Gear.exe - Vision of Decentralized Bridgeless Computing Extension to Ethereum Network

Nikolay Volf, Andrei Panin, Eugene Way

Version 0.2 | Sep 2025

Overview of Gear.exe

Gear.exe is a groundbreaking decentralized compute network designed to significantly enhance the computational capabilities of Layer-1 blockchain networks, starting with Ethereum. Unlike conventional Layer-2 solutions, Gear.exe introduces a transformative approach to decentralized application (dApp) architecture. It provides a real-time, high-performance, parallel execution environment with near-zero gas fees and a pre-confirmation mechanism for instant user feedback — while canonical finality remains unchanged and fully aligned with Ethereum's native finality. Gear.exe integrates with the familiar Ethereum toolchain — such as MetaMask, Etherscan, Hardhat, and Tenderly — enabled by technologies like Sails that expose typed interfaces and simplify integration.

The mission of Gear.exe is to enable developers to build scalable, efficient, feature-rich, and user-friendly dApps. As a real-time co-processor working alongside Ethereum, Gear.exe enhances computational power without causing liquidity fragmentation or requiring asset bridges. By maintaining Ethereum's robust security and liquidity, it empowers developers to create applications with Web2-level user experiences while utilizing the unique advantages of blockchain technology.

Each program in Gear.exe maintains its own isolated state, message queue, and execution context — effectively acting as an independent compute unit. Programs are executed on-demand across a decentralized network of Executor nodes, with no shared memory or global storage between them. As a result, each program can be naturally viewed as its own "individual rollup." Collectively, these programs form a highly parallelizable compute layer — a "swarm of rollups" — enabling Gear.exe to achieve massive horizontal scalability. This architecture allows multiple programs to execute simultaneously without data contention or coordination overhead, providing developers with unprecedented flexibility to implement resource-intensive logic at scale without sacrificing performance or composability.

For developers, the process begins with identifying the most computationally intensive parts of their application's business logic. These components are extracted from Solidity-based smart contracts and reimplemented as WebAssembly (WASM) programs executed off-chain by Gear.exe. These programs can then be invoked as needed, radically reducing operational complexity and gas-related costs.

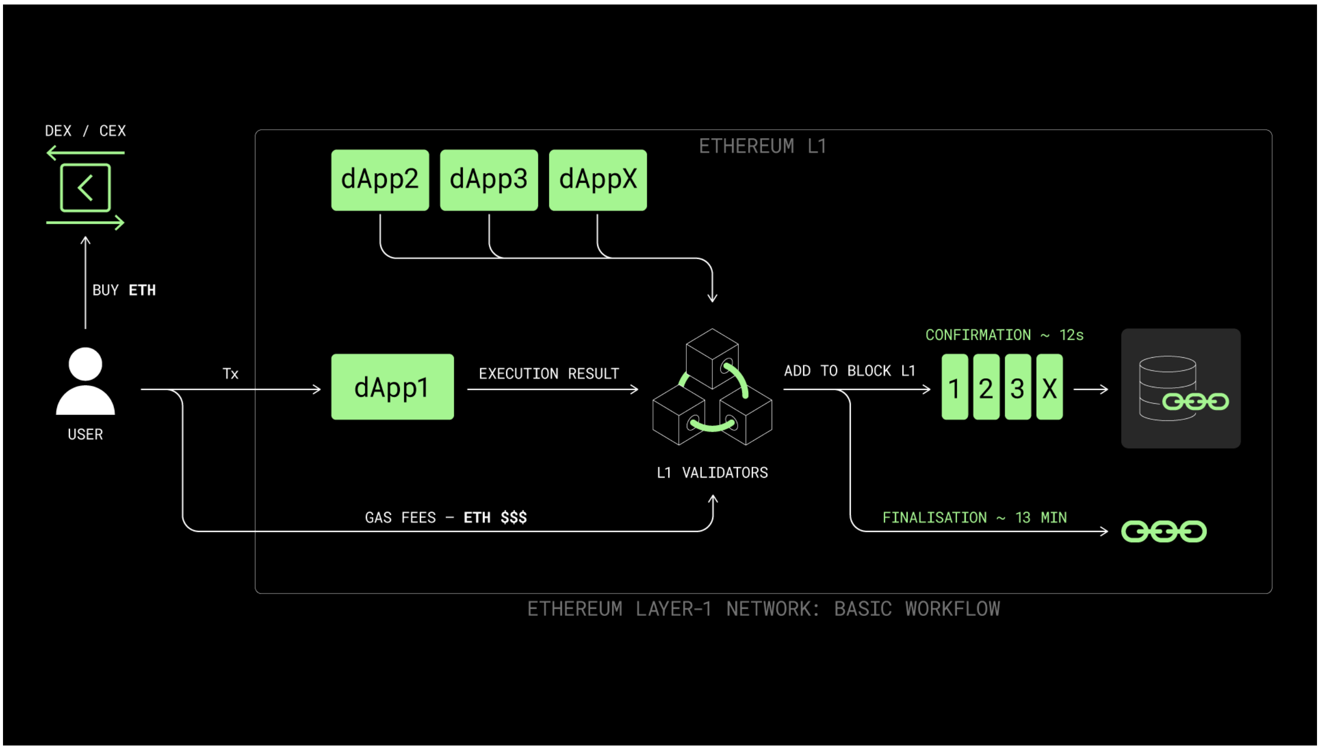
This approach is particularly attractive for existing applications constrained by Ethereum's resource limits, as well as for projects that were previously impractical to launch due to high execution costs or performance bottlenecks that degrade the user experience.

Gear.exe: Revolutionizing Ethereum

Why Ethereum Needs Gear.exe

Ethereum remains the dominant blockchain for decentralized application development, but it faces significant challenges that hinder its scalability and usability. The network's inability to process transactions in parallel, its slow finality times, and its high gas fees are critical barriers for developers and end users alike. These limitations are especially pronounced in high-demand sectors such as DeFi, gaming, and enterprise applications, where responsiveness and affordability are crucial to user adoption.

The root of these challenges lies in Ethereum's single-threaded architecture. The lack of parallel processing limits the network's computational throughput, making it difficult to handle complex or resource-intensive operations. Block confirmation time about 12 seconds introduces sensitive delays to user interactions. Finality times, averaging around 13 minutes, intensify the problem. While transactions in a block are usable after one block confirmation, applications requiring high security typically wait for finalization to ensure immutability. High gas fees (i.e., expensive computations) further deter adoption, particularly for applications that require frequent or intensive computations.



