Neon EVM Composability:
A Unified Framework for Ethereum–Solana Interaction

White Paper

Miroslav Nedelchev[1], *Grzegorz Gancarczyk*[1]

Reviewed by:

Oleg Sukharev[2], *Andrey Falaleev*[2]

[1]Authors (miroslav.nedelchev@neonfoundation.io, grzegorz.gancarczyk@neonfoundation.io)

[2]Reviewers (os@neonfoundation.io, a@neonfoundation.io)

April 11, 2025

Abstract

This white paper introduces the composability functionality within Neon EVM, a feature that enables EVM–based smart contracts to directly interact with Solana blockchain programs. By combining Ethereum's developer tools and execution environment with Solana's processing capabilities, Neon EVM provides a framework for interoperable dApp development across both blockchains.

We address specific technical considerations crucial to Ethereum–Solana interoperability, including account ownership discrepancies, bridging ERC-20-compatible (SPL-wrapped) tokens, transaction atomicity, and revert semantics. Additionally, we outline the composability architecture, security and validation mechanisms, instruction preparation guidelines, program-derived address (PDA) usage, associated token account (ATA) management, and specialized payer account models required for Solana account creation and rent exemption.

The technical explanations provided herein are intended to offer developers and blockchain stakeholders a thorough understanding of Neon EVM's composability mechanism, facilitating integration between Ethereum–compatible smart contracts and native Solana functionalities.

Contents

1 Introduction

The composability mechanism provides interoperability, allowing Ethereum-based smart contracts to directly invoke and interact with programs native to Solana. By abstracting away complexities inherent to cross-chain interactions—such as divergent account models, token standards, transaction semantics, and revert mechanisms—Neon EVM offers developers a unified execution environment. This environment enforces transaction atomicity and manages accounts without additional developer intervention.

This white paper presents an in-depth technical description of Neon EVM composability. We outline its architectural design, security considerations, operational guidelines, account management strategies, and recommended practices for instruction preparation. The objective is to equip blockchain developers, architects, and stakeholders with knowledge enabling them to leverage Neon EVM's unified composability solution effectively.

2 Executive summary

2.1 Overview of composability

Composability in Neon EVM is an interoperability mechanism enabling Ethereum Virtual Machine (EVM)-based smart contracts to invoke and integrate with native programs on the Solana blockchain. It provides Solidity-based contracts with direct access to Solana's functionality, extending Ethereum's development paradigms into Solana's execution environment.

The primary features of Neon EVM composability include:

- **Direct cross-chain calls**: Solidity contracts within Neon EVM can invoke Solana program instructions to perform token bridging (SPL-to-ERC20 compatibility) and other operations.
- **Atomic transaction semantics**: Neon EVM ensures strict atomicity of cross-platform transactions. Solana instructions invoked via composability either execute completely or revert fully, thereby preventing partial state transitions. This transactional atomicity simplifies smart contract logic and enhances reliability in cross-chain interactions.
- **Internal management of multi-step transactions**: The composability mechanism internally manages scenarios involving complex or multi-phase transaction flows, accommodating Solana's transaction-size and compute-unit constraints. Developers interact with composability through familiar Ethereum-based workflows, without explicit management of underlying Solana-specific execution details.
- **Account and state management abstractions**: Composability incorporates management of program-derived addresses (PDAs), associated token accounts (ATAs), and specialized payer accounts. These mechanisms facilitate token custody, rent-exempt account provisioning, and secure instruction signing, respectively, ensuring Ethereum-to-Solana state transitions and secure cross-platform interactions.

2.2 Composability technical approach

Neon EVM composability facilitates cross-platform interactions between Ethereum-compatible smart contracts and native Solana programs through an approach that abstracts away inherent complexities. This interoperability is implemented using several critical components and design considerations:

- **Precompile extensions**: A dedicated precompile contract (at address `0xFF000000000000000000000000000006`)) provides a standardized Solidity interface for interacting with Solana. Developers prepare Solana instruction data within their Solidity code and pass it to this precompile. The precompile then requests the execution of this instruction on Solana, handling aspects like account resolution and atomicity enforcement.
- **Token interoperability via ERC20ForSPL**: The ERC20ForSPL interface serves as an Ethereum-compatible abstraction for managing SPL tokens. Unlike traditional Ethereum token standards, ERC20ForSPL does not maintain state within Solidity contract storage. Instead, it directly operates on state within Solana's program-derived accounts (PDAs) and associated token accounts (ATAs). Token transfers, balances, and approvals are handled through Solana-native account modifications, ensuring efficiency and consistency across platforms.

- **Account and signer management**: Program-derived addresses (PDAs) are deterministically generated accounts exclusively managed by the Neon EVM program, ensuring secure ownership and control of token and data accounts on Solana. For SOL provisioning (required for Solana account creation and rent exemption), Neon EVM introduces dedicated payer accounts uniquely associated with each Ethereum msg.sender. When a developer passes a Solana instruction requiring SOL (e.g., for account initialization) to the composability precompile, the necessary SOL is provisioned via this associated payer account during the transaction execution. These payer accounts also function as signers for Solana instructions requiring explicit authorization, providing secure account creation and instruction signing capability.

- **Execution and resource management**: Composability transactions maintain atomic execution semantics across Ethereum–Solana interactions, internally handling complexities such as transaction-size limits, compute-unit constraints, and conditional state updates without developer intervention. By managing resource allocation and transaction validation internally, Neon EVM provides consistent transaction outcomes and prevents partial state modifications.

- **Validation and security framework**: A validation protocol verifies account ownership, instruction legitimacy, and operator account isolation. These validation steps mitigate security risks, unauthorized account access, and improper state changes. By isolating operator accounts and carefully managing transaction execution contexts, Neon EVM composability ensures robust security across all cross-chain interactions.

3 Composability overview

3.1 Definition and motivation

Composability within Neon EVM refers to the technical capability enabling Ethereum Virtual Machine (EVM)-compatible smart contracts deployed on Neon EVM to directly invoke and manage interactions with native programs on the Solana blockchain. This interoperability layer allows Solidity-based contracts to access Solana-specific features, functionality, and on-chain programs without departing from the Ethereum development environment or requiring separate integration efforts.

The motivation for composability arises from distinct technical strengths present within the Ethereum and Solana ecosystems. Ethereum offers extensive smart contract tooling, a mature developer experience, and a widely adopted application ecosystem. In contrast, Solana provides parallel execution, low latency, high transaction throughput, and resource-efficient computation. However, historically these blockchain ecosystems have operated independently, with friction in integrating functionality across platforms.

