Nexus: A New Standard for Decentralized Autonomy

The Talus Labs Team
hi@taluslabs.xyz

21 August 2025
v1.1

Abstract

Blockchain provides a well-tested and reliable environment for hosting high-value computation and assets. Equipping blockchain with modern AI capabilities can significantly enhance its functionality and user experience. In this work, we introduce the **Talus Agentic Framework** to standardize onchain services and coordination protocols, enabling the modular onboarding of AI services as onchain components. The modular design supports varying levels of onchain security, ensuring auditability and transparency in a decentralized manner with acceptable performance.

We implement **Nexus** to realize the Talus Agentic Framework on the Sui network, serving as a platform for building and utilizing onchain AI services with composability and secured financial capability. Developers can build two types of applications on Nexus: (i) Agent as a Service, which integrates secure onchain AI services or workflows to enhance existing Web3 applications; and (ii) Agent Marketplace, which uses onchain infrastructure to monetize AI services, such as amortizing AI computation costs or rapidly deploying and monetizing state-of-the-art AI technology.

1 Introduction

Blockchain technology is a decentralized and trust-minimized platform for providing global functionalities such as immutable storage, execution, and randomness (via a verifiable randomness function (VRF) [MRV99]). This paper defines an **onchain service** as a set of smart contracts that expose a specific functionality for external parties to use. In general, new functionalities available onchain can create new opportunities and improve the behavior of existing onchain services, enabling people to access cheaper and safer services more easily. One of the most promising capabilities to be extended is the availability of artificial intelligence (AI) services.

Current AI agents are autonomous software programs that perceive environments, use large language models (LLMs) to make decisions, and execute actions through tools to achieve specific goals. An agent can be tuned for specific tasks, and coordinating multiple agents with ordered steps to tackle more complex challenges creates a more promising approach—an agentic workflow [SEKK24]. At Talus, we believe AI agents have the potential to revolutionize the entire digital world with the help of blockchain technology.

Integrating AI agents with blockchain technology creates a powerful synergy:

Trust and Transparency: Onchain deployments bring transparency to AI execution processes with configurable trust. Since onchain execution inherits security from the consensus layer,

onchain services can provide functionality with transparency and auditable service quality. Users can also apply onchain verification to secure the outputs of an AI service.

Automated Finance: AI agents can automatically manage onchain assets, execute trades, and allocate resources with predefined rules and dynamic market conditions. On the other hand, blockchain provides a trustless financial infrastructure to monetize AI capabilities. Integrating these two technologies expands the subject and strategy space for decentralized finance (DeFi) and autonomous organizations, enriching the onchain ecosystem.

The combination of AI agents and blockchain technology can also introduce new possibilities to existing Web3 industries:

Finance: Since AI agents can automate onchain assets, this immediately implies that we can use them to implement autonomous trading systems and yield-generating strategies. These agents inherit security from onchain computation, creating a fair investment market that anyone can join transparently, with a minimized information gap.

Governance: AI agents can simplify decentralized autonomous organizations (DAOs) by preprocessing user requests and converting them into valid proposals with proper auditing and evaluation before community voting. AI agents make the entire process transparent and standardized, ultimately contributing to high-quality onchain collective decision-making.

Gaming & Virtual Worlds: AI-driven non-player characters and virtual assistants can interact directly with smart contracts in a predefined manner to offer dynamic in-game experiences. AI agents can transparently provide additional randomness and fairness.

1.1 Related Works

This section reviews two existing directions for onboarding AI services onchain: onchain AI model computation and offchain agents with wallets.

1.1.1 Onchain AI Model Computation

The most straightforward and ambitious way to integrate AI services onchain is to write all computations of the AI service into a smart contract. This idea aligns with the fact that most blockchain virtual machines (VMs) provide a Turing-complete computation engine, capable of executing arbitrary programs, including AI algorithms. With deterministic randomness onchain, all validators in the blockchain system can agree on the input and output of the model, thereby enabling the use of AI services. Due to computational resource limitations, most prior attempts have only implemented traditional machine learning methods (e.g., decision trees, support vector machines, etc.). One of the leading players in this field is *DecideAI* [Teaa], which successfully brought GPT-2.0 onto the Internet Computer (ICP) [Teab] network. Projects like *Tensorflowsui* [Tead] also provide infrastructure to run general deep learning models through smart contracts. However, hardware limitations make further improvement of model capability difficult, and it is extremely costly compared to offchain AI services, making its use cases very limited.

One variant of onchain AI model computation transforms the computation task into a verification task, using zero-knowledge proofs (ZKPs) [GMR85] to ensure onchain verifiability. ZKPs can guarantee that an offchain computation procedure correctly follows its onchain commitment. Therefore, we can compile the AI algorithm program into a provable circuit that allows an offchain AI agent to generate a proof for verifying its output. *zkML* [zT] is a pioneer in this direction, successfully

onboarding TensorFlow Keras models into circuits that can be verified onchain. As ZKP technology advances, more powerful verifiable onchain AI models will become possible.

1.2 Offchain Agents with Wallets

The second approach uses mature offchain agent solutions and exposes their inputs and outputs through an oracle system [ZCC+16, BCC+21]. An oracle system builds a bridge between onchain and offchain data. An offchain agent can use these oracle capabilities to communicate with the onchain state. Typically, the agent uses a wallet and an offchain secret to manage its onchain assets, and listens to onchain queries through an outbound oracle. Secure computation environments such as Trusted Execution Environments (TEEs) [CD16] or authenticated communication [CHH97] are widely used to ensure the security of this service. *ElizaOS* [WGN+25] is one of the leading solution providers in this direction. *ElizaOS* is an offchain agent solution that provides plugins for agent communication with the blockchain. The identity of the agent is represented through a wallet plugin. This offchain agent architecture provides flexibility in agent functionality without requiring heavy onchain development.

This strategy has a decentralized variant that replaces the single trusted offchain agent with an independent decentralized computation system. In general, a properly configured decentralized system can provide better security guarantees, although likely with increased latency. The decentralized network of agents can use various mechanisms, from auditor voting to consensus algorithms, to agree on the outputs and use a threshold signature scheme (TSS) [Sho00] to validate them onchain. *Ritual* [Teac] is a service provider following this approach, using a subset of computation nodes for computing model outputs. This subnet typically operates with independent security assumptions to ensure that tasks assigned to the system are correctly computed.

1.2.1 Limitations

Existing works have three main limitations:

Lack of composability: Making AI services composable with existing onchain services requires a standard protocol to define requirements (i.e., input schema), liability (i.e., output schema), and entitlement (i.e., payment in the crypto context). Existing agent solutions lack interoperability and composability, as they do not provide general input/output schemas or payment contracts between onchain callable services. As a result, onchain services struggle to interoperate and cannot efficiently reuse existing ecosystem tools.

Insufficient asset management capabilities: Onchain asset management is tied to the ownership mechanisms of onchain virtual machines, which typically rely on digital signature schemes. While modern AI models excel at decision-making and automated execution, they also introduce the risk of hallucination. This creates the need for fine-grained permission control for AI services managing onchain assets, which must be integrated into a general and composable framework.

Limited service quality choices: Onchain AI services face a fundamental trade-off between performance and security. Onchain computation is generally expensive and non-scalable compared to offchain alternatives, while offchain computation involves greater risks in terms of computational security. Applications must navigate this trade-off to optimize user experience according to market demands—a challenge that cannot be solved by a single solution.

One solution to the above issues is a standardized onchain protocol that allows secure interoperability between AI services and onchain services. AI services and other onchain services can be designed modularly and composed together through the protocol, where each component can have independent verifiability and cost considerations based on arbitrary business logic. In addition, we can use smart contracts to set up permissions for each module of the service, provide transparent and secure asset management, and minimize the trust over offchain private keys.

1.3 Our Contributions

We propose the Talus Agentic Framework (TAF) as a composable framework [BDKJ23, BMZ24] to define onchain services with interoperability. The Talus Agentic Framework can fully leverage different solutions for onboarding AI services with optimized cost and performance per user requests. Thus, we can provide real onchain AI agents and agentic workflow services. Our results are twofold:

1. We define the Talus Agentic Framework based on modular onchain services called Talus Tools. In our framework, a Talus Agent is an aggregation of standard onchain services (including those providing AI services) that support various underlying AI system implementations. Talus provides AI services in a tool-based solution, where onchain services (AI or not) are abstracted as Talus Tools and integrated with others through composable execution flows. We implement the Talus Agentic Framework on the Sui Network through the Nexus engine. Nexus enables developers to integrate state-of-the-art (SOTA) AI technologies with composability, censorship resistance, and a native payment protocol. Our design provides flexibility between security and performance, allowing other onchain services to integrate offchain components easily.