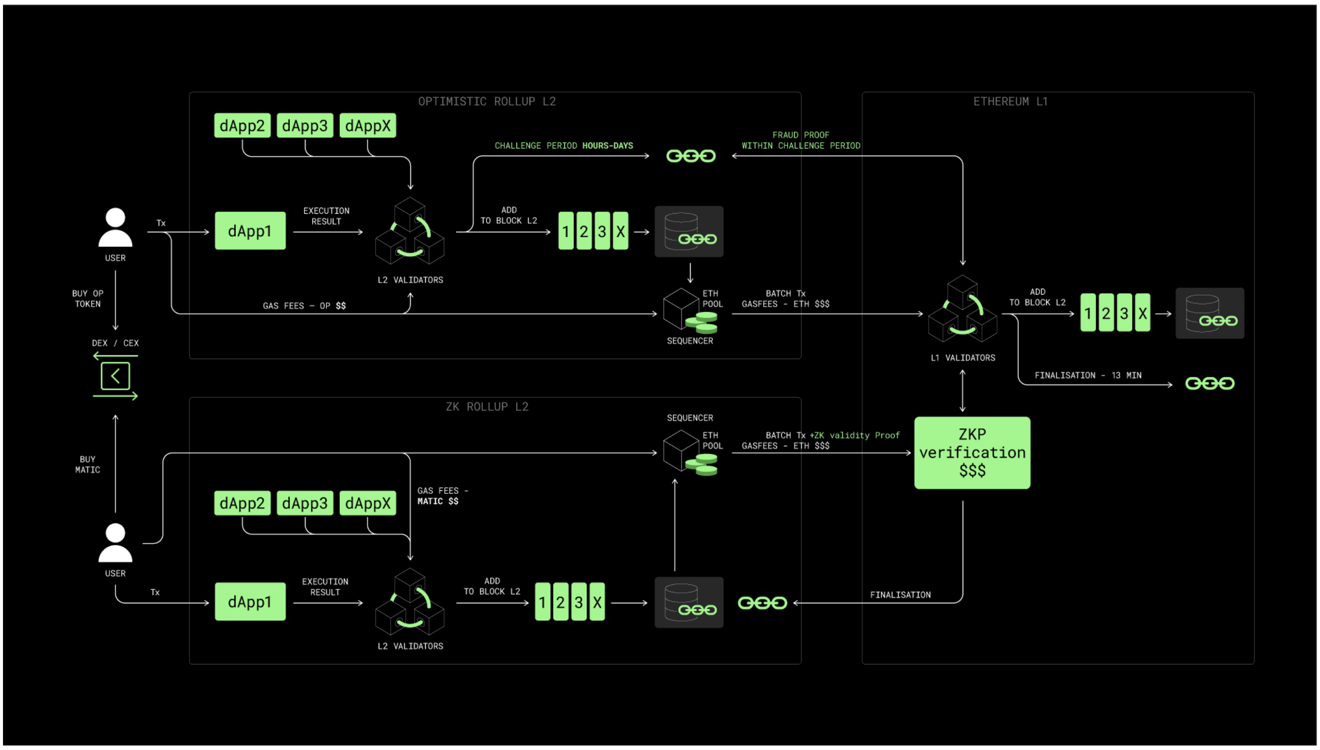
Ethereum Layer-1 Network: Basic Workflow

Layer-2 solutions such as Optimistic Rollups, ZK Rollups, and Based Rollups have attempted to address these issues by offloading transaction processing from the Ethereum main chain. However, while they improve scalability, they introduce trade-offs that limit their effectiveness in certain scenarios.

Optimistic Rollups rely on a lengthy challenge period for security, delaying transaction finalization from a few hours to several days, depending on the specific rollup implementation.

ZK Rollups, while faster, impose significant computational overhead due to the resource-intensive nature of proof generation that includes a combination of complex cryptographic operations, large circuit sizes, and the need for rigorous guarantees of correctness and privacy.

Both approaches often operate in isolated environments, fragmenting liquidity and complicating interoperability.



Interaction of L2 Networks with Ethereum L1: Basic Workflow

In 2023, Based rollups were proposed as an alternative, leveraging Ethereum's Layer-1 protocols for sequencing and decentralization. While these rollups reduce reliance on token-based mechanisms and simplify certain operations, they inherit scalability limitations due to shared transaction sequencing and data availability constraints. They sacrifice transaction flexibility needed for custom transaction sequencing, which can hinder their effectiveness for certain specialized use cases.

So What?

Gear.exe presents a fundamentally different approach by functioning as a decentralized compute network **fully integrated with Ethereum**. Unlike rollups, where smart contracts are deployed separately on Layer-2 chains, Gear.exe keeps all operations native to Ethereum. This design allows programs running on Gear.exe to interact seamlessly with Ethereum's existing smart contracts, eliminating the need for asset bridging and avoiding liquidity fragmentation. Developers can utilize Ethereum's robust ecosystem without the additional complexity introduced by traditional Layer-2 solutions.

Another critical advantage of Gear.exe lies in its **memory capacity**. With up to 2GB of memory allocated per program, Gear.exe enables developers to execute resource-heavy computations that are impractical on Ethereum or Layer-2 rollups. For comparison, Ethereum and Optimistic Rollups are constrained by gas limits, which indirectly restrict memory usage to a fraction of what Gear.exe provides. Similarly, ZK Rollups, while efficient in compressing data for on-chain validation, impose strict limitations on memory to prioritize proof generation efficiency.

Gear.exe's expanded memory allocation opens the door for advanced use cases such as Monte Carlo simulations, AI model training, and real-time data analysis.

The **multi-threaded execution** engine further sets Gear.exe apart. Ethereum and most rollups process transactions sequentially, limiting throughput and creating bottlenecks in high-demand scenarios. In contrast, Gear.exe supports parallel execution, allowing multiple computations to run simultaneously. This innovation is particularly beneficial for latency-sensitive applications, including high-frequency trading platforms, gaming environments, supply chain monitoring systems and more.

At the core of this design is Gear.exe's naturally parallel execution model: each program has its own isolated state and queue, enabling thousands of programs to be executed concurrently without coordination bottlenecks.

This parallelism extends beyond the execution layer into the very architecture of Gear.exe itself, offering powerful horizontal scalability options. While current deployments may use a unified validator set and single Router, the system is inherently designed to support multiple scaling paradigms in the future:

- Single Router with unified validator set: All validators serve all programs under one Router instance — simple and consistent.
- Multiple independent Routers (clusters): Each Router has its own validator set and serves a distinct pool of programs, enabling Gear clusters to scale horizontally — all anchored to Ethereum L1.
- Subgrouped validators within a single Router: Validators are partitioned to handle subsets of programs, reducing synchronization overhead and improving scalability within a single cluster.

These potential architectures are not mutually exclusive — they can be combined to form a modular and adaptive execution fabric, where computation scales both within programs (via threads) and across program clusters. The result is a compute network that grows with demand while preserving Ethereum-level security and canonical finality.

Cost efficiency is another defining feature of Gear.exe. By offloading intensive computations to its decentralized network, Gear.exe reduces the costs associated with executing complex logic. Additionally, it introduces a reverse gas model, where developers can cover transaction fees for users. This approach provides a frictionless experience similar to Web2 applications, enabling developers to design user-friendly dApps that prioritize accessibility and adoption.

Gear.exe also enhances user and developer experience by allowing off-chain transactions with pre-confirmations. Unlike Layer-2 solutions that often delay finality due to challenge periods or proof generation, Gear.exe delivers **immediate computation results** before they are finalized on Ethereum. This capability ensures real-time responsiveness while maintaining the security guarantees of blockchain-based systems.

The use of **Rust** as the primary programming language for Gear.exe programs further differentiates it from traditional Ethereum development. Rust is a widely used, general-purpose language known for its performance and safety, offering a robust ecosystem that is accessible to a broad range of developers. In contrast, Solidity, Ethereum's native language, is blockchain-specific and requires a steeper learning curve. By leveraging Rust, Gear.exe simplifies the development process while enabling the creation of more powerful and maintainable applications.

In summary, Gear.exe addresses Ethereum's limitations and surpasses the capabilities of Layer-2 solutions by offering seamless Ethereum integration, expanded memory capacity, parallel execution, cost-efficient processing, and developer-friendly tools. By bridging the gap between Ethereum's security and the performance demands of modern applications, Gear.exe is paving the way for the next generation of decentralized applications across industries such as finance, gaming, AI tools, math modeling, supply chain management and many more.