By implementing composability, Neon EVM addresses these integration challenges, providing developers a mechanism to extend Solidity smart contract capabilities into the Solana execution environment. Key interoperability issues addressed through composability include:

- **Account ownership models**: Ethereum's contract-storage approach significantly differs from Solana's account-based data storage model. Composability abstracts away these differences by internally managing interactions via deterministic Solana accounts, such as program-derived addresses (PDAs) and associated token accounts (ATAs).
- **Token standard compatibility**: ERC20-compatible SPL-wrapped tokens are managed through composability, providing Solidity developers with a familiar token interface without explicit handling of Solana-native token accounts.
- **Atomic transaction semantics**: Composability ensures transactions spanning Ethereum contracts and Solana instructions remain atomic. Operations executed on Solana either complete entirely or revert fully, maintaining consistent transaction states across both blockchain environments.
- **Transaction execution management**: Neon EVM internally manages transaction complexities such as Solana's compute-unit constraints and instruction data limits. This abstraction simplifies developer workflows and ensures predictable outcomes for cross-platform transactions.

3.2 High-level architecture

The Neon EVM composability feature is architected to facilitate interactions between Ethereum Virtual Machine (EVM)-compatible smart contracts and native Solana programs. Its architecture integrates Ethereum's execution environment with Solana's blockchain infrastructure through several coordinated components:

- **Neon EVM program**: Deployed directly onto the Solana blockchain, the Neon EVM program serves as the runtime environment for executing Ethereum-compatible bytecode. It interprets EVM transactions, manages contract storage, and securely handles state transitions, integrating with native Solana programs and account management mechanisms.

- **Precompile contract interfaces**: Specialized precompiled contracts extend standard EVM execution, handling Solidity calls to invoke Solana-compatible instructions. These precompiles abstract complex account operations, instruction encoding, token transfers, and state synchronization between the EVM and Solana, providing developers with standardized interfaces for Solana interoperability.
- **Account and token management layer**: Neon EVM composability utilizes deterministic program-derived addresses (PDAs), associated token accounts (ATAs), and payer accounts to securely manage tokens, account creation, rent exemption, and required signer authorization on Solana. This structured management ensures secure, predictable handling of assets and account states across platforms.
- **Neon proxy and emulation service**: A client-side proxy component prepares Solana transactions based on Ethereum-compatible transactions. This layer handles account discovery, transaction emulation, validation of execution constraints (such as compute-unit budgets and instruction data sizing), and transaction packaging, thereby ensuring consistent execution and predictable results.
- **Security and transaction validation**: Neon EVM employs validation protocols to ensure the integrity and correctness of cross-chain interactions. This includes verifying account permissions, transaction signatures, and enforcing secure isolation between operators, Ethereum-based contracts, and Solana accounts, effectively mitigating potential cross-chain execution risks.

3.3 Support for Arbitrary Solana Programs

Neon EVM composability provides support for interactions with arbitrary Solana programs. While initial implementations focused primarily on widely used, standard Solana programs, the composability framework is designed to accommodate generalized invocation of any Solana program implemented using the Berkeley Packet Filter (BPF).

To facilitate interactions with custom or third-party Solana programs, Neon EVM leverages precompiled contract interfaces. Developers specify essential parameters such as the Solana program ID, required accounts, and instruction-specific data within Solidity contracts. The Neon EVM program translates these Solidity instructions into correctly formatted Solana instructions at runtime, ensuring execution on the Solana blockchain.

Key capabilities enabling support for arbitrary programs include:

- **Dynamic account discovery and preparation**: Neon EVM identifies and prepares the necessary Solana accounts based on the contract-specified program interactions, managing permissions and ownership through program-derived addresses (PDAs) and associated token accounts (ATAs).
- **Flexible instruction encoding**: Developers define instruction data and parameters directly within Solidity contract calls, which Neon EVM dynamically encodes into appropriate Solana instruction formats at execution time. This flexibility ensures compatibility with diverse Solana program interfaces, regardless of complexity.
- **Automatic instruction execution and atomicity**: Composability ensures instructions targeting arbitrary Solana programs execute atomically within the context of the originating Ethereum transaction. This design guarantees that the Solana instructions either succeed completely or revert entirely, preserving the integrity and consistency of

cross-platform state transitions. For example, if a single transaction contains multiple composability calls (invoking Solana instructions 1, 2, and 3 sequentially), and instruction 3 fails during Solana execution after 1 and 2 have successfully completed, the *entire* transaction reverts. This includes rolling back the effects of the successful Solana instructions (1 and 2) as well as any state changes made by the Solidity contract logic executed prior to the failing instruction 3.

- **Secure authorization and signing**: Neon EVM manages secure instruction authorization using dedicated payer accounts associated with Solidity contracts, fulfilling Solana's signer requirements without exposing private keys or requiring manual intervention.

4 Composability execution model

4.1 Conceptual overview

Neon EVM composability manages the execution of Ethereum-compatible smart contracts that invoke Solana-native program instructions. From the perspective of Solidity developers, interactions with Solana programs appear consistent with familiar Ethereum execution semantics. Internally, Neon EVM handles encoding, preparation, execution, and validation of cross-platform instructions. Note that while regular Solidity logic is translated into corresponding Solana operations, the composability feature requires Solidity contracts to prepare Solana instruction data that is processed natively on Solana.

The composability mechanism operates with strict atomic transaction semantics. Each Ethereum-to-Solana transaction initiated by a Solidity contract executes as a single atomic entity, meaning all embedded Solana instructions either fully succeed or revert entirely. This approach ensures state integrity and eliminates the need for developers to explicitly handle partial or intermediate states.

Internally, Neon EVM employs a carefully orchestrated process to execute cross-platform interactions:

- **Transaction preparation and account resolution**: The Neon proxy dynamically identifies required Solana accounts, determines necessary instruction data, and encodes transaction details based on the Ethereum transaction submitted by users or smart contracts.
- **Dynamic instruction execution**: Neon EVM securely executes Solana-compatible instructions based on the actions defined in Solidity. It manages the execution context, ensuring that instruction data, account permissions, token balances, and computational resources align with Solana's operational constraints and account structures.
- **Internal resource management**: Neon EVM handles resource allocation, including compute-unit budgets and transaction data-size constraints inherent to Solana's blockchain environment. This internal management ensures consistent transaction outcomes without requiring explicit developer awareness or management.
- **State finalization and atomicity enforcement**: Upon execution completion, Neon EVM ensures atomic state transitions. Transactions either commit fully to the blockchain state or revert completely, preserving cross-chain consistency and simplifying contract-level logic.

4.2 Execution workflow

Neon EVM composability execution workflow provides a structured process enabling Ethereum-compatible smart contracts to invoke and execute Solana-native instructions. This workflow manages complexities involved in cross-chain instruction translation, validation, and execution, ensuring consistent outcomes. The general composability execution workflow involves the following technical steps:

1. **Solidity transaction initiation**: A Solidity smart contract or externally-owned account (EOA) initiates an Ethereum-compatible transaction intended to interact with a Solana program. The transaction may include specific instruction parameters, token operations, and account management details defined within the Solidity contract.

