If you ChatGPT facts about Ethereum, you learn that the Ethereum network is open-source, you learn that it's decentralized, and it can be thought of as a world computer. If you dig a little deeper, you learn that “Ethereum's current transactions per second (TPS) is 15.6, with a maximum TPS of 62.34.” With an increasing user base year after year, becoming more mainstream, and being more accepted by our leaders, it begs the question how does this decentralized computer work? How is not crashing? How is it never going down, even companies as big as Google, have been known to drop the ball, what makes Ethereum so special. The answer in short is testing. Testing is usually an engineer's least favorite part of the lifecycle development, but it tends to be the most important. Ethereum has three types of testing, Execution Layer testing, Consensus Layer testing, and Cross-Layer testing.

All three are incredibly important to make sure that Ethereum grows properly, remains secure, and allows for seamless user interaction.

This essay will focus on the Execution Layer testing. The main test behind the Execution Layer test is the EVM test. While simple to set up, the EVM test is designed to test whether an Ethereum client adheres to the specifications set by Ethereum. It is checking to see that all clients produce the same output when given the same input, the input being a single transaction or even multiple transactions. If the clients are giving out different outputs then that means there is something wrong with the consensus layer or there is a fork in the chain.

The important characteristics of this test are the pre-state, environment, transaction, and post-state. The pre-state is the storage of the blockchain, containing accounts with balances, nonces, and smart contracts. Testing the entire chain is too cumbersome and expensive, so generating the pre-state allows us to test a “slice” of the chain. The environment specifies our basic requirements, such as a timestamp, previous block hashes, and block number. It also sets the total gas limit, base fee, and hard-fork activation times. The total gas limit sets a “stop-limit” on test execution, with a higher gas limit allowing for more extensive testing of larger smart contracts. The base fee ensures that every client calculates the correct transaction fee. Hard-fork activation times let us test future forks, ensuring the EVM handles new features or transactions correctly. By setting the fork activation time into the future, we can verify that transactions are either correctly rejected or accepted based on the fork status. The transaction is the message to the blockchain that acts on it. Without a transaction, nothing would be tested because nothing would change. Typically, the transaction involves sending code to observe its effects. The post-state is the final state we expect our slice of the chain to look like, verifying the test we wrote. Without a good post-state, our test can be useless.

Once we have our test set-up ready to go, we need to run them. This process is called test filling, and it is the process of compiling a test source code into a fixture that can be consumed by an execution client. In other words, converting the pre-state to the post-state. What separates this process from a unit test, is that the test fixture is consumable by all the clients. Allowing our test fixture to be consumed by all clients, lets us also check the consensus mechanism of the chain, by looking for discrepancies in the post-state.

Using the above setup, we can create different types of tests, such as state testing, fuzzy differential state testing, blockchain testing, and blockchain negative testing. The state testing is a straight one, you start with your pre-state and combine it with a single transaction. After combining the pre-state with the transaction, you get your output post-state. Your output post-state should show the modified balance, new/modified code, and the modified storage. It should also line up with your expected post-state, before running the test. A different flavor of state testing is fuzzy differential state testing. The main difference between the two is that with fuzzy, rather than having a known smart contract that was designed by you, a tool called fuzzy EVM is used. Fuzzy EVM alters the known smart contract code with some semi-randomness. This randomness can be used to test unexpected or unthought-of edge cases. Sending in your transaction with the pre-state and fuzzy smart contract will generate an output post-state. However, since we introduced randomness, rather than directly looking at the output post-state, we look at the memory verification and gas consumption. We also compare the output post-state of one Ethereum client against the others. If we see any discrepancies, then that is a sign, we have a bug or a mistake was made.

Our other tests, including blockchain testing and negative blockchain testing, also contain a pre-state. However, in these tests, the pre-state is initiated within the genesis block. The genesis block is then followed by other blocks to simulate a chain of length n, where n is the number of blocks created. We then pass these blocks to the execution client, which we expect to verify each client. Finally, we look at the chain head of the execution client, where we can expect to see our output-post state. Similar to our state-testing, if the output-post state does not match our expected post state, then we know there is an issue. The main difference between blockchain testing and negative blockchain testing is that with negative blockchain testing, we send an invalid block to the execution client. Since we send an invalid block, we expect to get a rejection from the execution client and by looking one block back, we can expect to see the correct post-state and chain head.

In summary, the testing of Ethereum is very important, and while today we only focused on one branch, we saw how rigorous and in-depth Ethereum testing can go. These tests can be created and run from two different repos, one written in Python and the other in C++.