2. We formalize two types of products based on Nexus: Agent as a Service (AaaS) and Agent Marketplace. AaaS focuses on improving onchain services with transparent AI technologies through the Talus Agentic Framework. Web3 service providers can use Talus Agents as asynchronous components to enhance their offerings and functionalities with transparent and secure AI services. Agent Marketplace represents another type of service aimed at enabling closed-source AI service monetization through onchain infrastructure. Users can select one or multiple agents to complete tasks collaboratively or competitively in a decentralized manner using an onchain payment infrastructure and output verification.

This white paper is organized as follows: Section 2 explains the background knowledge of blockchains and AI agents. If you are already familiar with the topic, you can skip it. Section 3 explains the Talus Agentic Framework (TAF) as a solution to realize onchain agentic workflows, and Section 4 introduces Nexus as an implementation of TAF. Section 5 presents the two categories of onchain services Nexus aims to support. Finally, Section 6 explains our future directions.

2 Preliminary

In this work, we use the following notations:

Onchain service: A callable service based on smart contracts.

AI service: A service that exposes AI models for calculating inputs and returning outputs.

AI agent: An AI service that processes both user input and external data, plans and performs actions through integrated tools (iteratively), and provides outputs.

Onchain AI agent: An AI agent that can be triggered through a smart contract.

Figure 1 illustrates the scope and position of the Talus Agentic Framework (Section 3) and existing works.

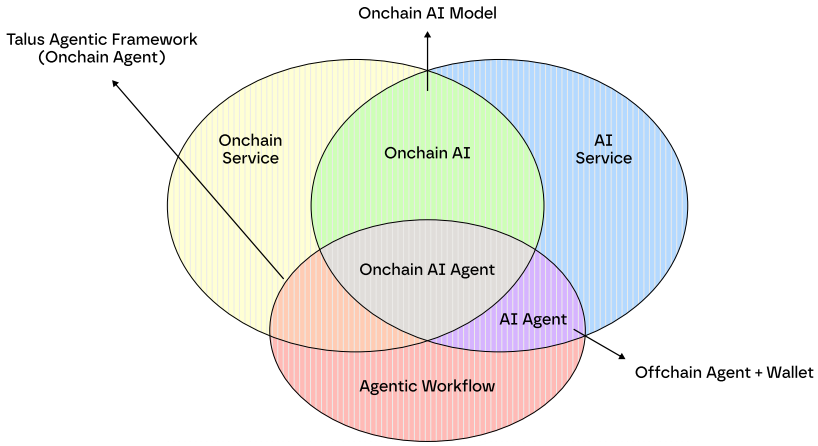

Figure 1: Position of the different concepts and related works

2.1 Blockchain Technology

This work uses a public blockchain as the coordination platform [PHdB22] for AI services. Public blockchains are typically structured in layers [ZXD+17]. In this report, we focus on the core software layers: the consensus layer and the execution layer. These layers work together to securely manage the onchain state (i.e., the ledger) and execute programs (smart contracts) with transactions (external inputs and commands) on a decentralized network (Figure 2). The security of the consensus algorithm and the determinism of the execution layer imply the security of services defined on smart contracts.

The consensus layer is responsible for agreeing on the ledger's state and the order of transactions across all nodes. It ensures that all participants (validators) reach a consistent view of valid transactions and blocks. Once a block of transactions is agreed upon and added, it becomes extremely difficult to alter records without breaking the consensus rules.

The execution layer is where blockchain programs and end-user applications reside. The core of the execution layer is a virtual machine (VM) that defines how the global state can be modified. Developers write programs for this VM as smart contracts to define business logic and interaction interfaces. In the public blockchain model, users interact with smart contracts through wallets to update the ledger. These smart contracts are transparently placed onchain and are executed faithfully, assuming a secure consensus layer. Therefore, blockchain enables secure decentralized applications.

2.1.1 Secure Onchain Service

This work assumes all users have an ideal wallet to interact with onchain services accessible through the blockchain execution layer. Therefore, a secure onchain service can be exposed as a secure decentralized application that satisfies blockchain-adapted safety and liveness [AS87]:

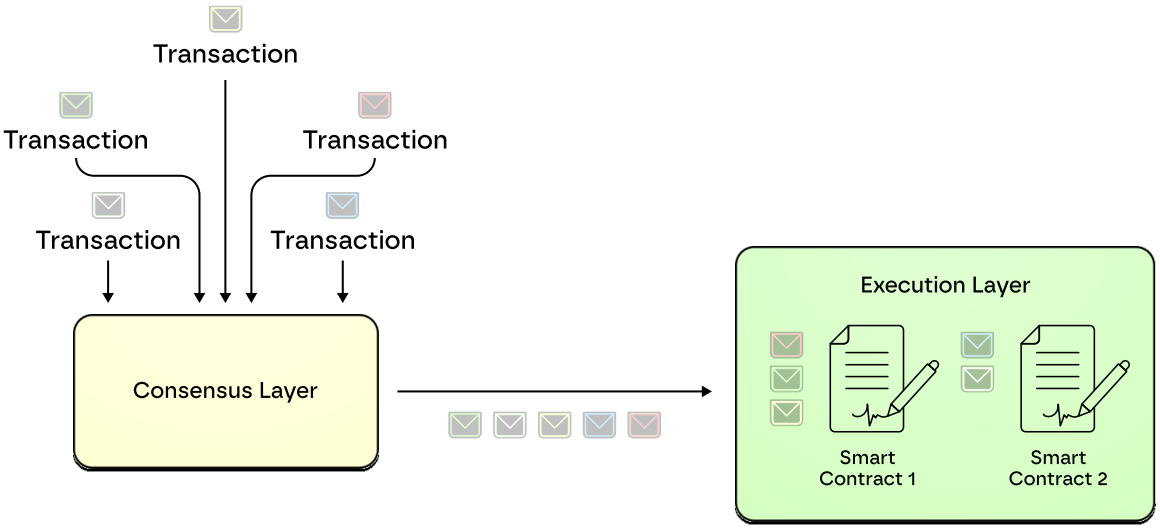

Figure 2: Functionalities of a blockchain

Safety: If an onchain service is triggered, its execution will always follow the contract where it is defined.

Liveness: If an onchain service is triggered, it will always output a result that follows the contract where it is defined.

Onchain services can be secured in three different ways, each offering different guarantees and trade-offs. Figure 3 gives an example of cost estimation for different security sources.

Onchain Computation: The most direct approach is inheriting the security of the consensus layer and the deterministic VM through onchain computation. Deploying the service logic as a smart contract naturally ensures safety and liveness. However, computation in the onchain VM is generally expensive and is measured through a gas system in most public blockchains.

Verifiable Computation: To reduce the onchain computation complexity, proof schemes have been introduced, which use a shorter verification procedure to ensure that offchain computation is correct. Depending on the statement to verify, onchain verifiable computation can use correctness proofs to ensure correctness or assume the computation is correct and open a window for fault proofs (optimistic proofs) [CSYW24]. The onchain verification guarantees safety and liveness, while the proof scheme ensures that the business logic has been properly computed. This procedure has two disadvantages: the offchain proof logic is often more complex than the original computation task, and the liveness of the offchain component requires additional mechanisms.

Trusted Computation: Trusted parties are introduced to ensure the security of offchain components. The most direct solution is to execute all offchain logic in a TEE, such as Intel Software Guard Extensions (SGX) or other confidential computing services. However, this does not resolve the liveness issue, which typically requires a decentralized system to improve the robustness of offchain components. Decentralization improves offchain logic, similar to how the

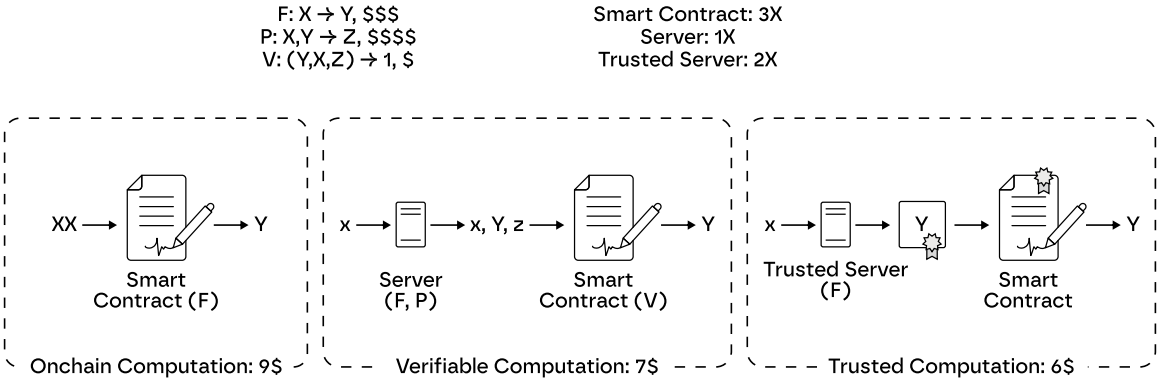

Figure 3: Cost of Security

consensus layer secures onchain logic, but on a smaller scale. Tokenomics forms a supplementary mechanism that further enhances trust under the assumption of rational participants. Well-designed token economics can ensure that all participants of a decentralized system are incentivized to behave honestly.