2. **Neon proxy account preparation and validation**: The Neon proxy dynamically analyzes and emulates the Solidity transaction to identify all required Solana accounts (including PDAs and associated token accounts), necessary token balances, and compute-unit requirements. The proxy assembles this information into a properly formatted Solana transaction.

3. **Transaction encoding and Solana instruction generation**: Solidity-defined operations are translated into Solana-compatible instructions through precompiled contract interfaces. This includes encoding necessary program IDs, account addresses, instruction-specific data, and managing signer permissions. Token transfers and account initialization requests are prepared according to Solana's account model requirements, ensuring the transaction meets Solana's runtime constraints.

4. **Atomic instruction execution**: Neon EVM securely executes the encoded Solana instructions as an atomic transaction. All included instructions either succeed in entirety or revert as a complete unit, maintaining consistent state across Ethereum and Solana execution contexts. Developers receive transaction receipts indicating final outcomes, aligned with standard Ethereum transaction receipt formats.

5. **Resource and state finalization**: Upon successful execution, Neon EVM commits state transitions to Solana blockchain accounts, securely updating token balances, account ownership, and any associated data. Unused computational resources and unused SOL (provided initially through the payer accounts) are automatically returned to the Neon EVM operator, optimizing resource usage efficiency.

6. **Transaction outcome reporting**: Transaction results, including event logs and state changes, are reported back to the originating Ethereum-compatible environment. Solidity smart contracts or user interfaces can process these results through standard Ethereum transaction receipts, maintaining a familiar development workflow.

4.3 Token bridging via ERC20ForSPL

Neon EVM facilitates interoperability with Solana tokens through the ERC20ForSPL interface, a Solidity-compatible token abstraction that enables management and transfer of SPL-standard tokens within Ethereum-compatible smart contracts. Unlike conventional Ethereum token standards such as ERC20, ERC20ForSPL's primary distinction is that it does not maintain token balances within Solidity contract storage; these are read directly from the underlying Solana accounts. However, it does manage standard Ethereum allowances within Solidity storage using the familiar `approve` method. Solana-specific allowances are handled separately via a distinct `approveSolana` function, which operates directly on Solana state. Consequently, core operations like balance checks and transfers primarily manipulate token state stored within Solana blockchain accounts.

Key technical attributes of ERC20ForSPL bridging include:

- **Direct Solana token account manipulation**: Token balances, transfers, approvals, and other token operations performed through ERC20ForSPL translate directly into modifications of Solana-associated token accounts (ATAs) and program-derived addresses (PDAs). This approach leverages Solana's native account management, providing accurate, real-time token state without redundant or intermediate storage in Ethereum-style contracts.

- **Solidity-compatible token interface**: ERC20ForSPL exposes an ERC-20-compatible Solidity interface, allowing Ethereum-based smart contracts and user interfaces to interact with SPL tokens using familiar function signatures such as `transfer(address,uint256)`, `transferFrom(address,address,uint256)`, and `balanceOf(address)`. Underlying Solana account operations remain entirely abstracted from the Solidity developer's perspective.
- **Efficient token account handling**: Tokens initially held within Neon EVM-managed PDAs may require transfer to contract-controlled ATAs to facilitate certain composability operations. Advances in ERC20ForSPL implementations continue to simplify these token transfer and custody requirements, reducing overhead and improving the efficiency of token bridging processes.
- **Atomicity and security**: ERC20ForSPL token operations executed through Neon EVM composability inherit strict atomic execution semantics. Token transfers and interactions with Solana programs occur within atomic transactions, either completing entirely or reverting fully, thereby ensuring secure and predictable token state transitions.

Through this token-bridging mechanism, Neon EVM composability effectively unifies Ethereum-compatible smart contracts with Solana's native token standards, providing developers access to the token ecosystems of both blockchains within a consistent, Ethereum-compatible development environment.

4.4 Compute unit considerations and instruction preparation

Composability in Neon EVM leverages Solana's atomic transaction model, executing multiple instructions within a single, indivisible transaction. To maximize the effectiveness of composability calls and ensure predictable transaction outcomes, the following best practices should be observed during instruction preparation and execution:

1. While not a strict requirement, it is highly recommended as a best practice to perform as much Solidity logic and instruction data generation as possible *before* initiating the first composability request within a transaction. This approach optimizes resource usage by streamlining execution and reducing the computational overhead consumed during the atomic on-chain interaction phases. Conversely, executing significant Solidity logic *between* consecutive composability calls consumes part of the transaction's overall compute budget, increasing the likelihood of reaching Solana's Compute Unit (CU) limit prematurely.
2. Minimize intermediate Solidity logic between consecutive composability calls. By doing so, developers can significantly optimize the available computational resources, resulting in more reliable transaction execution.
3. In scenarios involving sequential composability instructions, where the outcome of an earlier instruction serves as input for a subsequent instruction (e.g., Instruction #1: swapping Token A for Token B, followed by Instruction #2: depositing Token B into a lending protocol), the recommended approach is:
    - Fully prepare instruction calldata and account metadata for the first instruction (#1).
    - For dependent instructions (#2 onwards), predefine only the required Solana program IDs and account metadata, leaving the instruction calldata open or placeholder.

  - After the initial composability instruction successfully executes, dynamically retrieve the resulting state (e.g., token balances) within the same atomic transaction execution flow and finalize the calldata for the subsequent instruction accordingly. This approach ensures accuracy and optimal resource utilization within the atomic context.

4.5 Atomicity, error handling, and simulation considerations

Composability transactions within Neon EVM maintain strict atomic execution semantics, ensuring that all invoked Solana instructions execute completely or revert entirely, without intermediate state transitions. This atomicity simplifies smart contract logic, as partial execution results do not require explicit handling by developers.

Several considerations related to atomicity, error handling, and simulation are important for efficient composability implementation:

- **Atomic execution model**: Transactions invoking Solana programs through composability adhere strictly to atomic transaction rules. Should any Solana instruction within a composability transaction fail—such as due to unmet logical conditions in decentralized exchange interactions (e.g., insufficient slippage tolerance)—the entire transaction reverts. Developers can rely on predictable transaction outcomes, simplifying error handling at the contract level.
- **Simulation and state prediction**: Pre-transaction simulations or off-chain emulations (e.g., via `eth_estimateGas` or Neon Proxy emulation) can detect certain errors, such as incorrect instruction formatting or missing Solana account data. Transactions failing these pre-execution validations will not proceed to broadcasting. However, exact state changes—such as precise token reserve outcomes in dynamic market conditions—cannot be guaranteed via simulations, as actual on-chain state might vary between simulation and execution due to concurrent blockchain transactions.
- **Optimizing for simplicity and predictability**: Composability excels particularly in straightforward interactions such as token swaps, token minting, and single-operation deposits. Developers managing complex transaction sequences or multi-step conditional interactions should follow recommended best practices, preparing instruction data dynamically based on real-time on-chain outcomes for accurate and efficient execution, as described in the previous section.