Finally, the architectural foundation of Gear.exe is inherently chain-agnostic. While the current implementation is deeply integrated with Ethereum, the same execution model can be ported to other Layer-1 protocols. In this way, Gear.exe becomes not only a co-processor for Ethereum, but a general-purpose execution layer capable of extending scalable execution power to any chain that supports smart contract anchoring and event propagation.

Key Features and Advantages

Gear.exe offers a suite of features that address the scalability and usability challenges faced by existing blockchain solutions. These features are designed to empower developers and enhance the user experience, making Gear.exe a powerful tool for modern dApp development.

Seamless Integration with Ethereum

Gear.exe network is fully integrated with Ethereum and operates directly with native Ethereum's smart contracts. This compatibility ensures that developers can adopt Gear.exe without needing additional tokens, interfaces, or complex configurations. Users and developers can keep using Ethereum's existing tools and infrastructure they are familiar with for developing and interacting with Solidity-based smart contracts, including MetaMask, Etherscan, popular developer frameworks, environments, debugging tools, IDEs (Thirdweb, Tenderly, The Graph etc). Interaction with Gear.exe programs does not require custom ABI encoding/decoding — thanks to Sails, developers can call programs directly using familiar Ethereum tools.

Parallel Execution

Gear.exe's architecture inherently supports parallel execution of programs, leveraging multiple CPUs to handle computational workloads efficiently. This capability allows developers to distribute tasks across several threads, enabling faster processing for applications like AI models, financial simulations, and complex gaming logic. By optimizing workloads for parallel execution, Gear.exe significantly boosts throughput and reduces bottlenecks, ensuring that even the most demanding applications can operate seamlessly.

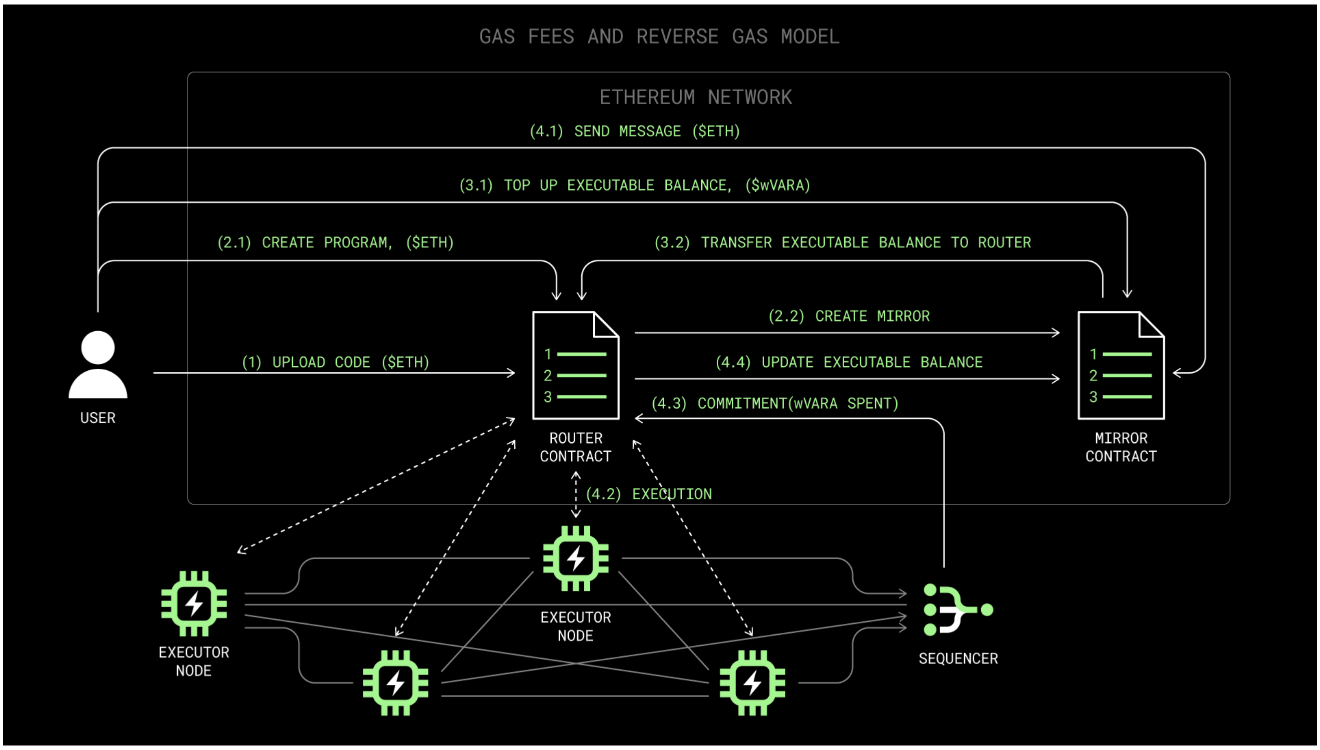
Programs are inherently isolated and can run in parallel across executors. Additionally, developers can design their logic to further distribute workloads across multiple programs, enabling natural horizontal scaling.

Advanced Programming Environment

Gear.exe provides developers with a cutting-edge programming environment by combining the power of WebAssembly (Wasm) with the flexibility of Rust, a widely adopted and developer-friendly language. Wasm programs on Gear.exe enable high-performance, lightweight execution, while Rust's rich ecosystem and safety features make it easier to write, test, and maintain complex applications. Additionally, Gear.exe supports up to 2GB of memory per program (current limit), significantly exceeding Ethereum's gas-constrained execution where effective memory rarely exceeds a few MB. This combination empowers developers to create larger, more sophisticated applications, such as financial simulations, AI models, and real-time gaming systems, without being hindered by traditional blockchain limitations.

Reverse Gas Model and Flexible Gas Management

Besides the fact that Gear.exe minimizes the costs associated with decentralized computation by offloading resource-intensive tasks to its network, it also introduces a Reverse Gas model, shifting the cost of execution from users to the program itself. This approach ensures a seamless and accessible user experience, enabling broader adoption of decentralized applications (dApps).



Gas Fees and Reverse Gas Model

In Gear.exe, programs maintain two types of balances:

- Executable Balance: Dedicated solely to program execution. If this balance is depleted, the program cannot process new messages until replenished.
- Free Balance: Acts as a general-purpose wallet for funds earned by the program, which can be withdrawn or converted into Executable Balance if supported by the program logic.

This model allows anyone to send messages without incurring extra compute costs beyond the base Ethereum transaction fee. The Executable Balance is consumed during execution, while funds are distributed to the network's Executors as rewards. Developers can design applications that fund their Executable Balance through revenue models like user payments, fees, or even sponsorships.

The reverse gas model enhances accessibility and usability, eliminating user-side complexity while promoting scalability and efficiency for dApp creators. This makes Gear.exe particularly suited for applications that prioritize user adoption and real-time responsiveness, such as financial services, gaming platforms, and enterprise solutions.

Real-Time Computation Result and Pre-confirmations

For latency-sensitive applications, Gear.exe introduces its own technical implementation of a well-known pre-confirmation mechanism. This feature allows developers to access computation results immediately after execution, even before the transaction is finalized on-chain. By bridging the gap between decentralized security and Web2-like responsiveness, this capability enables the development of cutting-edge applications in finance, competitive gaming, and other industries. Pre-confirmations provide early results for UX, while canonical finality still follows Ethereum's finality rules.