However, the trade-off between performance and security may vary across different services. Therefore, supporting as many possible security sources as possible reflects the compatibility of a framework for coordinating onchain services.

2.2 AI Agent

In recent years, an AI agent has been defined as an autonomous software entity with a Large Language Model (LLM) as its core reasoning engine and multiple tools or plugins for extended functionalities. For example, a conversational agent can interact with users via various interfaces, use the LLM for chain-of-thought (COT) reasoning, store conversation history in a database, and utilize external tools (such as a search index) to retrieve information from the internet. To make an AI agent service onchain, we must first understand its architecture and then determine the appropriate methods for onboarding each component as an onchain service.

2.2.1 Architecture

Modern AI agents [YZY+23] incorporate components analogous to a human problem-solver [POC+23], with four modules: perception (input via web, SMS, voice), cognition (e.g., LLM reasoning), memory (agent context and chat history in a database), and tools (external knowledge bases or functionalities). These components are orchestrated to resolve tasks given by users. The overall flow is iterative, as shown in Figure 4.

First, the agent receives input from the user (Step 1). This input can be a command, question, or request. The agent then initiates the perception phase, where Perception Tools evaluate the current state of the Execution Environment (Step 2.1). Next, the evaluations are parsed, meaning they are processed and transformed into a structured format that the AI can understand and reason over (Step 2.2). This may include extracting intent, filtering signal from noise, or identifying relevant context.

The agent then moves into the planning phase (Step 2.3), where it uses memory and context to determine a course of action. This could involve reasoning through a chain of steps, selecting tools,

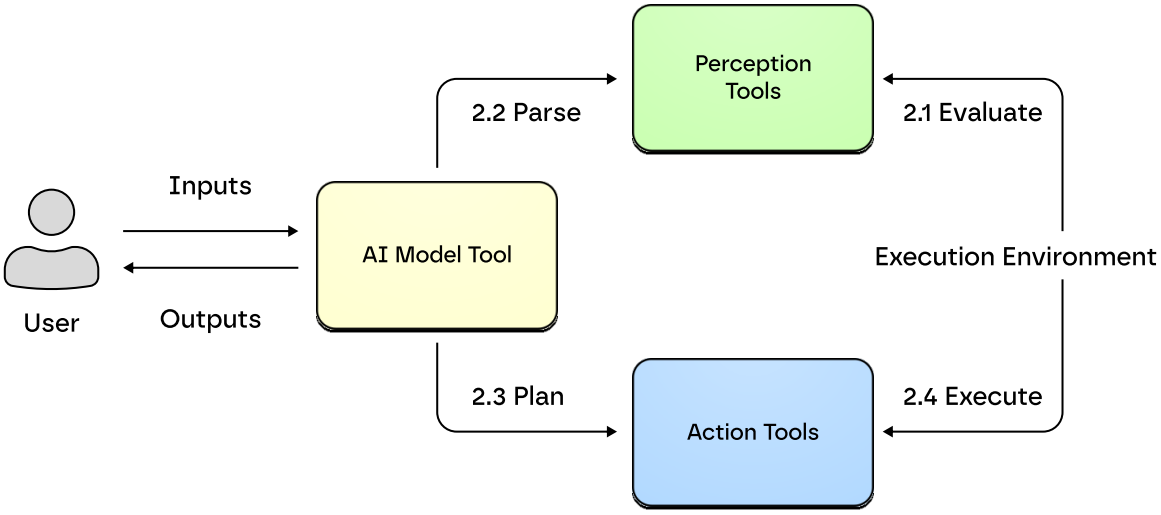

Figure 4: General Architecture of an AI agent.

or deciding how to proceed based on the parsed environment and user input. Following that, the agent triggers Action Tools to carry out the plan (Step 2.4). This could include executing smart contract functions, interacting with APIs, updating states, or any external operation within the execution environment.

The outcomes of these actions produce new data or changes in state, which are again evaluated by the Perception Tools, closing the loop and starting a new cycle of evaluation → parsing → planning → action. This continues until the agent has completed its task and delivers the final output back to the user (Step 3).

2.2.2 AI Agent Decomposition

Using the above architecture, an AI agent service can be divided into three functionalities:

Triggered By Inputs: This function corresponds to the perception phase, which defines and collects inputs for the AI service from the environment. This stage is critical for the service's security because if the service accepts malformed (or malicious) inputs, the outputs cannot be evaluated as promised.

Generate Outputs: This function is central to modern AI agents and represents their capability for autonomous problem-solving. The agent should be able to decide which subset of its tools to trigger and generate the corresponding inputs for those tools.

Trigger Tools: Given the decision and inputs, the AI agent service must trigger the appropriate tools and ensure they are executed correctly. If the action is to yield the final output, the agent must also ensure that the result is deliverable and meets the requirements of the output procedure.

This decomposition of AI agents (Figure 5) enables the representation of an AI service with a standard interface and supports composability.

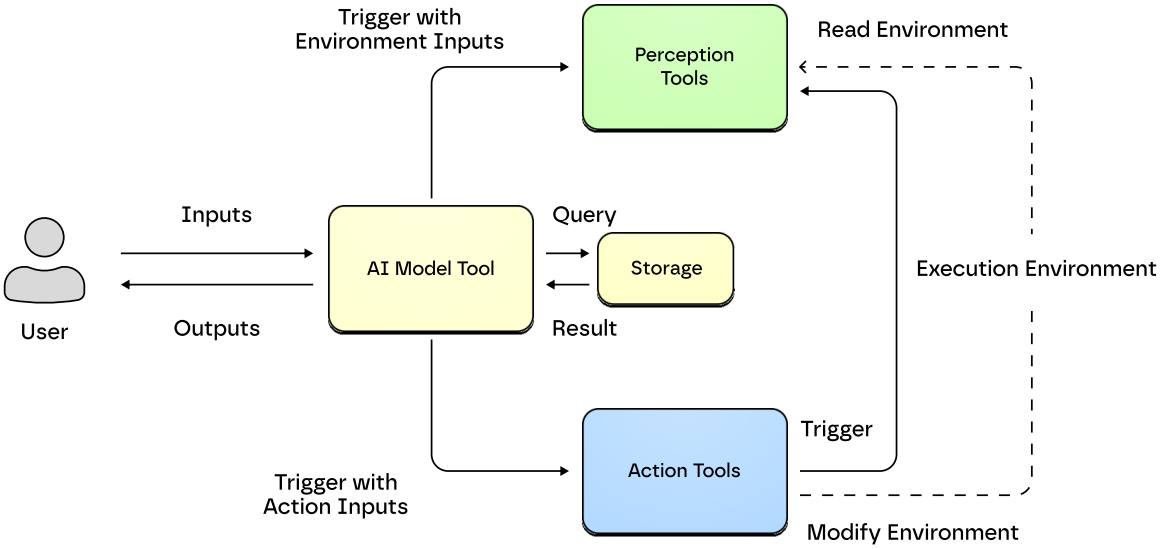

Figure 5: Decomposition of an agent service into functionalities: triggered by inputs, generate outputs, and trigger tools.

3 Talus Agentic Framework (TAF)

Talus Labs is building a platform for implementing onchain AI agent services in a composable, transparent, and censorship-resistant manner. Our approach is based on two key observations:

1. The onchain environment only recognizes onchain callable services. Therefore, we aim to encapsulate offchain AI services in a format that can be processed by the onchain virtual machine. As a result, other parties—whether offchain entities or onchain users—can interact with the AI service as they would with any onchain service, supported by a transparent service-level agreement (SLA).

2. Onchain coordination determines interoperability and security. All services, whether computed onchain or offchain, define their security guarantees through smart contracts, which in turn enable protocol-based interoperability. Thus, a protocol is required to define the security and liability of each service.

Based on these two design principles, we define the Talus Agentic Framework and build Nexus as the infrastructure for developers to create and use Talus Agents. In this section, we define the Talus Agentic Framework to describe our system. We begin with the minimal onchain service — the Talus Tool.