5 Technical details

This section offers an in-depth look at the internal mechanisms, data structures, and workflows that enable Composability in Neon EVM. Building on the high-level overviews, we focus here on how Neon EVM manages Solana accounts, executes precompile functions, and preserves security and atomicity.

5.1 Neon EVM program architecture

The Neon EVM is implemented as a native Solana on-chain program, architecturally designed to facilitate the execution of Ethereum Virtual Machine (EVM)-compatible smart contracts directly within Solana's blockchain environment. It serves as the primary runtime environment for Solidity-based contracts, managing transaction validation, EVM bytecode interpretation, storage management, and integration with native Solana accounts and instructions.

The core components and operational logic of the Neon EVM architecture include:

- **EVM execution engine**: At the heart of Neon EVM is an Ethereum Virtual Machine interpreter implemented directly on Solana. It supports standard Ethereum opcodes, enabling execution of compiled Solidity contracts. This execution engine manages the contract lifecycle, maintains EVM state transitions, and executes bytecode instructions according to Ethereum specifications, ensuring compatibility and outcomes consistent with Ethereum behavior.
- **Solana-native account integration**: Neon EVM uses Solana's native account model extensively. Contract state, balances, code storage, and transaction metadata are stored in Solana accounts. To securely isolate Ethereum state, Neon EVM manages program-derived addresses (PDAs) deterministically generated from contract addresses and predefined seeds. Each PDA is exclusively managed by Neon EVM, guaranteeing secure and deterministic storage of EVM state within the Solana environment.
- **Composability precompile interfaces**: Neon EVM integrates specialized precompiled contracts to translate Solidity instructions into Solana-compatible operations. These precompiles handle complex interactions—such as token transfers, account creations, and external Solana program invocations—by dynamically preparing and executing Solana instructions at runtime, simplifying Ethereum-to-Solana cross-platform communication.
- **Account creation and rent-exemption management**: Neon EVM manages Solana account creation requirements, automatically allocating SOL via dedicated payer accounts to achieve rent exemption for newly created accounts. Unused SOL allocations are automatically returned upon transaction completion, ensuring efficient resource utilization without requiring explicit developer management.
- **Resource and compute unit allocation**: Neon EVM internally manages computational resources, adhering to Solana's compute-unit limits and transaction size constraints. It dynamically estimates computational requirements for Solidity transactions executed as Solana instructions, ensuring that transactions either succeed within these constraints or revert predictably, thereby simplifying transaction reliability and consistency.
- **Neon proxy integration and transaction emulation**: A specialized Neon proxy layer off-chain emulates transaction execution prior to broadcasting, performing pre-validation of computational resource usage, account availability, instruction format correctness, and

account permissions. This emulation step provides early detection of transaction issues, improving reliability and reducing on-chain execution errors.

- **Secure state isolation and validation mechanisms**: The Neon EVM architecture employs account isolation, validation checks, and permission management to protect contract execution integrity. All transaction inputs, account interactions, and state transitions are explicitly validated and isolated, mitigating unauthorized access and ensuring secure, predictable state updates.

5.2 Precompile extensions for Solana calls

Neon EVM provides specialized precompiled contracts designed explicitly to facilitate direct Solidity-to-Solana interactions, handling cross-platform communication. These precompile contracts function as built-in extensions to the standard EVM instruction set, automatically encoding Solidity method invocations into the corresponding Solana-compatible instruction format for native execution. In the context of composability, Solidity contracts are expected to prepare the Solana data directly, rather than undergoing a translation process.

Furthermore, each individual instruction request to Solana passed to the precompile at address `0xFF00000000000000000000000000000000000006` is executed as a CPI call from the Neon EVM program to the targeted Solana program (as specified by the `programId` in the instruction). This delegation ensures that the Neon EVM program acts on behalf of the original `msg.sender`, securely relaying instructions and preserving the intended authority and context throughout the execution process.

The precompile extensions handle critical operational responsibilities, including:

- **Solana instruction encoding and execution via CPI**: Solidity contract calls directed toward the composability precompile (`0xFF00...06`) contain developer-formatted Solana instruction data. The precompile interface validates this input. Subsequently, the Neon EVM Solana program itself executes the requested operation by making a Cross-Program Invocation (CPI) call to the target Solana program ID specified within the instruction data. This CPI is performed by the Neon EVM program acting securely on behalf of the original Ethereum msg.sender, effectively delegating the instruction execution within the Solana environment. Precompiles also manage the inclusion of necessary account metadata consistent with Solana program interfaces.

- **Dynamic account discovery and management**: Precompile extensions automatically identify and resolve Solana accounts required for specific instructions, including program-derived addresses (PDAs), associated token accounts (ATAs), and explicitly defined accounts specified within Solidity calls. Account metadata—including ownership, permissions, and signer authority—are managed by the precompile logic, eliminating explicit developer intervention.

- **SOL provisioning via payer accounts**: Many Solana operations (e.g., token account creation, data storage initialization) require SOL for rent exemption and account creation. Precompile extensions internally handle SOL provisioning via dedicated payer accounts deterministically associated with the original Ethereum caller (`msg.sender`). Upon instruction execution, any unspent SOL allocations are automatically returned to the Neon EVM operator, managing resources efficiently.

- **Secure signer authorization**: Certain Solana instructions require authorized signer accounts to execute specific operations (e.g., metadata creation via Metaplex). Precompile extensions manage these signer requirements through associated payer accounts, ensuring instructions are correctly authorized without private-key exposure or additional manual management by the contract developers.

- **Atomicity and error management**: Precompile extensions enforce strict atomic transaction semantics. Transactions invoking Solana instructions either execute fully or revert entirely upon encountering any execution error. Developers do not explicitly handle partial execution scenarios, simplifying contract logic and ensuring consistent, secure cross-platform state transitions.
- **Off-chain transaction emulation support**: Precompile contracts integrate with the off-chain Neon proxy layer, supporting pre-transaction emulation and validation processes. This emulation includes verifying compute-unit constraints, instruction correctness, and account accessibility, identifying errors prior to on-chain execution, and reducing execution reverts.

5.3 Program Derived Addresses (PDAs) and ERC20ForSPL token management

Program derived addresses (PDAs) form an essential part of Neon EVM's internal account management, enabling deterministic, secure, and private-keyless interaction with Solana blockchain accounts. PDAs are generated deterministically using predefined seeds, such as Ethereum contract addresses or user-specific identifiers, ensuring predictable and collision-free address derivation. These accounts are exclusively managed by the Neon EVM program, thus maintaining strict security, deterministic account management, and precise state handling.