No Own Blocks

Unlike traditional Layer 2 solutions such as Arbitrum and Optimism, which generate and store their own blocks, Gear.exe does not create blocks. Instead, it processes transactions and program state changes directly within its network, leveraging its decentralized compute architecture. Rather than producing blocks, Gear.exe checkpoints batched program state changes to Ethereum, ensuring security while avoiding L2-style block overhead. By avoiding block creation, Gear.exe eliminates the overhead associated with block production and consensus mechanisms, reduces latency, and enables real-time computation. This design enhances scalability and allows for more efficient resource utilization, making it ideal for applications requiring instant feedback and high computational throughput.

Core Components

Gear.exe redefines decentralized computation by operating as a P2P compute network rather than a standalone blockchain. It eliminates the need to produce its own blocks or maintain a shared state, focusing solely on efficient and reliable off-chain computation. Gear.exe relies on several key components that enable its interaction with the Ethereum ecosystem and execution of WASM-based programs. These components work together to provide a seamless, scalable, and efficient computational layer.

Gear Programs

Gear.exe programs are developed as WASM modules using the Gear Protocol framework, similar to Vara programs. These programs enable developers to implement arbitrary logic tailored to their applications.

Programs are uploaded to Ethereum as "blobs" — a form of data stored outside Ethereum's main state but accessible through archive nodes. This mechanism ensures that large datasets can be efficiently stored without burdening the Ethereum network's main state. Each Gear program can allocate up to 2GB memory, allowing for the execution of highly complex computations, a capacity that far exceeds the stricter memory constraints of Ethereum, Optimistic Rollups, Based Rollups, and ZK Rollups. Once uploaded and verified, the program becomes available for execution within the Gear.exe network.

This one-time upload and registration mechanism ensures the security and integrity of all Gear programs while simplifying the workflow for developers, enabling seamless program reuse across multiple dApp interactions.

Router Contract

The Router Contract, written in Solidity, serves as the primary interface between Ethereum and Gear.exe. This contract plays a pivotal role in bridging off-chain computations with Ethereum's on-chain infrastructure.

Key functions of the Router Contract include:

- **Program Management**: Developers can upload and manage WASM programs for execution within Gear.exe.
- **Result Handling**: The Router Contract finalizes validator-signed execution batches and applies state updates for associated Mirror Contracts.
- **Validator Coordination**: The contract maintains validator keys and activates validator sets sourced from Symbiotic.

The Router Contract is a central component, deployed once for the entire Gear.exe ecosystem, ensuring a single coordination point within Ethereum.

Mirror Contract

For every uploaded Gear program, a corresponding Mirror Contract is automatically deployed on Ethereum. This contract acts as the primary interface between the on-chain and off-chain environments, enabling smooth interaction between Gear.exe and Ethereum-based components. The deployment of Mirror Contracts for each Gear program ensures modularity and scalability.

Mirror Contracts handle three primary tasks:

- **Initiating Requests**: They emit events that trigger the execution of WASM programs within the Gear.exe network.
- **Receiving Results**: Mirror Contracts receive execution results from the Router Contract and relay them to other Ethereum-based smart contracts or dApps.
- **Typed Interfaces**: Mirror Contracts now expose typed ABI interfaces (via Sails), allowing program inputs and outputs to be interpreted natively. This design fully replaces the older Decoder Contract model, making Mirrors the default and universal interface layer.

Executors

Executors are the backbone of the Gear.exe network, functioning as decentralized nodes that execute Wasm programs. These nodes ensure the seamless operation of Gear.exe by maintaining redundancy, decentralization, and real-time computational capabilities. Unlike traditional blockchain nodes, Executors operate without a shared storage root, focusing entirely on program execution and result validation.

The responsibilities of Executors include:

- **Event Detection**: Executors monitor events emitted by Router and Mirror Contracts on Ethereum. These events signal the need to retrieve and execute specific Wasm programs stored in the Gear.exe network.
- **Program Execution**: Upon detecting a valid event, Executors fetch the corresponding program, execute its logic, and produce results. These computations leverage Gear Protocol's Wasm runtime, ensuring high performance and flexibility.
- **Result Signing**: Executors sign computation results, which are then aggregated and finalized through the Router.
- **Peer Coordination**: Executors communicate through a peer-to-peer (P2P) network, ensuring fault tolerance and redundancy across the Gear.exe ecosystem.

Executors are selected through Symbiotic Protocol's restaking mechanism, which aligns economic incentives with performance and reliability. Misbehavior, such as producing inaccurate results, is deterred by a robust slashing mechanism that reduces the offending Executor's stake. This economic accountability ensures that the network remains secure and trustworthy. Executors can also provide **pre-confirmations** — off-chain attestations that give users immediate feedback before final settlement on Ethereum.

Middleware

Middleware connects Gear.exe to the Symbiotic restaking protocol, ensuring that technical execution is backed by economic security. It manages operator registration, validator elections, reward routing, and slashing enforcement. By aligning operators and stakers with transparent incentives, Middleware guarantees both accountability and resilience of the network.

In practice, Middleware coordinates the flow of value and responsibility:

- **Operator Lifecycle**: registering operators (executor nodes), tracking their performance, and enabling or disabling them within the network.
- **Validator Elections**: selecting and rotating validator sets based on restaked collateral, ensuring fairness and continuity of security.

- **Rewards Distribution**: routing rewards both to node operators for their work and to stakers who provide collateral.
- **Slashing and Discipline**: enforcing penalties for misbehavior or downtime, preserving the integrity of execution.

By combining these roles, Middleware acts as the economic backbone of Gear.exe — making sure that every off-chain computation is not only technically correct but also economically secured and accountable.

