the Intelligent Contract.

The intelligent contract then evaluates:

- Is the source already in the database, or has already been rejected?
- Is the summary accurate to the source?
- Is the new piece of information relevant to the information sought by the database?

If it meets all the criteria, it will be stored in the database, and the proposer will be rewarded with tokens by the World Database protocol.



8.3 True DAOs

A DAO, or Decentralized Autonomous Organization, is a type of organization with its rules encoded into a smart contract. DAOs aim to revolutionize how organizations are run, making them more democratic, transparent, and efficient. However, in practice, the on-ledger component of current DAOs is limited to aggregating the votes of token-holders.

Using the capabilities of Intelligent contracts, it would be possible to create the first truly Autonomous Decentralized Organizations, capable of responding and adapting to an ever-changing set of circumstances without human intervention.

Such a DAO could have a plain-text constitution outlining its goals and rules. It could hold assets, both in GenLayer and on other platforms through the LayerZero integration. It could connect to trusted data sources on GenLayer, such as the World Database outlined above, as well as on the internet.

It could make decisions to manage its treasury - buy and sell tokens, deploy them in other protocols.

It could issue bounties, evaluate and accept grant proposals, evaluate their completion by looking directly on the web and issue tokens as a reward.

To operate, it would have to be invoked - it could reward the invoker with tokens to cover their gas costs, when the transaction results in a meaningful new decision of the DAO.

9 Conclusion

In this paper, we introduced a new decentralized platform called GenLayer for running programs called Intelligent Contracts, which extend the concept of smart contracts by integrating them with natural language instructions and giving them the ability to read data from the web. With GenLayer, it is possible to apply the benefits of self-executing and self-enforcing transactions to a vastly broader set of use cases, continuing to expand the vision of providing an alternative to the traditional legal and financial systems suitable for the digital world, one which is equally and fairly accessible to

everyone in the world, humans and autonomous agents alike. Aside from immediate use-cases such as Intelligent Oracles, Trustlessly Verified Information Databases and True DAOs, we believe that GenLayer can become the legal system of choice for autonomous AGI agents to conduct business with each other.

References

[1] Vitalik Buterin. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf, 2014.

[2] Nick Szabo. Smart Contracts. https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html, 1994.

[3] Yuma Rao. Bittensor: A Peer-to-Peer Intelligence Market. https://drive.google.com/file/d/1VnsobL6lIAAqcA1_Tbm8AYIQscfJV4KU/view, 2022.

[4] Olas Network. Autonolas Whitepaper. https://olas.network/documents/whitepaper/Whitepaper%20v1.0.pdf, 2023.

[5] OpenAI (2023). GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.

[6] Johnson, D., Menezes, A., and Vanstone, S. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security, Vol. 1, No. 1, August 2001, pp. 36-63.*, 2001.

[7] Clément Lesaege, Federico Ast, and William George. Kleros. https://kleros.io/static/whitepaper_en-8bd3a0480b45c39899787e17049ded26.pdf, 2019.

[8] Daniel Larimer April 3, 2014 Delegated Proof-of-Stake (DPOS) https://how.bitshares.works/en/master/technology/dpos.html, 2014.

[9] Micali, S., Rabin, M., and Vadhan, S. Verifiable Random Functions. *In Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS '99).*, 1999.

[10] Justin Chih-Yao Chen, Swarnadeep Saha, Mohit Bansal ReConcile: Round-Table Conference Improves Reasoning via Consensus among Diverse LLMs *arXiv preprint arXiv:2309.13007v2*, 2024.

[11] Akbir Khan, John Hughes, Dan Valentine, Laura Ruis, Kshitij Sachan, Ansh Radhakrishnan, Edward Grefenstette, Samuel R. Bowman, Tim Rocktäschel, Ethan Perez Debating with More Persuasive LLMs Leads to More Truthful Answers *arXiv preprint arXiv:2309.13007v2*, 2024.

[12] Garay, J., Kiayias, A., and Leonardos, N. The Bitcoin Backbone Protocol: Analysis and Applications. https://eprint.iacr.org/2014/765.pdf, 2015.

[13] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and Transferable Adversarial Attacks on Aligned Language Models. *arXiv preprint arXiv:2307.15043*, 2023.

[14] Neel Jain et al. Baseline Defenses for Adversarial Attacks Against Aligned Language Models. *arXiv preprint arXiv:2309.00614v2*, 2023.

[15] Ashish Vaswani et al. Attention Is All You Need. *arXiv preprint arXiv:1706.03762*, 2017.

[16] Christian Szegedy et al. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.