Key functions and characteristics of PDAs in Neon EVM include:

- **Deterministic account generation**: PDAs are derived using deterministic methods, typically combining Ethereum addresses, specific identifiers, and fixed seeds to produce Solana addresses that require no private keys. This ensures secure and predictable interactions without manual account management.
- **Secure asset custody and isolation**: PDAs securely hold and manage assets, tokens, or data for EVM-based contracts. As no external parties hold keys to these accounts, asset custody remains entirely secure and inaccessible except via properly authorized Neon EVM transactions.
- **Integration with Solana token accounts**: Neon EVM utilizes PDAs to store SPL token balances associated with EVM addresses. Token state changes, such as balance transfers and approvals performed by Solidity contracts via ERC20ForSPL, directly modify the state stored within these PDA-managed Solana token accounts.

The ERC20ForSPL interface provides:

- **Solana token integration**: Token operations in Solidity—such as `transfer(address,uint256)`, `transferFrom(address,address,uint256)`, and `balanceOf(address)`—are internally executed by directly manipulating the associated Solana token account state, eliminating redundant storage and ensuring accurate real-time token balances.
- **Efficient token account handling via ATAs**: Associated token accounts (ATAs), explicitly owned by smart contracts or user-controlled addresses, simplify token management

by providing direct control of tokens during composability operations. Recent improvements in ERC20ForSPL enable tokens to be deposited directly into these ATAs, potentially bypassing intermediate PDAs and reducing operational overhead and complexity in certain scenarios.

- **Atomic token state updates**: ERC20ForSPL inherits strict atomicity provided by Neon EVM composability transactions. Token transfers and interactions executed through composability are performed atomically, ensuring all token state updates either commit entirely or revert completely, maintaining state consistency and integrity.

5.4 Payer account for SOL provisioning and instruction signing

Interactions with Solana programs frequently involve account creation or initialization, which requires payment of a specific amount of SOL to ensure accounts meet the Solana blockchain's rent-exemption criteria. In Neon EVM's composability model, transactions originate from Ethereum-compatible (EVM-based) wallets holding NEON tokens (used for gas), not necessarily SOL. To bridge this gap, Neon EVM introduces a special account mechanism known as the *payer account*.

The payer account has a deterministic one-to-one association with the original `msg.sender`—the Ethereum address initiating the composability request. This payer account serves two primary purposes:

- **Provisioning SOL for account creation**: When a smart contract requires the creation or initialization of Solana accounts (e.g., token accounts or general data-storage accounts) via composability, the necessary SOL for account rent-exemption must be provided upfront. The payer account fulfills this requirement by temporarily receiving SOL from Neon EVM operators upon request. For example, initializing a typical SPL token account requires allocating approximately 165 bytes, corresponding to roughly 0.0016 SOL (rent varies). During the composability instruction, the contract implicitly requests the specific SOL amount needed for the accounts it creates, which the Neon EVM program facilitates by transferring SOL from operators to the payer account to cover these costs.

  After the transaction concludes, any unspent SOL balance allocated for rent but ultimately not needed (e.g., if an account already existed or the transaction reverted before creation) from the payer account is automatically returned to the Neon EVM operator. Thus, the underlying mechanism ensures cost efficiency, avoiding excess SOL consumption or prolonged fund storage in the payer account. The user's cost is reflected in the NEON gas fees covering the entire operation, including the SOL provisioning service.

- **Signing instructions requiring authorization**: Certain Solana program instructions (e.g., associating SPL token metadata via Metaplex standards, or interactions requiring the authority over an account to sign) require explicit signatures from authorized accounts. The payer account provides this necessary authorization capability when the EVM contract logic dictates it. Smart contracts may utilize the payer account as a signer for Solana instructions whenever explicit authorization from an account associated with the original caller (`msg.sender`) is mandated. The payer account thus facilitates interoperability between EVM-compatible smart contract logic and native Solana signing requirements, acting as a proxy signer controlled by Neon EVM on behalf of the user's intent.

Behind the scenes, Neon EVM automatically derives and manages a unique payer account address for each Ethereum `msg.sender`. Smart contracts can retrieve the Solana address of their associated payer account via the `getPayer()` method provided by the composability precompile contract at address `0xFF00000000000000000000000000000000000006`.

Importantly, the SOL amounts disbursed through the payer account originate from the Neon EVM operator infrastructure, funded by the overall transaction fees paid in NEON by the user. The initial EVM transaction signer approves NEON token spending for gas and fees only once—during the initial transaction signing. The NEON gas limit approved strictly caps the total resources expendable for the execution of all subsequent instructions associated with that transaction, including the cost of SOL provisioning. It is not possible to exceed this predetermined NEON gas limit.

5.5 Detailed composability execution and internal transaction handling

Neon EVM composability employs a detailed internal transaction handling mechanism to manage complex, multi-step interactions with Solana programs. This mechanism provides deterministic and atomic cross-chain transaction execution.

Internal transaction scheduling and resource allocation

For transactions involving multiple dependent Solana instructions invoked via composability calls within a single Ethereum transaction, Neon EVM manages the sequence atomically. It evaluates the state after each instruction to potentially inform subsequent ones, all while managing the overall compute budget. Figure 1 illustrates this internal sequencing.

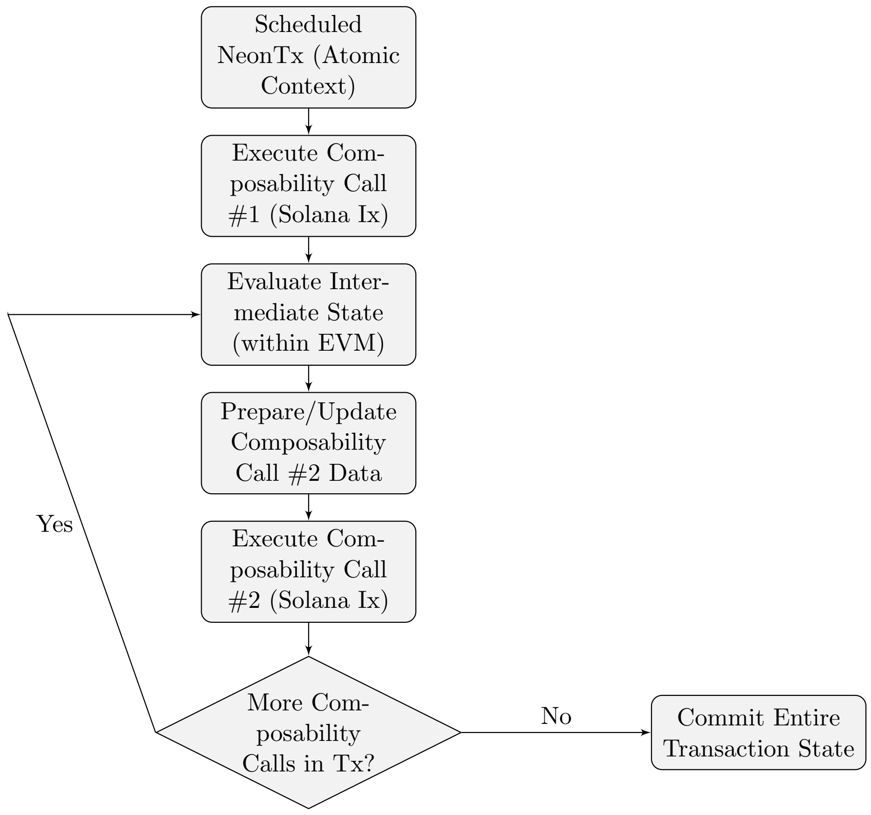

Scheduled NeonTx (Atomic Context)
Execute Composability Call #1 (Solana Ix)
Evaluate Intermediate State (within EVM)
Prepare/Update Composability Call #2 Data
Execute Composability Call #2 (Solana Ix)
More Composability Calls in Tx?
Yes
No
Commit Entire Transaction State