3.1 Talus Tool

A **Talus Tool** is an onchain service that can be triggered to generate an output given corresponding inputs. A Talus Tool executes its functionality, modifies onchain and offchain state if necessary, and eventually produces an output. Depending on the security source, a Talus Tool can be either onchain or offchain, as shown in Figure 6.

Onchain Talus Tool: The functionality of an onchain Talus Tool is completely hosted in a smart contract function with well-defined inputs. A tool can be a swap function of an Automated Market Maker (AMM) smart contract among some predefined token types. Triggering the tool allows the caller to swap their tokens for a given amount.

Offchain Talus Tool: An offchain Talus Tool includes an onchain identity and an offchain entity that holds part of the functionality. This entity can be an external service or any Web2 API that processes specific tasks based on given inputs. When invoking an offchain Talus Tool through its onchain identity, an onchain event signals the offchain entity to process the request and return the result via its onchain identity.

Talus Tools are defined based on whether the security model segregates the source of trust. One example is a zero-knowledge proof tool that ensures a verifiable computation procedure of given inputs. We need two Talus Tools to accomplish verification: an offchain tool that provides computation and proof, and an onchain tool that performs onchain verification. The user can freely decide whether or when to leverage the onchain verification based on their use cases and freely compose the tools.

A Talus Tool must stake assets to be held economically accountable for honest task execution. Any type of service can be represented as a Talus Tool, from small helper functions to an LLM-based reasoning engine or even a full AI agent. A service provider can host its AI agent service as a tool or provide an LLM proxy for inference.

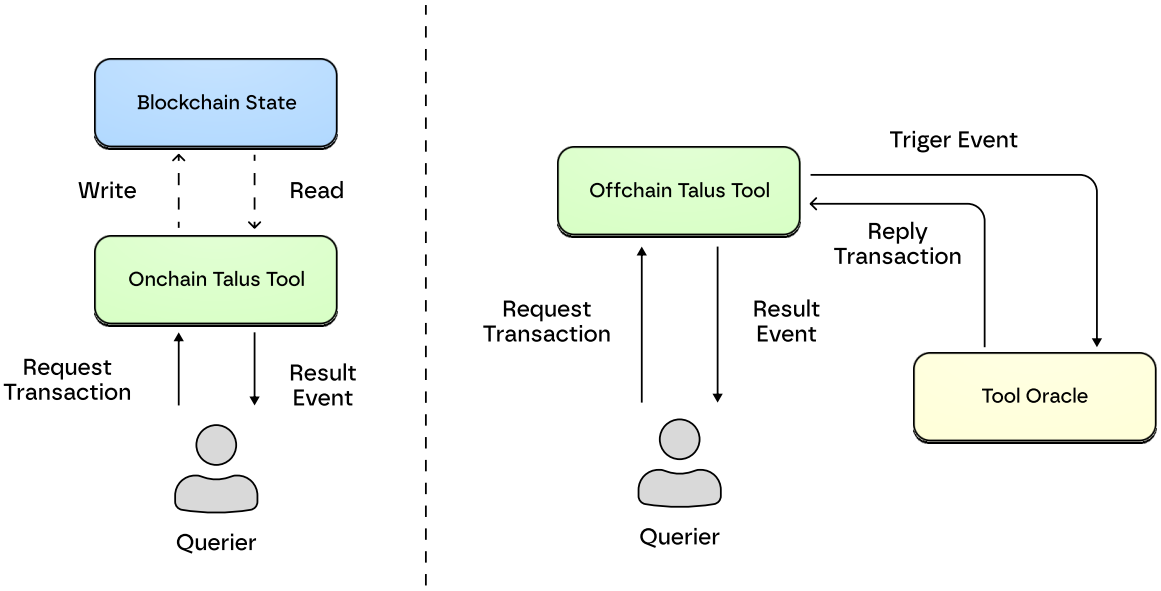

Figure 6: Talus onchain and offchain tools. Green for first-party onchain components, blue for third-party onchain components, and yellow for offchain components.

3.2 Talus Workflow

The second concept is the **Talus Workflow**, which represents the execution flow of a compound onchain service. Given the limitations of onchain resources, we use a finite Directed Acyclic Graph

(DAG) $D = (V, E)$ to represent the workflow. Here, V is the vertex set and E is the edge set. A DAG has a unique start vertex ($s \in V$) and a set of end vertices $\{t_i\} \subset V$ to represent all possible execution paths of a compound onchain service. In this DAG, each vertex $v_i \in V$ represents a Talus Tool, and an edge $e = (v_i, v_j) \in E$ denotes that the output from the source vertex v_i is consumed as the input of the destination vertex v_j. We say a vertex v is activated in two cases:

1. $v = s$ is the start vertex, and a DAG execution provides input to the vertex;
2. An activated vertex v_i outputs data to v through an edge $e = (v_i, v)$.

Given that D is a finite DAG, a DAG with at least two vertices will terminate after at most $2^{(|V|-2)}+1$ activations (since at least one terminal vertex will be triggered only once). The execution of D may terminate at some end vertex t_i with completed DAG execution or due to tool failure.

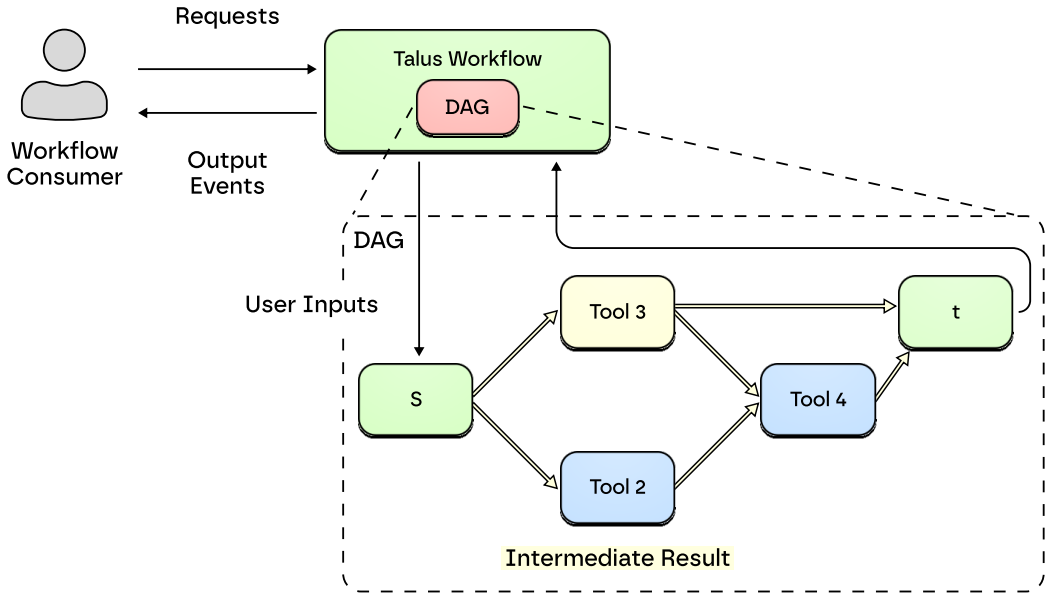

Figure 7: In the workflow execution, the user only needs to initiate a request and wait for the result.

A Talus Workflow may represent an onchain AI service or an onchain agentic workflow service (if it contains multiple onchain AI agent services). For a Talus Tool of an onchain AI agent service, incoming edges to vertex v represent the perception phase (collecting data), while outgoing edges from v represent all possible actions the agent may take (generating actions). An activated vertex may trigger a subset of its outbound edges, corresponding to different outputs. If all necessary inputs are available for a vertex, it will be activated (triggering tools for action). Thus, we can flatten an AI agent architecture as a DAG (Figure 7) of Talus Tools that can terminate in finite steps.

Due to onchain resource limitations, we explicitly avoid onchain cyclic routes by forcing the onchain execution to follow an acyclic graph. We preserve the merits of iteration in two ways: First, iteration brings more randomness into the group, which is costly for onchain verification but cheap for offchain computation. Therefore, the Talus Tool can introduce iteration in its offchain service and only submit results onchain. Second, if the iteration requires onchain actions that need to be verified (e.g., executing onchain contracts and moving assets), then the developer can integrate onchain tools into the DAG and reenter the offchain tools as another vertex in the DAG. With these

two workarounds, Talus Workflows can preserve the ability to terminate under constrained resources and benefit from iterative computation.

A Talus Workflow represents the decomposition of complex onchain services into smaller Talus Tools. This decomposition into modular components has the following implications:

- Each Talus Tool has an independent security guarantee, such that workflow design can predefine possible failures or outputs of tools and keep the tool functioning with minimal overhead from the workflow.
- Each edge in the workflow DAG represents a protocol between two onchain services for consistency between tools. Nexus supports flexible configuration of these edges, covering aspects such as data transmission and verification. For example, tools may provide optionally encrypted data transmission for secret sharing or computation proof schemes for onchain verifiability.
- Payment for a workflow can be traced at the runtime of each tool, providing a transparent view of service delivery and payment. This transparency enables workflow consumers to evaluate costs and select tools that best meet their requirements for service quality and price.