Integration of Ethereum dApps with Gear.exe

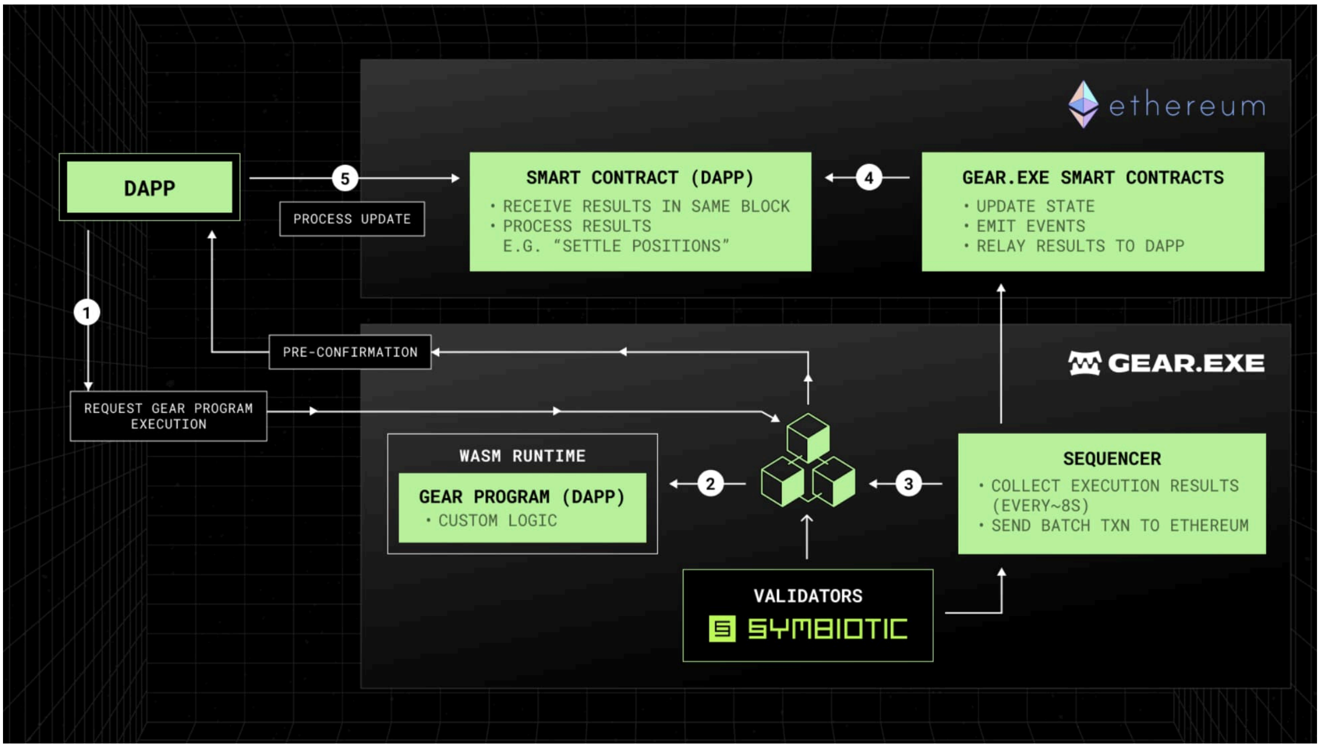
Methods

Gear.exe offers two distinct methods for integrating Ethereum dApps, allowing developers to choose the approach that best suits their application's requirements.

The first method, **Event-Based Integration**, relies on Ethereum smart contracts emitting events to request off-chain computations. These events are detected by Executors within the Gear.exe network, triggering the execution of the specified Wasm program. Once the computation is complete, the results are sent back to Ethereum through the Mirror Contract. This approach ensures a decentralized interaction between Ethereum and Gear.exe, maintaining the security and integrity of the process.

The second method, **Native Integration**, allows dApps to directly interact with their Gear programs via Remote Procedure Call (RPC). Unlike the event-based approach, native integration bypasses the need for Ethereum events, enabling real-time interactions with the Gear.exe network. This method is particularly advantageous for applications that require immediate results, as it leverages Gear.exe's pre-confirmation mechanism to provide outputs instantly.

Both integration methods are designed to be developer-friendly and scalable, ensuring that dApps can seamlessly incorporate Gear.exe's computational power without compromising security or performance.



This diagram illustrates native integration of an Ethereum-based dApp with Gear.exe

Brief Workflow for dApp Developers

1. **Define the Computationally Intensive Part.** Identify the resource-heavy segment of your dApp's business logic and rewrite it in Rust using Gear Protocol's Sails library. Compile the program into a Wasm module and generate an IDL (Interface Definition Language) file to describe its interface.
2. **Upload Your Wasm to Ethereum.** Publish your Wasm code as part of a special blob-carrying transaction (EIP-4844). The blob is stored outside Ethereum's main state, but remains accessible via archive nodes. The blob serves as the canonical source of your program's code for Gear.exe initialization.
3. **Initialize Your Program in Gear.exe.** With a single action, activate your Wasm program on Gear.exe. This initialization process uploads the code to Gear.exe, establishes the program's initial state, and automatically deploys a corresponding Mirror Contract on Ethereum. The Mirror Contract serves as the ABI-compatible interface, representing your dApp within the Ethereum ecosystem and facilitating seamless interaction between the two environments.
4. **Leverage Lightning-Fast Computation.** Interact with your program by submitting messages through Ethereum, paying only the transaction fee for message submission. Alternatively, use the RPC interface to access your dApp's functionality directly without incurring additional costs.
5. **Finalization and Real-Time Availability.** Once your transaction is included in an Ethereum block, the computation is finalized and made available according to Ethereum's native finality mechanism. However, Gear.exe's pre-confirmation

mechanism allows your dApp to utilize the results of computations instantly, even before the transaction is finalized on-chain. This feature ensures a near-instantaneous response time, bridging the gap between blockchain finality and real-time interaction.

Uploading programs and interacting with them is quite simple thanks to the developer-friendly tools provided by Gear.exe. Through the Gear IDEA, anyone can easily integrate their Ethereum application with efficient computations on Gear.exe, upload a program, read its state, send a message, and much more.

Security and Executor Selection

Regardless of the integration approach, Executors are critical to Gear.exe's operation. Their selection and management are governed by a decentralized re-staking mechanism facilitated by the Symbiotic Protocol. This process ensures that Gear.exe maintains a secure and scalable compute network by dynamically managing the set of Executors responsible for program execution.

Symbiotic Protocol provides the infrastructure for this election process, serving as an exchange hub for three primary stakeholders: **stakers**, **operators**, and the Gear.exe **network** itself. Together, these actors create a robust and decentralized Executor selection mechanism tailored specifically to Gear.exe's requirements.

Executor Selection Workflow

Gear.exe configures the operator set, establishing parameters such as staking limits and the maximum allowable stake for individual operators. Operators, who run Executor nodes, are elected based on their ability to attract stakers who delegate collateral (**e.g., ERC-20 representations of VARA or other supported assets**) to them. This delegated stake determines their eligibility to serve as active Executors. Once elected, operators participate in program execution and sign results collectively, which are then finalized by the Router Contract. The list of active Executors is continuously updated and pushed to the Router Contract, which governs Gear.exe's decentralized compute infrastructure.

Key elements of the selection process include:

1. **Stake Allocation:** Gear.exe establishes operator sets, defines staking requirements, and locks stake amounts for predefined epochs to maintain network stability.
2. **Symbiotic Vault Integration:** Vaults manage the staking process, allocate collateral to operators, and enforce strategies specific to Gear.exe's execution needs.

Key Actors in Gear.exe Executor Selection

- **Gear.exe Network:** Defines the decentralized infrastructure required to execute programs, configures operator sets, and establishes staking parameters. Gear.exe also ensures that stakers and operators are appropriately rewarded for their contributions.
- **Stakers:** Provide economic security by delegating collateral to operators. In return, they receive a share of the rewards distributed by Gear.exe.
- **Operators:** Operate Executor nodes to execute programs on Gear.exe. They benefit from Symbiotic Protocol's ability to pool stakes across multiple stakers, enabling efficient security for Gear.exe without requiring isolated infrastructure for each staker.
- **Vaults:** Act as intermediaries in the staking process, handling deposits, withdrawals, and slashing events. Vaults also distribute staking rewards based on performance and provide historical data for external reward contracts.

Rewards and Incentives

Gear.exe ensures that stakers and operators are properly incentivized for their roles within the network. Rewards are calculated off-chain by Gear.exe, which generates a Merkle tree structure to facilitate secure and transparent claims by participants. The rewards are divided into:

- **Operator Rewards:** For maintaining and running Executor nodes.
- **Staker Rewards:** For providing the collateral that secures Gear.exe's operations.

This flexible reward logic allows Gear.exe to adapt its incentive structure as needed, ensuring long-term sustainability.

Slashing and Misbehavior

Symbiotic incorporates a robust slashing mechanism to deter malicious behavior. If an Executor produces inaccurate results or engages in misconduct, a special Middleware contract can initiate a slashing request to Symbiotic. Symbiotic's **Slasher module** validates these requests and enforces penalties by reducing the stake of the offending operator or validator. This ensures economic accountability and strengthens the overall integrity of the network.