Figure 1: Internal Neon transaction scheduling for sequential composability calls within one atomic transaction

Error handling and atomic revert semantics

Atomicity is paramount. If any Solana instruction invoked through composability fails, or if resource limits are exceeded, the entire transaction's state changes on Solana (and consequently, the perceived state in Neon EVM) are discarded. Figure 2 depicts this.

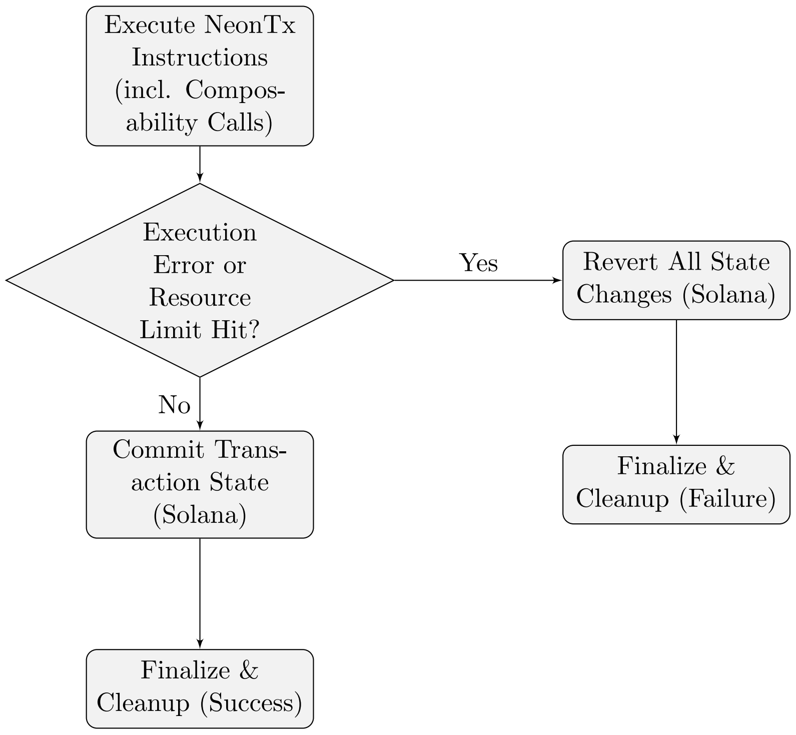

Execute NeonTx Instructions (incl. Composability Calls)
Execution Error or Resource Limit Hit?
Yes
Revert All State Changes (Solana)
Finalize & Cleanup (Failure)
No
Commit Transaction State (Solana)
Finalize & Cleanup (Success)

Figure 2: Neon EVM internal error handling and atomic revert semantics

This explicit internal composability transaction-handling logic ensures deterministic and secure execution of complex multi-step cross-chain transactions within the atomic guarantees of a single Solana transaction.

5.6 Neon proxy and emulation layer

The Neon proxy and emulation layer acts as an essential intermediary between Ethereum-compatible clients (like wallets or dApp frontends) and the Neon EVM on Solana. Its primary responsibility involves transaction preprocessing, off-chain emulation, computational resource estimation, and final transaction encoding, ensuring the feasibility and validity of Ethereum-style transactions within Solana's operational constraints before they are submitted for on-chain execution.

The explicit functionality and detailed architecture of the Neon proxy and emulation layer include:

- **Transaction preprocessing and validation**:
  - Incoming Ethereum-compatible transactions (signed raw transactions) are initially parsed by the Neon proxy, which validates parameters, formatting, signature, and adherence to Neon EVM semantics.

  - Preprocessing involves explicitly identifying the Solana accounts required for the transaction's execution, including those needed for composability calls, based on the target contract and method signature.

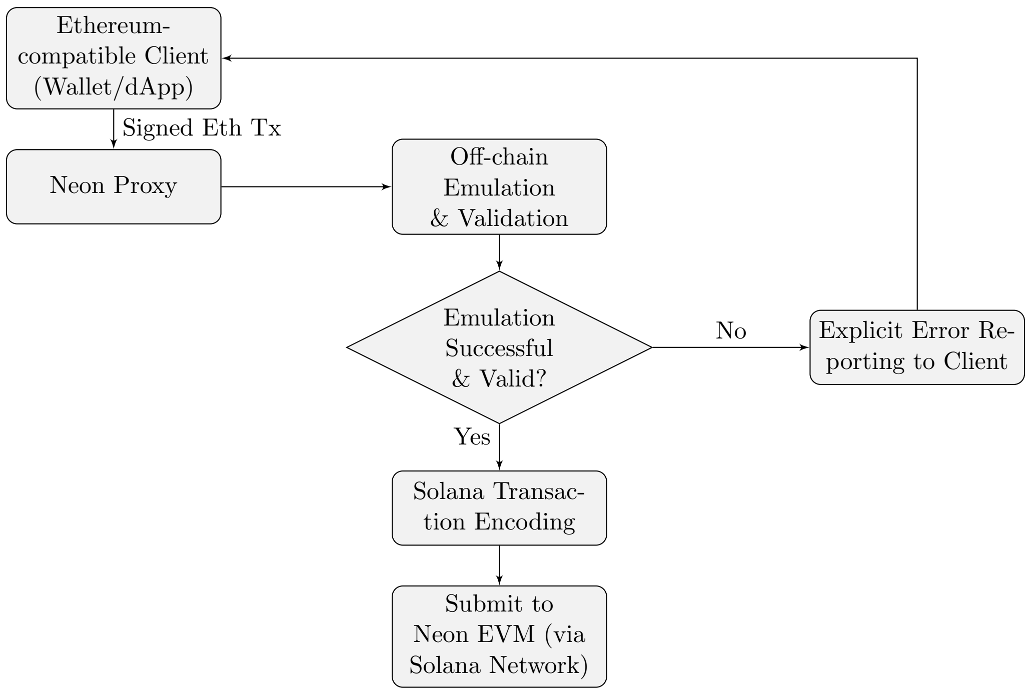
- **Off-chain execution emulation**:
  - The Neon proxy layer performs an off-chain simulation (emulation) of the transaction execution using a local instance or model of the Neon EVM logic. This critical step explicitly calculates the compute-unit (CU) consumption, predicts potential state changes, identifies necessary account creations (and associated SOL costs), and detects likely revert conditions *before* submitting the transaction on-chain.
  - Emulation enables preemptive identification of failures due to incorrect instructions, missing accounts, resource constraint violations (CU limit, transaction size), or insufficient user balance to cover gas, reducing transaction failures and unnecessary on-chain load.