3.3 Talus Agent

A Talus Agent is an onchain identity associated with one or more onchain assets and workflow services (Figure 8). A Talus Agent can represent an AI agent if one of its workflows uses onchain AI services. The term Talus denotes using Talus Workflows and distinguishes it from other onchain AI agent services not defined within TAF.

Compared to other agent systems, the Talus Agent has two main differences:

Decentralized onchain identity: A Talus Agent is not controlled by a wallet, which allows auditable usage of the agent's assets without relying on a private key. The agent is configured through smart contracts and workflows, so all supported behaviors are stored onchain and can be invoked by users who trigger one of the workflows.

Composable service integration: A Talus Agent can be a Talus Tool within any Talus Workflow, enabling easy cooperation with any other onchain services. This feature facilitates the rapid development of multi-agent systems and the discovery of new business opportunities for existing agents. Application developers do not need to build everything from scratch—they can simply select a suitable Talus Agent from the market. The transparent payment structure also helps agent developers analyze costs and identify demand for cheaper, more secure, or more powerful Talus Tools.

One important feature of the Talus Agent is its independent management of onchain assets. A general-purpose agent should be capable of managing its assets to fully leverage workflows for transparent and auditable asset operations. Our design offers two main advantages:

1. Asset ownership can be defined onchain, rather than relying on cryptographic signatures. This provides full transparency and enables agent developers to use auditing to enhance security.
2. The Talus Workflow can exercise controllable authority over agent assets, combining the capabilities of well-defined onchain services with the security guarantees of the onchain system.

Ultimately, the Talus Agent supports an ecosystem in which developers can define diverse agents using custom workflows. Each agent can operate with distinct private assets, managed through onchain services and identified via onchain identities.

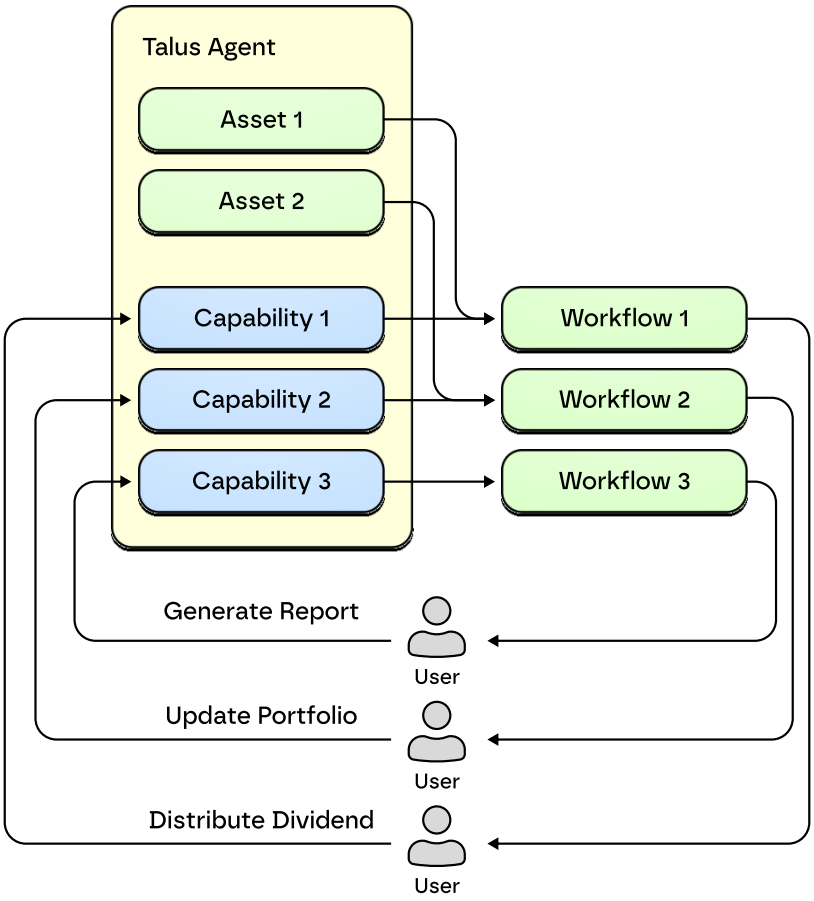

Figure 8: Using a Talus Agent as a consumer of workflows with business logic.

4 Nexus

With all components of TAF in place, we can describe the Nexus system. Nexus is a decentralized framework that enables the creation, deployment, and management of Talus Workflows that support customizable Talus Agents on the Sui Network. Talus chooses the Sui Network for three reasons:

Onchain Logic Security: The design of the Sui Move VM inherently enhances security, simplifying the development of secure protocols for managing valuable resources—a key feature of onchain logic.

Flexible Object Model: Sui Move provides an object model that effectively abstracts resources, enabling Talus to securely manage and evaluate intelligence outcomes with precision and adaptability, distinguishing it in the Web3 space.

High Performance: The architecture of the Sui Network and its VM design support efficient concurrency, allowing Talus to scale by processing multiple transactions simultaneously without compromising security or integrity.

Nexus provides the infrastructure for developing Talus Agents. It contains two main components:

1. A bundle of smart contracts that define Talus Tools, Talus Workflows, and corresponding operations to build the Talus ecosystem.

2. An offchain Leader system that serves as an oracle for offchain Talus Tools and as a messenger for workflow execution.

A user only needs to interact with the onchain components and wait for the Nexus offchain components to accomplish the task according to the onchain definition of the Talus Workflow (Figure 9).

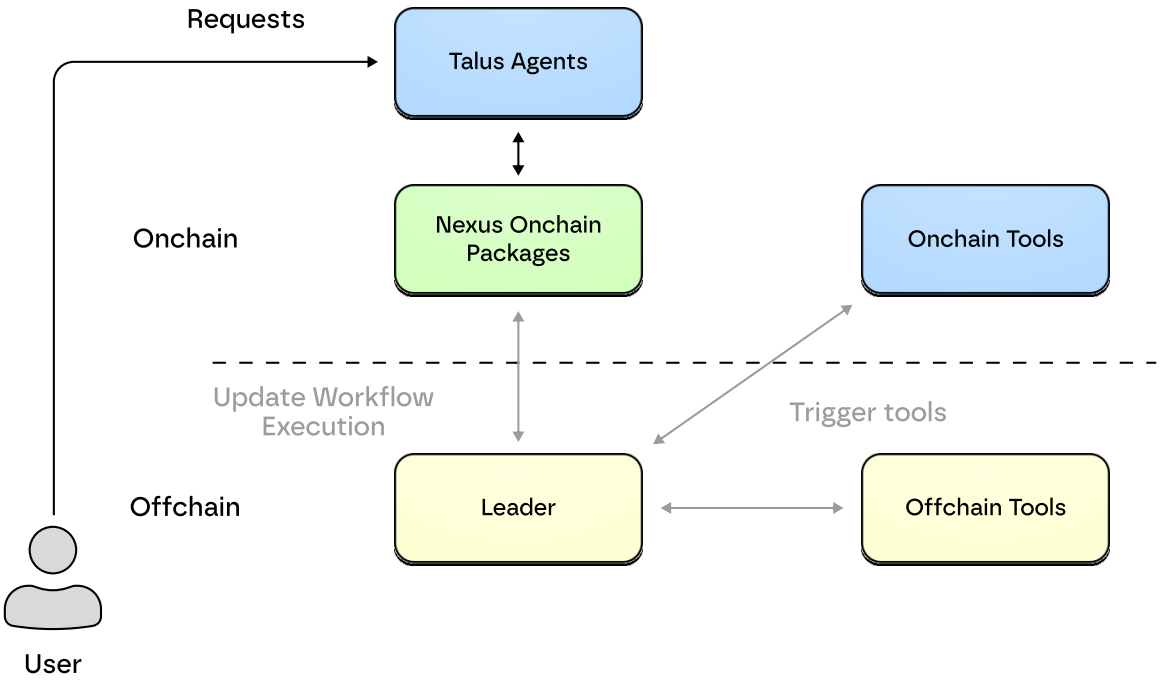

Figure 9: User flow of Nexus.