Attracting Executors

Running a Gear.exe node is designed to be mutually beneficial for operators and stakers. With the added appeal of rewards and the flexibility provided by Symbiotic's Vault and staking mechanisms, many Vara validators are expected to run their own Gear.exe nodes, further bolstering the network's security and scalability.

Economic Model

Gear.exe's economic model is built to support scalable, efficient, and sustainable decentralized applications (dApps). It introduces mechanisms like the **reverse gas model** and a **dual-balance system**, enabling programs to operate seamlessly while maintaining cost transparency and flexibility.

Fundamental Aspects

Reverse Gas Model

Gear.exe uses a reverse gas model, where the cost of executing a program is deducted from the program's **Executable Balance** instead of being paid by the user. This means users only pay the Ethereum transaction fee (in ETH) for sending messages to Mirror Contracts, while the computational costs of Gear program execution are covered by the program itself. This approach simplifies interactions for users and makes programs more accessible.

Dual-Balance System

Programs in Gear.exe maintain two types of balances:

- **Executable Balance:** Dedicated to execution costs. If this balance is zero, the program cannot process messages.
- **Free Balance:** Serves as a wallet for funds earned or deposited into the program. These funds can be withdrawn by the program creator or converted into Executable Balance if the program's logic permits.

Funding Mechanisms

Programs in Gear.exe maintain their Executable Balance through multiple methods:

- **Developer or Sponsor Funding:** The program creator or external sponsors can directly top up the Executable Balance via Ethereum transactions, ensuring the program remains operational without requiring user contributions.
- **Revenue-Based Replenishment:** Programs can replenish their Executable Balance using revenue generated through operational activities, such as fees, commissions, or trading spreads.

- **User-Driven Contributions:** Programs may be designed to accept small payments (value) from users as part of their interactions. A portion of these payments can be converted into Executable Balance, creating a self-sustaining model for the program.

Executor Rewards

When a program executes, the consumed portion of its Executable Balance is deducted and recorded in the Router Contract. These funds are later distributed to Executors and proportionally shared with stakers who delegated collateral, incentivizing reliable computation and maintaining network security.

Transparency and Tracking

Developers and users can query the current Executable Balance of a program via RPC calls using the program's state hash. Mirror Contracts publish the program's state commitment, which can be queried to verify balances and execution costs.

Economic Patterns

Developers can design their programs to follow various economic patterns based on their application's goals and revenue model:

- **Patron Model:** The program creator funds the Executable Balance, allowing users to interact with the program for free.
- **Revenue-Supported Model:** The program generates income (e.g., through fees or commissions) and uses part of this revenue to replenish its Executable Balance.
- **User-Paid Execution:** Users include a small value with their messages, which is converted into Executable Balance, enabling the program to fund itself through user interactions.

Advantages of the Model

- **Clear Cost Allocation:** Users pay only for sending Ethereum transactions, while programs handle computational costs. This distinction simplifies budgeting and encourages dApp adoption.
- **Adaptability:** Developers can implement various funding strategies, tailoring the economic structure to the specific needs of their application.
- **Resource Optimization:** The reverse gas model ensures efficient use of program funds, with balances directly linked to execution and general-purpose needs.

- **Network Incentives:** Executors are rewarded for computation, promoting a robust and secure decentralized execution environment.

Use Cases and Target Audience

The versatility of Gear.exe makes it ideal for a wide range of applications across various industries. Its computational power, scalability, and user-friendly design open up new possibilities for developers and enterprises alike.

In the financial sector, Gear.exe will transform DeFi platforms by enabling faster and more cost-effective execution of complex financial operations. Decentralized exchanges, for example, can benefit from near-instant trade finalization and reduced fees, enhancing their appeal to traders and liquidity providers.

The gaming industry is another area where Gear.exe shines. Gaming platforms can deliver real-time interactions and seamless gameplay. This capability is particularly valuable for multiplayer environments and strategy games that require low-latency processing. Most current Web3 games focus primarily on the marketplace side of gaming, such as NFTs and trading, whereas Gear.exe is designed to enable seamless in-game play, real-time transactions, and mass usage. By addressing the computational demands of modern gaming, Gear.exe paves the way for immersive and scalable Web3 gaming experiences.

Gear.exe also plays a pivotal role in artificial intelligence and machine learning applications. Developers can use its parallel execution capabilities to train and deploy AI models efficiently, leveraging the network's computational power without incurring excessive costs.

In supply chain management, Gear.exe can process large datasets generated by IoT devices off-chain, such as temperature readings or GPS coordinates, and sends only the most relevant insights on-chain. This approach will reduce costs while maintaining the transparency and security of blockchain technology.

Automated Risk Management for DeFi Protocols

Effective risk management is a critical component for decentralized finance (DeFi) protocols. These systems often rely on third-party risk assessment providers to deliver updated risk scores, which must be reflected on-chain to inform portfolio adjustments and other decisions. Traditionally, automating this process requires centralized off-chain components or oracle systems, introducing inefficiencies and potential points of failure.

Gear.exe offers a decentralized solution by enabling direct integration with third-party risk services. Using its high-performance computational environment, risk providers can seamlessly process and transmit updated scores or optimized portfolio recommendations directly to Ethereum. This integration eliminates the need for intermediaries and enhances the speed and reliability of risk management workflows.

For instance, a hedge fund operating on a DeFi platform could leverage Gear.exe to receive real-time risk updates. The platform automatically processes these updates and executes on-chain adjustments, such as portfolio rebalancing, without requiring additional manual intervention. This approach not only streamlines operations but also enhances the responsiveness and security of the entire risk management process.

AI & ML

Gear.exe also unlocks new opportunities in artificial intelligence and machine learning. Thanks to parallel execution and the ability to scale horizontally both at the program level and across the network architecture, workloads such as training, inference, and real-time AI services can run efficiently in a decentralized environment. This makes use cases like decentralized AI models and marketplaces for machine intelligence not only possible, but practical.

High-Frequency Trading (HFT)

High-frequency trading (HFT) requires ultra-low latency, rapid decision-making, and high throughput — characteristics traditionally considered unattainable in decentralized environments. Existing DeFi protocols, constrained by Ethereum's block times and finality delays, struggle to deliver the responsiveness required for advanced market-making or arbitrage strategies.

Gear.exe changes this paradigm. Pre-confirmation enables trading engines to parallel execution model, Gear.exe enables trading engines to execute and confirm operations within milliseconds, while still preserving Ethereum-level security once transactions are finalized. This design makes it possible to build decentralized exchanges on top of Ethereum that rival the speed and efficiency of centralized platforms.

Inspired by pioneering systems like HyperLiquid, Gear.exe extends the concept to Ethereum:

- Sub-second order matching and real-time liquidity updates are possible through pre-confirmed off-chain execution.
- Deterministic and auditable settlement ensures that once Ethereum finality is reached, results are fully secure and tamper-proof.
- Horizontal scalability of the Gear.exe architecture allows trading workloads to be distributed across clusters, removing throughput bottlenecks.

When combined with decentralized AI agents, Gear.exe unlocks even more powerful capabilities. Autonomous trading agents can be trained and deployed directly on Gear.exe, continuously adapting strategies, optimizing liquidity provision, and executing trades at high speed. This synergy of AI-powered decision-making with Gear.exe's low-latency execution layer lays the foundation for the next generation of on-chain financial infrastructure — fast, intelligent, and fully compatible with Ethereum.