- **Compute-unit budgeting and estimation**:
  - Accurate estimation of computational resources (CUs) required for on-chain execution occurs explicitly during the emulation step. The proxy calculates the expected CU usage for the EVM execution and any embedded Solana instructions from composability calls, ensuring the total remains within Solana's per-transaction budget (1.4 million CU).
  - Transactions projected to exceed computational budgets or data limits are explicitly identified during emulation. The Neon proxy typically rejects these transactions prior to on-chain submission, providing informative errors back to the client.

- **Final transaction encoding and submission**:
  - Upon successful emulation and validation, the Neon proxy explicitly encodes the original Ethereum transaction into a Solana transaction format targeting the Neon EVM program. This involves packaging the EVM bytecode execution instructions, necessary Solana account metadata (including PDAs, ATAs, payer accounts, program addresses), signer authorizations, and the estimated computational resource request.
  - The proxy submits this fully validated, formatted Solana transaction to the Solana network for execution by the Neon EVM program, increasing the likelihood of successful and predictable execution.

- **Error handling and revert detection**:
  - Transaction emulation explicitly identifies potential errors such as insufficient funds for gas, invalid nonces, logical instruction failures (detectable off-chain), account initialization failures (if dependencies are known), or computational budget overruns.
  - Transactions failing during emulation are typically not broadcast. Instead, the proxy returns explicit error messages (often compatible with Ethereum RPC error standards) and computational usage details to the client, facilitating debugging and transaction adjustment prior to resubmission.

The detailed architecture of the Neon proxy and emulation layer interaction is illustrated explicitly in Figure 3:



Ethereum-compatible Client (Wallet/dApp)
Signed Eth Tx
Neon Proxy
Off-chain Emulation & Validation
Emulation Successful & Valid?
No
Explicit Error Reporting to Client
Yes
Solana Transaction Encoding
Submit to Neon EVM (via Solana Network)

Figure 3: Detailed architecture of the Neon proxy and off-chain emulation layer

5.7 Gas and compute budget

Neon EVM composability transactions rely on a precise internal gas and compute-unit (CU) budgeting mechanism to manage computational resources effectively within the constraints of Solana's execution environment. Solana blockchain transactions have a fixed computational limit of approximately 1.4 million compute units per transaction block-wide (though individual transactions may request less). Neon EVM explicitly manages its portion of this resource through a defined budgeting model, ensuring deterministic execution outcomes relative to the requested budget.

Detailed considerations and explicit mechanisms for gas and compute-unit budgeting include:

- **Compute-unit estimation and accounting**:
  - The Neon proxy's off-chain emulation explicitly calculates the computational cost (in CUs) of composability transactions prior to on-chain submission. This includes estimating the CUs for EVM opcode execution and the CUs required for each Solana instruction invoked via precompiles.
  - The estimation process accounts for factors like computational complexity, instruction data size, number of accounts accessed, and potential dynamic state evaluations occurring during execution.
- **Gas to compute-unit relationship**:
  - Ethereum-style gas specified by the user in the transaction (`gasLimit`) serves as the primary input for budgeting. Neon EVM uses this gas limit, combined with the current gas price, to determine the maximum NEON fee the user is willing to pay.
  - Internally, Neon EVM translates the complexity of the requested operations (derived from the gas limit and emulation) into a required Solana compute-unit budget. There isn't a fixed public gas-to-CU conversion ratio; rather, the emulation determines the

  necessary CU budget for the specific transaction, which must be affordable within the NEON fee cap and below Solana's hard limits.
  - This ensures predictable resource allocation from the user's perspective (gas limit) while adhering to Solana's compute-unit model.

- **Dynamic execution resource management**:
  - During on-chain execution, the Solana runtime explicitly monitors compute-unit consumption against the budget requested by the Neon EVM transaction.
  - Neon EVM ensures its internal operations, including EVM execution and composability calls, stay within the requested budget. Transactions dynamically adapting to intermediate states (e.g., conditional logic, calls dependent on prior results) have their resource usage accounted for in real-time by the Solana runtime.
- **Atomicity and budget enforcement**:
  - Transactions explicitly exceeding the requested Solana compute-unit budget during on-chain execution trigger an immediate, atomic revert of the entire transaction by the Solana runtime.
  - This atomic enforcement mechanism maintains consistent and secure state transitions, ensuring transactions either fully complete within the computational budget or revert predictably without partial state changes.
- **Resource budget reporting and error handling**:
  - Transactions failing compute-unit budget constraints during off-chain emulation explicitly return detailed error messages (e.g., "out of gas" or equivalent) to the developer/client via the Neon Proxy.
  - If a transaction passes emulation but unexpectedly hits the CU limit on-chain (perhaps due to state changes between emulation and execution), the transaction simply reverts atomically on Solana, and the failure status is reflected back through standard Ethereum transaction receipt mechanisms (e.g., status 0).
  - Clear feedback from emulation helps developers adjust transaction complexity or increase the gas limit (and thus the potential NEON fee cap and requested CU budget) prior to resubmission.

5.8 Security and validation mechanisms

Neon EVM composability incorporates robust security and validation mechanisms designed to ensure transaction integrity, secure asset handling, isolation of execution contexts, and deterministic state management across Ethereum-compatible and Solana-native interactions. These measures protect against unauthorized access, ensure predictable behavior, and maintain the consistency of the Neon EVM environment.

The explicitly implemented security and validation components within Neon EVM include:

- **Transaction validation and pre-execution checks**:
  - The Neon Proxy performs rigorous off-chain transaction validation, explicitly verifying Ethereum transaction formatting, signature correctness (recovering the sender's address), nonce sequencing, and basic checks like sufficient balance for gas costs.

  - Pre-execution emulation explicitly validates instruction correctness (e.g., valid precompile calls, sensible parameters), Solana account existence/derivability, and compliance with computational resource constraints, preventing obviously invalid transactions from reaching the chain.

- **Operator account isolation and permission enforcement**:
  - Neon EVM infrastructure (including operators managing SOL for payer accounts) is explicitly isolated from user accounts and contract execution contexts. Operator roles are limited to facilitating resource allocation (like SOL provisioning) based on validated requests originating from user transactions.
  - Permissions within the Neon EVM program itself ensure that only authorized actions (e.g., executing EVM bytecode, interacting with precompiles, managing PDAs derived from specific seeds) can occur, preventing arbitrary state manipulation.