4.1 Nexus Onchain Components

The onchain components are initially deployed on the Sui Network, with smart contracts written in Sui Move. We primarily use the object system defined in the Sui Move VM to represent the onchain identities of Talus Tools, workflows, and agents. There are three types of onchain Move packages relevant to Nexus and Talus Agents (Figure 10):

Nexus Onchain Packages (NOPs): Maintained by the Talus Labs team, NOPs enable the deployment and execution of agent workflows. NOPs include definitions of core Talus concepts (Tools and Workflows), interfaces and protocols for using tools and workflows, and a registry service for managing both onchain and offchain Talus Tools.

Tool Packages: These packages consume NOPs and define the onchain tools. The Talus Labs team will provide some default tools, and more will come from third-party tool developers.

Once deployed, the tool developer can register the package's entry method as a tool within the Nexus onchain components.

Talus Agent Packages (TAPs): TAPs consume NOPs and various Talus Tools to define Talus Agents. Talus Labs will provide some default TAPs for creating agents, while third-party developers are welcome to publish their TAPs. These packages allow developers to customize Talus Agents to trigger workflows alongside other custom onchain business logic.

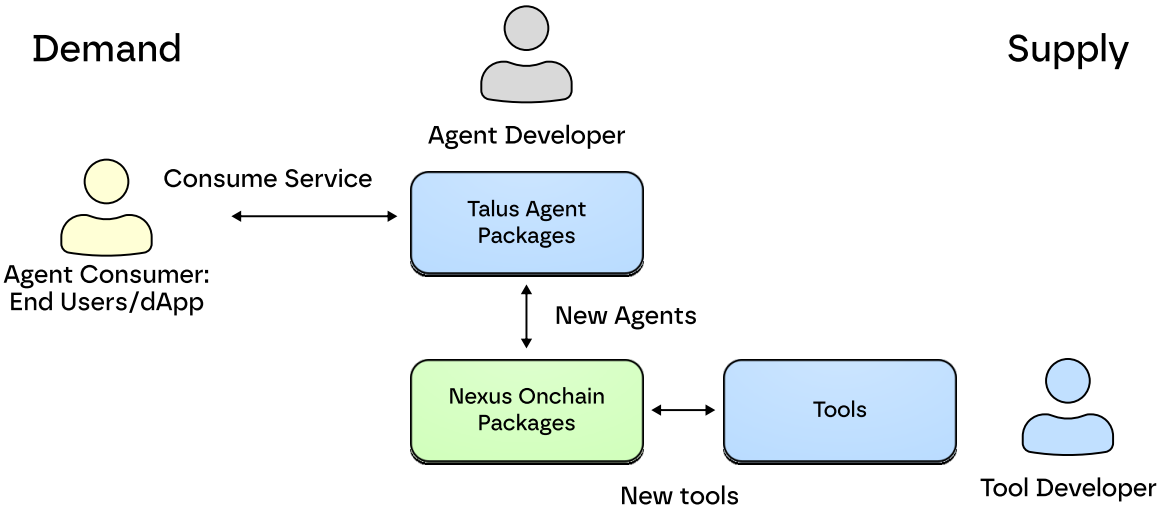

Figure 10: The relationship between different smart contract packages and the roles of developers within Nexus.

NOPs contain three packages: *primitive*, *interface*, and *workflow*. The *primitive* package defines all data structures required by TAPs and Tool packages, enabling NOPs to recognize messages from these third-party packages. The *interface* package defines the protocols through which TAPs integrate Nexus functionality. The *workflow* package contains Nexus's core logic. Tool developers use NOPs to register tools and workflows, while agent developers use the registered workflows via the interface package.

The *workflow* package defines a *DAGExecution* object for tracking the status of a Talus Workflow execution. An execution is a sequence of vertex activations representing the execution of Talus Tools. When a vertex (Talus Tool) in the DAG has all necessary inputs, it initiates a *walk* object and emits an event to the Leader system. A *walk* object can be in one of the following states: Active, Successful, Failed, or Consumed (Figure 11). An Active *walk* indicates that the Leader system is processing the execution of a vertex and its following vertices. A *DAGExecution* is successful if its current vertex reaches the Successful state. Conversely, if any vertex fails, the *DAGExecution* is considered failed. A Consumed *walk* means that the Talus Tool execution is complete, but the next tool still awaits inputs from other *walk*s. If all *walk*s are consumed, the execution also fails.

For each DAG walk, the Leader must provide data onchain to activate the next vertex. This data does not have to be the raw input required by the next tool. If the tool supports intermediate data transmission, the onchain data can be a reference to decentralized storage, along with a commitment or signature for integrity verification. This flexibility reduces gas consumption for intermediate results and supports privacy-preserving workflows via integration with permissioned storage services. These features will be supported in the *primitive* package following protocol upgrades.

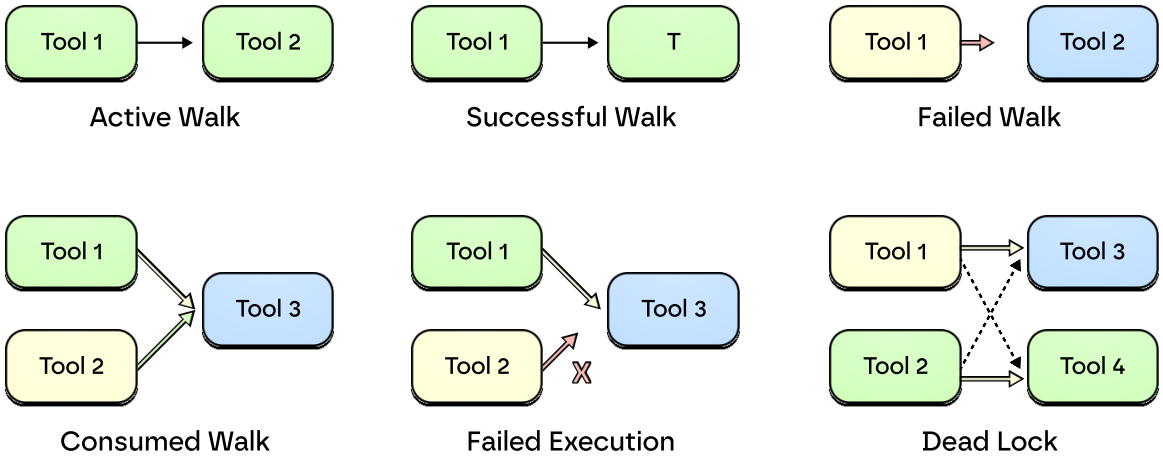

Figure 11: Green arrows represent active walk(s), orange arrows represent consumed walk(s), and red arrows represent failed walk(s). A deadlock occurs if all walks are consumed, and the workflow faces a liveness issue by design.

Each Talus Tool is associated with a *tool_gas* object that specifies the payment method for triggering the tool. Payments can be set as a fixed token amount or a valid license for a specified period. To execute a Talus Workflow, the user needs to deposit the payment as a *gas_budget* in a globally unique *gas_service* object, where Nexus will confirm sufficient gas before executing Talus Tools. For each Leader transaction, the Leader will trigger the gas sync to collect the gas fee and tool payment from the budget.

4.2 Nexus Offchain Components

Nexus supports offchain tools to enable a broader range of service providers for agent developers and users. The Nexus Leader system functions as an oracle for these offchain tools and orchestrates communication as a messenger for each edge in a Talus Workflow.

More concretely, one workflow execution for the Nexus Leader is represented in Figure 12. The Leader must:

- Listen for events emitted by Talus Workflow executions.
- Store all events in an indexer for tracking.
- Look up Talus Tool information from the cache or onchain tool registry.
- Request execution of the (onchain/offchain) tool and provide inputs according to the definition of the tool.
- Fetch the tool execution result (or error) and return the outputs to the workflow. Retry until the tool's liveness failure condition is met.

All edges in the workflow DAG can be executed autonomously through the Leader's actions without requiring user intervention. Additionally, the Leader system can offer features such as timers for triggering workflows or optimizing onchain cost by caching requests, if allowed by the protocol.