Off-Chain Financial Simulations

Large-scale financial simulations, such as Monte Carlo simulations or portfolio optimizations, are essential tools for analyzing risk and making informed decisions in decentralized finance (DeFi). Monte Carlo simulations involve running thousands or even millions of randomized scenarios to model potential outcomes and assess the probability of different events occurring. For example, they are widely used to forecast portfolio performance under varying market conditions, helping to quantify risk and identify optimal strategies for investment.

However, executing these computations directly on Ethereum is both costly and time-consuming due to high gas fees and the network's limited computational capacity. While Layer-2 solutions like Optimistic Rollups and ZK Rollups aim to reduce costs and increase scalability, they still inherit constraints from Ethereum. Optimistic Rollups rely on fraud proofs and extended challenge periods, which delay finality for DeFi applications requiring real-time responses. ZK Rollups, on the other hand, involve computationally expensive proof generation processes, making them less efficient for running large-scale simulations or real-time optimizations.

By contrast, Gear.exe offloads these intensive computations entirely off-chain, allowing DeFi platforms to process simulations or optimizations efficiently while maintaining seamless integration with Ethereum for critical on-chain actions. Once the computations are complete, results such as updated risk scores or optimized portfolio configurations are seamlessly transmitted back to Ethereum. These results can then inform on-chain actions, such as portfolio adjustments, in real time.

For instance, a hedge fund operating on a DeFi platform could use Gear.exe to continuously run advanced risk assessment algorithms. The outputs from these simulations are used to automatically rebalance portfolios on-chain, ensuring optimal performance and minimizing risk exposure. This approach improves the speed and cost of financial decision-making in DeFi environments.

Supply Chain & IoT Data Processing

In supply chain management, real-time data from Internet of Things (IoT) devices plays a crucial role in maintaining efficiency and ensuring quality control. For example, sensors may continuously monitor conditions such as temperature, location, or humidity for shipments. However, processing and storing this vast amount of data directly on-chain is neither cost-effective nor feasible due to the constraints of blockchain scalability and high transaction costs.

Metrics such as temperature thresholds, location tracking, or anomaly detection can be computed within the Gear.exe network, significantly reducing the computational load on the blockchain. Only critical results or actionable alerts are then transmitted on-chain, ensuring cost efficiency and data relevance.

A logistics company managing temperature-controlled shipments can integrate Gear.exe into its supply chain monitoring system. IoT sensor data is processed off-chain, and if a shipment exceeds a predefined temperature threshold, Gear.exe triggers an on-chain event. This event may alert stakeholders or initiate predefined actions, such as rerouting the shipment or adjusting storage conditions.

Off-Chain Voting System

Large-scale decentralized autonomous organizations (DAOs) face significant challenges when implementing on-chain voting systems. The high gas costs associated with processing votes, especially for mechanisms like weighted or quadratic voting, can make the process prohibitively expensive. Additionally, the public nature of on-chain voting compromises member privacy, and as the number of participants grows, scalability becomes a major obstacle.

Gear.exe can offer an efficient alternative by enabling DAOs to process votes off-chain while retaining the integrity and trust required for decentralized governance. Voting logic can be executed within Gear.exe's. Only the final tally and essential results are submitted on-chain, significantly reducing costs and computational overhead.

For example, a DAO with 10,000 members can integrate Gear.exe into its governance framework. Members sign their votes off-chain, ensuring privacy and minimizing gas fees. The Gear program tallies the votes securely and submits the aggregated result to the blockchain.

Future Improvements

One of the most anticipated advancements of Gear.exe's development is the integration of multi-network support. While currently optimized for Ethereum, Gear.exe's design makes it capable to operate across other Layer-1 ecosystems, such as Solana, Near, BNB etc. This multi-chain compatibility could allow developers to leverage Gear.exe's features across a broader range of blockchain environments, fostering greater interoperability and innovation.

Additionally, Gear.exe may incorporate zk-SNARKs to enhance privacy and security. These zero-knowledge proof technologies enable computations to be verified without revealing underlying data, making them ideal for applications requiring confidentiality. As zk-SNARKs become more practical and scalable, their integration into Gear.exe will further solidify its position as a leader in decentralized computation.

Continuous optimization is another key focus. Regular updates to the platform will enhance computational efficiency, reduce latency, and improve the developer experience, ensuring that Gear.exe remains at the forefront of blockchain innovation.

Summary

Gear.exe represents a paradigm shift in decentralized computation. By addressing Ethereum's scalability and cost limitations, it empowers developers to build dApps that deliver unmatched performance and user experience. Its parallel execution capabilities, near-zero gas fees, and seamless integration make it a transformative solution for industries ranging from finance and gaming to supply chain management and artificial intelligence.

As Gear.exe continues to evolve, its focus on multi-network compatibility and cutting-edge technologies will enable it to redefine the possibilities of blockchain development. Developers and users alike are invited to join the Gear community to explore the full potential of this revolutionary platform.

Gear.exe is under active development and is being continually improved each day, with regular commits to the public repository.

Would you like to become part of the Gear community and learn more about Gear.exe? Make sure to join the Gear x Vara Discord or Telegram! Or send an email at hello@gear-tech.io.

Glossary

Actor Model

A computational model where individual components, called actors, operate independently and communicate with each other through messages. This approach enables parallel processing and high scalability, which are integral to Gear.exe's architecture.

Archive Node

An Ethereum node that stores the complete history of the blockchain, including all past states and transactions. Unlike full nodes, which only maintain the current state and recent transaction data, archive nodes retain historical data that allows developers and applications to access detailed information about any block or state from the chain's entire history. Archive nodes are essential for tasks like querying historical balances, accessing older smart contract states, or retrieving blobs uploaded for off-chain processing, as utilized by Gear.exe.

Based Rollups

A type of Layer-2 scaling solution that relies directly on Layer-1 protocols for sequencing and data availability. Unlike traditional rollups that use dedicated infrastructure, based rollups integrate deeply with the underlying blockchain, leveraging its decentralization and security guarantees. This alignment with Layer-1 simplifies operations by removing the need for native tokens or separate trust assumptions. While based rollups benefit from Ethereum's censorship resistance and robust consensus, they inherit its limitations, such as slower transaction finality and shared scalability constraints. Additionally, transaction flexibility is often reduced because sequencing and execution must conform to Layer-1 rules.

Blob

A large binary object stored on the Ethereum network as part of a transaction. In Gear.exe, Wasm code is uploaded as a blob, which resides outside Ethereum's main state but is accessible via archive nodes.

dApp (Decentralized Application)

A software application that runs on a blockchain or decentralized network. dApps are powered by smart contracts and provide users with transparent and trustless interactions without relying on centralized servers.

Executor

A decentralized node within the Gear.exe network responsible for executing Wasm programs. Executors retrieve programs, perform computations, and generate signed results, ensuring the network's reliability and scalability.

Executable Balance

A dedicated balance maintained by a Gear program to cover execution costs. Each time the program processes a message, a portion of this balance is consumed and later distributed to Executors and stakers. If the Executable Balance reaches zero, the program becomes inactive until replenished.

Finality

The point at which a transaction or computational result is considered immutable and irreversible. On Ethereum, finality typically occurs after ~12.8 minutes, but Gear.exe enhances this by providing pre-confirmation mechanisms for near-instant feedback.