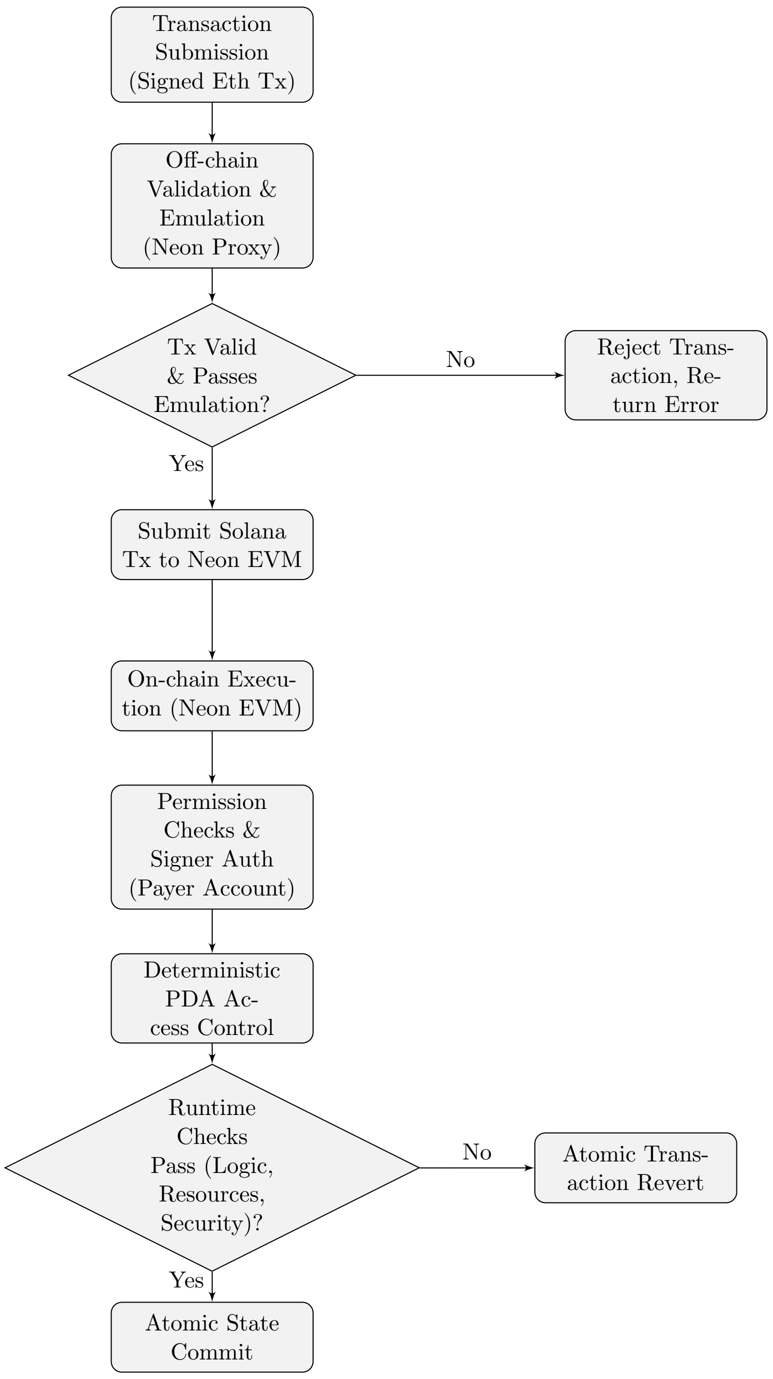
- **Secure signer authorization via Payer Accounts**:
  - Neon EVM explicitly utilizes deterministically derived, program-controlled payer accounts to sign specific Solana instructions requiring authorization on behalf of the original `msg.sender`.
  - Since these payer accounts have no externally held private keys and are controlled solely by the Neon EVM program logic triggered by a user's transaction, this provides secure signing capabilities without exposing user keys or requiring complex intermediate signing steps.

- **Atomic transaction enforcement**:
  - Neon EVM explicitly leverages Solana's native atomic transaction model. All operations within a single Neon EVM transaction (including EVM execution and all composability calls) either succeed together or fail together, reverting all state changes atomically.
  - This eliminates risks associated with partial execution states and ensures data consistency across the Ethereum (emulated) and Solana layers.

- **Deterministic PDA account management**:
  - Program-derived addresses (PDAs) used for storing EVM state (contract code, storage, nonces) and managing SPL token balances (via ERC20ForSPL) are deterministically generated based on user/contract addresses and predefined seeds.
  - These PDAs are exclusively controlled ('owned') by the Neon EVM program itself, preventing unauthorized external modification or access and ensuring predictable, secure state management tied directly to the corresponding EVM entity.

- **Error handling and state revert logic**:
  - Neon EVM explicitly implements robust error handling. Upon detecting transaction errors (invalid opcode, precompile failure, resource limit violation, assertion failure), permission issues, or security policy breaches during on-chain execution, Neon EVM ensures the Solana runtime triggers an atomic transaction reversion.
  - This securely rolls back all state changes attempted within that transaction, preserving the integrity of the blockchain state.

The explicit security and validation workflow employed by Neon EVM composability is illustrated clearly in Figure 4, highlighting checks at different stages.



Transaction Submission (Signed Eth Tx)
Off-chain Validation & Emulation (Neon Proxy)
Tx Valid & Passes Emulation?
No
Reject Transaction, Return Error
Yes
Submit Solana Tx to Neon EVM
On-chain Execution (Neon EVM)
Permission Checks & Signer Auth (Payer Account)
Deterministic PDA Access Control
Runtime Checks Pass (Logic, Resources, Security)?
No
Atomic Transaction Revert
Yes
Atomic State Commit

Figure 4: Explicit Neon EVM security validation and execution isolation workflow

6 Conclusion

Composability represents a significant advancement in blockchain interoperability, providing a unified framework for interaction between Ethereum-compatible smart contracts and native Solana programs. By abstracting the complexities of cross-chain communication—including disparate account models, token standards, and execution semantics—Neon EVM empowers developers to leverage the strengths of both ecosystems: Ethereum's mature development environment and tooling, combined with Solana's high throughput and low-cost execution.

This white paper has detailed the architecture, execution model, and core technical components underpinning this functionality. Key features such as the use of specialized precompiles, the ERC20ForSPL interface for token bridging, deterministic PDAs for secure state management, and the innovative payer account system for SOL provisioning and signing, collectively enable robust and efficient cross-chain operations. Furthermore, the emphasis on atomic transactions and comprehensive security validation ensures the integrity and reliability of interactions facilitated by Neon EVM composability.

The mechanisms described herein offer developers a complete toolset for building complex, cross-chain decentralized applications. As the blockchain landscape continues to evolve, solutions like Neon EVM composability that bridge ecosystems and enhance developer capabilities will be crucial in driving innovation and adoption across the Web3 space.