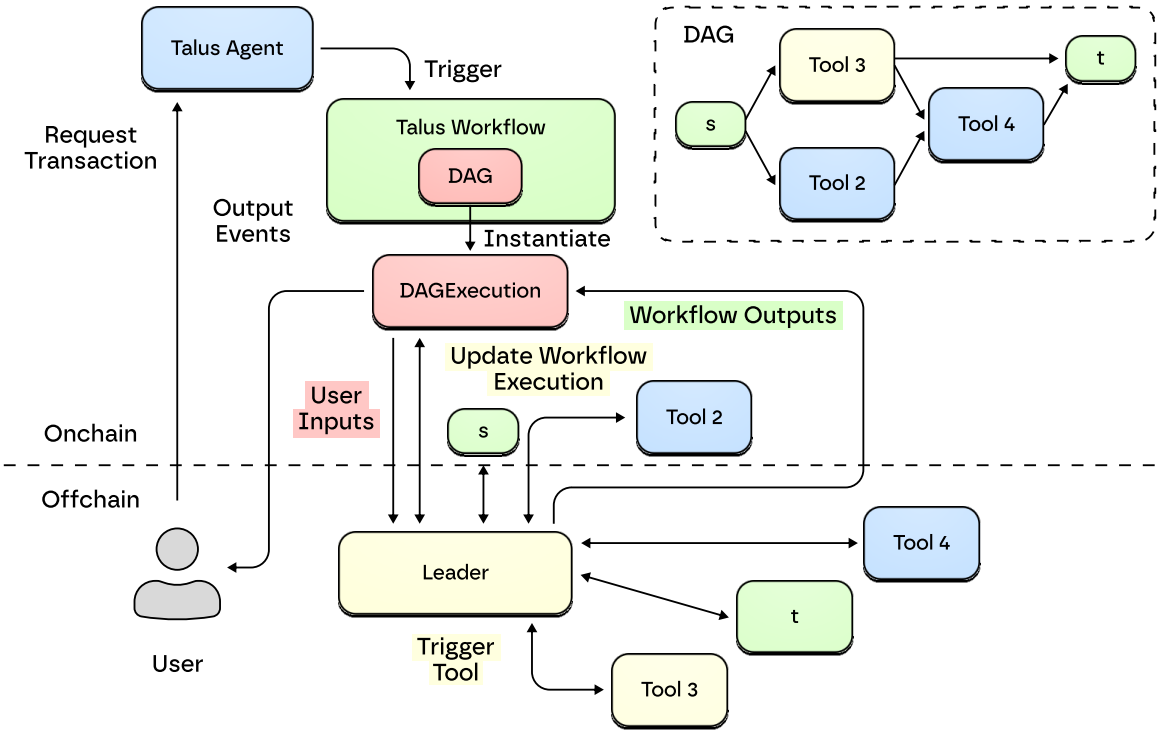

Figure 12: Complete user flow of the Nexus service.

For security, the Leader system supports onchain workflow execution with isolated permissions. It has no access to Talus Agents or user assets and merely ensures autonomous execution of Nexus workflows. As the Leader also acts as a proxy for tools, any permissions required by tools must be strictly verified.

4.3 Nexus Architecture Summary

The hybrid design of Nexus helps Talus Agent developers achieve the following:

Composability: Talus Agent developers can reuse Talus Tools and workflows defined independently onchain, and provide private inputs based on the agent's identity and permissioned assets. All tools and workflows share a unified onchain interface, and offchain messengers enable smooth composability between them.

Censorship Resistance: Talus Workflow executions are fully represented onchain, with configurable protocols for verifiability and communication. The visibility ensures all users have equal and censorship-resistant access to agent services and can customize Talus Agents with services that have transparent history records.

Autonomous Fee Collection: Payments in Nexus are handled onchain. Each Talus Tool must complete its task and pass a verification step before the next tool is triggered and paid for. Users can trace their payments and receive compensation if any step fails.

Balanced Performance and Cost: Talus supports traditional Web2 services via one or more tool providers onchain. This approach avoids the high costs of fully onchain services and leverages market dynamics to provide services at reasonable prices and quality.

Secure Integration: The offchain Leader system handles automation tasks, separating them from most onchain value flows. This reduces the workload for developers and users of complex onchain services that require coordination across multiple service providers.

4.4 Security Analysis

We now analyze the security of Nexus. At a high level, we require the workflow, as an onchain service, to ensure both safety and liveness. Workflow execution involves the NOPs packages, the Leader system, and the Talus Tool services. The NOPs and Tool packages are onchain components and inherit security from the consensus layer. Thus, we focus on the security of the offchain components: the Leader system and offchain Talus Tools.

4.4.1 Safety

A safe onchain service is defined as one where computation follows predefined rules. Therefore, the security of an offchain tool depends on the rules it is required to follow. We categorize offchain tools into three types:

Independent Offchain Tools: AI services are powerful for searching, analyzing, and executing actions for problems that traditionally require human effort. However, fully autonomous AI workflows still suffer from hallucinations and inconsistencies. As a result, the human-in-the-loop remains the mainstream approach for AI services. When onchain services involve human supervision, most of the value comes from the human operator (e.g., auditing results or approving actions). In such cases, enforcing strict onchain security for the offchain computation may be unnecessarily costly. Therefore, we provide a functionality-first option, where offchain tools are used without any safety guarantees. In this case, the agent builder must evaluate the trade-offs between cost and effectiveness and responsibly consume outputs based on their requirements.

Offchain Tools for Onchain Verifiable Results: When the agent developer seeks more autonomy in the workflow and requires the results to write to onchain state using resources, a verifiable-result design pattern can be used. In this design, any vertex (i.e., Talus Tool) that consumes inputs from another vertex must enforce strict permission restrictions and input validation. This approach retains flexibility in offchain computation but limits onchain operations to predefined, verifiable actions. For example, if an offchain tool executes a transfer for investment purposes, the agent can enforce receiver addresses and entry points through the *DAGExecution* object, thereby preventing unauthorized asset transfers.

Offchain Tools for Onchain Verifiable Computation: The highest level of security requires deterministic computation, where the offchain tool strictly follows committed onchain instructions. This design has two branches:

Verification-Based Talus Tools: The tool developer must deploy an onchain verification contract linked to the tool to ensure correctness. The offchain tool submits a proof along with the computation result. The workflow passes this proof to the verification contract, which, upon successful verification, activates the next DAG vertex. This approach requires a separate

onchain verification contract to validate the proof. However, the cost of verification may exceed that of the offchain computation due to the price of added security, especially considering the decreasing cost of AI model inference.

Trust-Based Talus Tools: The tool developer must register the public key of a trusted party onchain as part of the tool's properties (e.g., a threshold signature key or TEE device key). The tool is considered consumed only if the corresponding key signs the output. Signature verification can be implemented as a Talus Tool or integrated into Nexus's data transmission protocol.

As discussed, the safety level of each offchain tool can be customized per workflow. A workflow may choose to use a verification-based Talus Tool as an independent offchain tool to save the cost of onchain verification, or a trust-based Talus Tool for logic that is hard to verify with proofs. Regardless of the offchain tool used, result-based security can be enforced to ensure that onchain outputs are properly authorized and regulated. This flexibility enables a wide range of Talus Workflows adaptable to diverse use cases.

The safety of the Leader system relies on trust. Talus runs the Leader in a trusted execution environment. Furthermore, Leader nodes have limited permissions for onchain operations. Assuming secure communication channels between the Leader and offchain services, a workflow composed of safe tools can be executed safely. If an offchain tool is unsafe, the resulting onchain service may also be unsafe.

In summary, with properly defined onchain rules, a safe offchain Talus Tool can result in a safe onchain Talus Tool and ultimately deliver a safe onchain service. If the offchain service is unsafe, the onchain verification components can detect issues and halt the workflow, preserving the overall safety of the Talus Workflow.

4.4.2 Liveness

The liveness of the Leader system is ensured through distributed cloud deployment with geo-redundancy. A group of distributed Leader nodes serves the system, and if the main Leader node fails, backup nodes take over to keep the system alive. To account for potential message delays, all APIs triggered by the Leader are idempotent, meaning the tool will respond to the first request and ignore duplicates.

Tool liveness primarily depends on the Leader system. Assuming the Leader is live, an offchain tool will also result in a live onchain tool. If the offchain service fails to respond, the Leader marks the walk as failed and halts that part of the workflow, preserving liveness across the full execution. Therefore, the workflow remains live under the assumption of a live Leader.

Additionally, we use fallback edges in the DAG to address liveness issues. If one vertex execution fails, its fallback edge is triggered and sends the inputs of the failed vertex to a fallback vertex. This fallback mechanism allows agent developers to experiment with unstable features without compromising the liveness of the overall workflow.

4.4.3 Economic Security

We provide additional economic rationale for the safety of Talus Tools and the underlying security of Talus Workflows. Nexus adopts the SUI token for its internal pricing and charging mechanism. Our design aims to motivate Talus Tool providers to act honestly in their interest.

Trustless and Flexible Payment System: A tool provider receives guaranteed payment for each query they answer honestly, as defined through NOPs. They benefit from the trustless onchain

financial infrastructure and open market to offer their services. Nexus provides a flexible payment mechanism, allowing tool providers to adopt the business model that best suits their needs without additional platform fees.

Stake-based Tokenomics: Each tool provider must stake a fixed amount of a tokens onchain. If a tool fails due to a liveness issue, its stake will be slashed and distributed as compensation to all ongoing DAG executions that depend on the tool. If a tool charges b per query, it can support at most a/b executions. Accordingly, a workflow can support up to $a/b_{\max}$ executions, where $b_{\max}$ is the cost of the most expensive tool in the workflow.