Gear Protocol

The foundational framework behind Gear.exe that supports the creation and execution of Wasm programs. It provides the tools and runtime environment necessary for decentralized computation.

IDL (Interface Definition Language)

A file that describes the structure and interface of a Wasm program. Developers use IDL files to define how their Gear programs interact with external systems or smart contracts.

Mirror Contract

A smart contract deployed on Ethereum to act as an interface for a Gear program. Mirror Contracts enable communication between the Ethereum blockchain and off-chain computations performed on Gear.exe.

Mirror ABI Interface

The ABI (Application Binary Interface) exposed by Mirror Contracts on Ethereum. This interface allows Ethereum tools and dApps (e.g., MetaMask, Etherscan) to interact with Gear programs natively, without the need for additional encoding/decoding, by treating them as standard Ethereum smart contracts. Mirror ABI Interfaces act as a transparent proxy layer for Gear programs.

Middleware

A specialized contract that links the Router Contract with the Symbiotic Protocol. It manages responsibilities such as key registration, updates, and validator/operator accountability, including slashing requests. Middleware ensures secure coordination between execution results and staking economics.

Optimistic Rollups

A type of Layer-2 scaling solution that processes transactions off-chain and periodically posts summarized data (state roots) back to the Ethereum mainnet. Optimistic Rollups operate under the assumption that transactions are valid (“optimistically”) unless proven otherwise. To ensure security, they include a challenge period during which anyone can submit fraud proofs to contest invalid transactions. This mechanism provides scalability but introduces delays in transaction finality due to the need for a dispute resolution window.

Operator

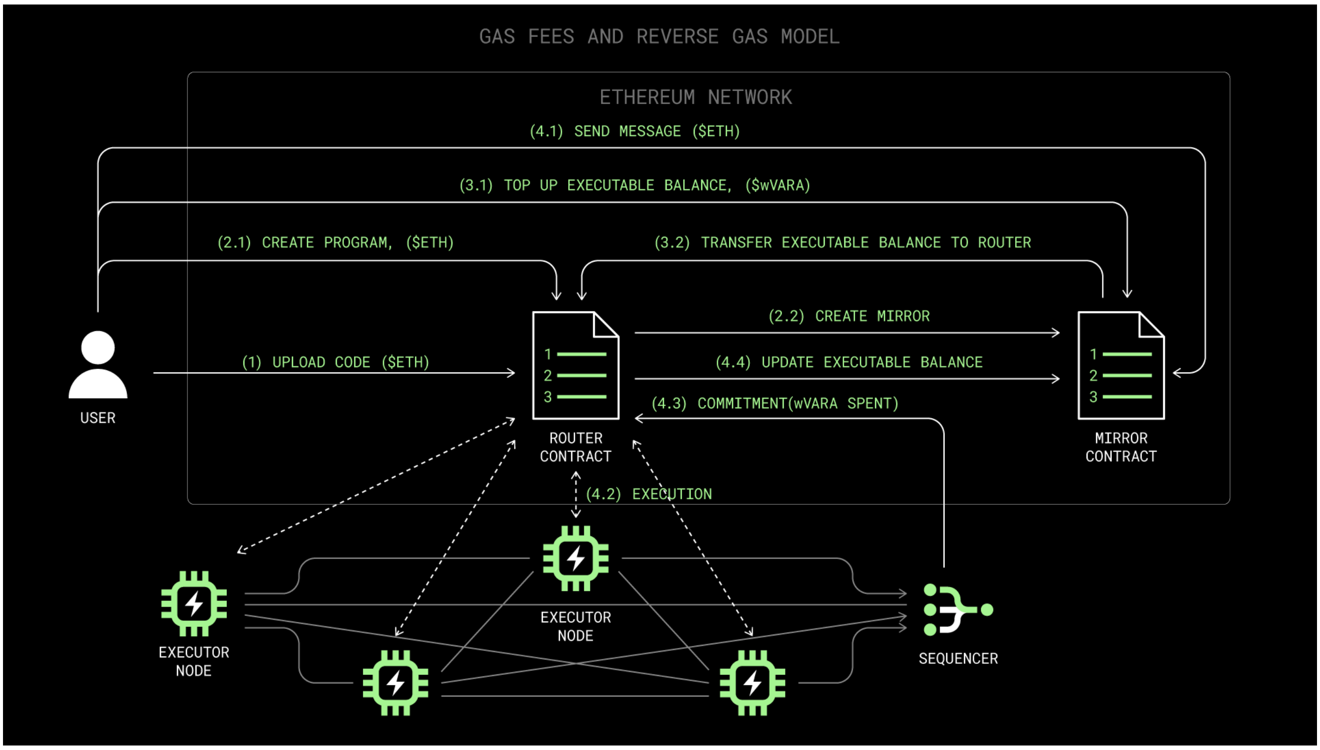
An entity responsible for running Executor nodes within the Gear.exe network. Operators are elected through the Symbiotic Protocol based on delegated stake and are accountable for reliable execution. They represent the economic and operational counterpart to Executors.

Pre-confirmation Mechanism

A feature in Gear.exe that provides computation results immediately after execution, even before the associated transaction is finalized in an Ethereum block. This enables faster feedback for latency-sensitive applications.

Reverse-Gas Model

An approach where developers cover transaction fees for end users, enabling dApps to deliver a seamless user experience. This model is supported by Gear.exe, allowing dApp developers to adopt monetization strategies similar to those used in Web2 applications.



Router Contract

The central smart contract in Gear.exe's architecture that coordinates interactions between Ethereum and the Gear.exe network. It handles program uploads, execution results, and state transitions.

Shared Storage

A blockchain or Layer-2 design feature where all participating nodes or entities share access to a unified state, including data and smart contract storage. This approach ensures consistency and transparency across the network but can limit scalability due to bottlenecks in data retrieval and update operations. Shared storage is a hallmark of traditional blockchains like Ethereum and many rollup solutions, where all transactions and state changes must be reflected across the network. Gear.exe avoids shared storage, instead decentralizing computations and managing state transitions dynamically through its architecture, enabling greater efficiency and scalability.

Slashing

A mechanism that penalizes Executors for malicious behavior or poor performance by reducing their staked collateral. This process ensures the economic accountability of Gear.exe participants and maintains the network's integrity.

Solidity

A high-level, object-oriented programming language specifically designed for writing smart contracts on blockchain platforms like Ethereum. It allows developers to define and implement the logic that powers decentralized applications (dApps).

Symbiotic Protocol

A decentralized restaking system used by Gear.exe to select and manage Executors. It facilitates staking, distributes rewards, and enforces penalties, ensuring a secure and scalable compute network.

Vaults

Intermediaries in the Symbiotic Protocol that manage the staking process for Executors. Vaults handle deposits, withdrawals, and rewards, as well as enforce slashing policies.

Wasm (WebAssembly)

A high-performance, lightweight binary format for executing code. Gear.exe uses Wasm programs to run decentralized computations efficiently and securely.

ZK Rollups (Zero-Knowledge Rollups)

A Layer-2 scaling solution that uses zero-knowledge proofs to validate transactions off-chain and post verified summaries on-chain. ZK Rollups employ cryptographic proofs (such as zk-SNARKs or zk-STARKs) to ensure the correctness of the batch without revealing the underlying transaction data. This approach enhances scalability, reduces gas costs, and offers faster finality compared to Optimistic Rollups, but at the cost of higher computational demands for generating proofs.