5 Applications

Talus, through the Talus Agentic Framework and Nexus engine, is a platform that helps Talus Agent developers build onchain AI agent services. This section describes the types of applications that benefit from the Nexus design. We define two categories of use cases supported by Nexus: Agent as a Service (AaaS) and Agent Marketplace.

5.1 Agent as a Service (AaaS)

In AaaS, developers compose various onchain Talus Tools into workflows and wrap them as Talus Agents to provide automated services that enhance existing Web3 applications. AaaS represents services that focus on security and transparency, which typically use open-source and reliable service providers for robustness. With Nexus's transparent safety and liveness guarantees, Talus Agents offer advanced AI capabilities and autonomous services as supplements to other Web3 services. Below are some example applications.

5.1.1 Agent-Enhanced Oracle

Unlike traditional oracle systems, AI agent-enhanced oracles can collect a wide range of data independently and from trusted sources. The agent can use onchain oracles to access verified data sources (based on trust or source verification), such as domain-verified websites, academic databases, knowledge bases, or legitimate news APIs. This flow ensures:

1. The agent's inputs are reliable, making its outputs trustworthy.
2. An automated oracle can review and publish data without human intervention.

This enables real-time generation of market digests and actionable insights. For example, an onchain AI agent can autonomously scan trusted financial news sources, synthesize key signals, and adjust a public index or trading strategy via smart contracts—with full auditability and no human in the loop.

5.1.2 Agent-Enhanced DeFi

A major bottleneck in DeFi is that many financial products require heavy computation. For instance, calculating the best swap route becomes increasingly intensive as liquidity providers grow. Implementing these algorithms fully onchain is often infeasible due to gas limitations.

A secure and intelligent onchain service that leverages offchain computation is therefore valuable. A Talus Agent can:

- Understand complex user requests (e.g., conditional token swaps).
- Take actions such as collecting supported token pairs from onchain systems and computing the cheapest swap route.
- Generate a list of onchain commands for an aggregated wallet to transfer assets and execute the swaps.

The agent uses transparent market data to optimize routes without increasing the onchain computational burden.

5.1.3 Agent-Enhanced Onchain Gaming

Onchain gaming remains a popular domain but is constrained by limited onchain randomness and interactivity. With Nexus, games can combine onchain economies and complex game design via offchain tools to generate rich content. A Talus Agent can provide:

1. 24/7 services via reliable service providers and the Nexus system with defined transparency and security guarantees.
2. Automatic processing of user requests and dynamic game experiences based on user status, agent resources, and game constraints.
3. Shared infrastructure, enabling game developers to reduce the effort required to integrate AI services and support open-world onchain gaming.

5.2 Agent Marketplace

The second category is Agent Marketplace, where Nexus helps AI agents and offchain service providers use onchain infrastructure. Talus Agents act as independent service providers for delivering monetized offchain services.

In this model, agents and underlying offchain services directly serve end users. Compared to AaaS, Agent Marketplace often requires stricter verification of final outputs rather than just the computation procedure. Nexus supports this need through verifiable workflows, where offchain computation can be blinded, but outputs are verified before payment and delivery.

5.2.1 Investment Through Agents

Using AI agents for dynamic investing has traditionally been difficult due to centralized permission management. Nexus provides three advantages:

1. Talus Agent contracts can control investment strategies, ensuring agents operate only within a secure, authorized range.
2. Agents can form collaborative workflows to adjust portfolios based on combined results.
3. Closed-source AI models can register as Talus Tools and monetize their services as consultants without handling wallet operations directly.

5.2.2 Outsourcing Through Agents

Since agents can hold independent assets, they can compete in tasks such as gaming or complex computational problems. Agent developers can fine-tune agents based on performance in these competitions.

One example is scientific outsourcing, such as protein structure prediction:

- Service providers can publish models as onchain tools and organize them through a large workflow to compete for real-world tasks.
- Task providers publish one-time heavy tasks onchain, incentivizing agents to solve them transparently and with economic rewards.

The onchain payment system reduces the overhead for task providers by automating payments to different AI agent services.

5.2.3 Proof of Concept (PoC) Through Agents

As AI models improve, bringing products to market becomes increasingly challenging due to competition and market fragmentation. Standalone AI services may struggle to attract users or monetize effectively. Nexus enables a separation between product logic and underlying AI technology. Talus Tool providers can offer services with finer granularity, such as daily usage or query-based pricing, significantly reducing the cost of trials and integration. Talus Agent builders can focus on improving services as onchain tools and collect payments accordingly. Product designers can explore these tools to build Talus Agents for rapid proof of concept (PoC).

For example, a designer could combine a text-to-image model with an NFT issuing tool to mint personalized NFTs. The workflow could then extend to a 3D modeling tool and place an order via an e-commerce integration tool for 3D printing. Finally, a logistics service-based tool could ensure that payment is only released once the item is delivered to the registered address.

6 Future Directions

The Talus Agentic Framework (TAF) defines how to integrate AI agents or broader offchain services onchain. The safety of these services relies on output verifiability, which is ultimately secured by the consensus algorithm. We will publish a series of papers introducing innovative Talus Agents within TAF.

For Nexus, one important direction is integrating more protocols—such as zero-knowledge proofs, authenticated communication, and third-party delivery proofs—to enhance the verifiability and security of Talus Tools and workflows. We aim to offer more choices to attract offchain and AI service providers to enter the Web3 ecosystem and collaborate with application builders.

Another direction is improving the efficiency of Nexus and workflow execution. The core function of the Nexus Leader is secure message delivery and workflow coordination. By incorporating proof schemes and onchain contract design, we aim to minimize trust in the Leader system and reduce gas consumption.

References

[AS87] Bowen Alpern and Fred B Schneider. Recognizing safety and liveness. *Distributed computing*, 2(3):117–126, 1987.

[BCC+21] Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, et al. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. *Chainlink Labs*, 1:1–136, 2021.

[BDKJ23] Kushal Babel, Philip Daian, Mahimna Kelkar, and Ari Juels. Clockwork finance: Automated analysis of economic security in smart contracts. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2499–2516. IEEE, 2023.

[BMZ24] Massimo Bartoletti, Riccardo Marchesin, and Roberto Zunino. Defi composability as mev non-interference. In *International Conference on Financial Cryptography and Data Security*, pages 369–387. Springer, 2024.

[CD16] Victor Costan and Srinivas Devadas. Intel sgx explained. *Cryptology ePrint Archive*, 2016.

[CHH97] Ran Canetti, Shai Halevi, and Amir Herzberg. Maintaining authenticated communication in the presence of break-ins. In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing (PODC 97)*, pages 15–24, 1997.

[CSYW24] KD Conway, Cathie So, Xiaohang Yu, and Kartin Wong. opml: Optimistic machine learning on blockchain, 2024.

[GMR85] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing (STOC 85)*, pages 291–304, New York, NY, USA, 1985.

[MRV99] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (FOCS 99)*, pages 120–130, 1999.

[PHdB22] Fernando Gomes Papi, Jomi Fred Hübner, and Maiquel de Brito. A blockchain integration to support transactions of assets in multi-agent systems. *Engineering Applications of Artificial Intelligence*, 107:104534, 2022.

[POC+23] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.

[SEKK24] Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. Enhancing ai systems with agentic workflows patterns in large language model. In *2024 IEEE World AI IoT Congress (AIIoT 2024)*, pages 527–532, 2024.

[Sho00] Victor Shoup. Practical threshold signatures. In *Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19*, pages 207–220. Springer, 2000.

[Teaa] DecideAI Team. Decideai whitepaper. Online; accessed Apr. 2025.

[Teab] Dfinity Team. The internet computer for geeks. Online; accessed Apr. 2025.

[Teac] Ritual Team. Introducing ritual. Online; accessed Apr. 2025.

[Tead] TensorflowSui Team. Tensorflowsui code repository. Online; accessed Apr. 2025.

[WGN+25] Shaw Walters, Sam Gao, Shakker Nerd, Feng Da, Warren Williams, Ting-Chien Meng, Amie Chow, Hunter Han, Frank He, Allen Zhang, Ming Wu, Timothy Shen, Maxwell Hu, and Jerry Yan. Eliza: A web3 friendly ai agent operating system, 2025.

[YZY+23] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[ZCC+16] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 aCM sIGSAC conference on computer and communications security*, pages 270–282, 2016.

[zT] zkML Team. zkml whitepaper. Online; accessed Apr. 2025.

[ZXD+17] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE international congress on big data (BigData congress)*, pages 557–564. Ieee, 2017.