Received: 6 August 2022 | Revised: 9 October 2022 | Accepted: 13 October 2022
DOI: 10.1002/qua.27035

REVIEW

International Journal of QUANTUM CHEMISTRY WILEY

Blockchain technology in quantum chemistry: A tutorial review for running simulations on a blockchain

Magnus W. D. Hanson-Heine[1] | Alexander P. Ashmore[2]

[1]School of Chemistry, University of Nottingham, Nottingham, UK
[2]Oxinet, Bath, UK

Correspondence
Magnus W. D. Hanson-Heine, School of Chemistry, University of Nottingham, University Park, Nottingham NG7 2RD, UK.
Email: magnus.hansonheine@nottingham.ac.uk, magnus.hansonheine@gmail.com

Abstract
Simulations of molecules have recently been performed directly on a blockchain virtual computer at atomic resolution. This tutorial review covers the current applications of blockchain technology for molecular modeling in physics, chemistry, and biology, and provides a step-by-step tutorial for computational scientists looking to use blockchain computers to simulate physical and scientific processes in general. Simulations of carbon monoxide have been carried out using molecular dynamics software on the Ethereum blockchain in order to facilitate the tutorial.

KEYWORDS
blockchain, computational science, distributed ledger, quantum chemistry

1 | INTRODUCTION

Blockchain and distributed ledger technology is widely used in areas ranging from finance [1], to medicine [2], energy markets [3], and transparent and censorship resistant computers [4]. Blockchains have been integrated into chemical databases for genomics [5–21], electron microscopy [22], and automated experimental chemistry [23], and these databases aim to improve both privacy and openness when storing, accessing, and sharing chemical and molecular data. Blockchains have been integrated with machine learning techniques for computational modeling in order to facilitate data sharing and collaborative or federated model construction [24–41]. Distributed ledgers have driven novel algorithm developments in quantum computing [42–52], blockchains have been used to create non-fungible tokens (NFTs) from scientific data [53], and blockchains have seen extensive use in the chemical industry [54–59].

However, the development of blockchains that are capable of performing generalized Turing complete computation now means that, in principle, the full range of computational science, quantum chemistry, and quantum physics simulations that are possible using conventional computers can also be implemented directly on blockchain computers. Quantum chemistry applications that use blockchain architectures to solve the Schrödinger equation have not yet been implemented; however, the implicit solutions for electronic structures that are contained within empirical parameterization of molecular mechanics and molecular dynamics force fields are available using existing software. Calculations of this type have been performed directly on a blockchain virtual computer to model the relatively simple physical process of a vibrating carbon monoxide molecule [60, 61]. The ability to perform computational and quantum science using blockchain computers is the focus of the current tutorial review.

2 | BLOCKCHAIN COMPUTERS—AN OVERVIEW

Blockchains are a series of algorithms that enable peer-to-peer consensus on the state and history of a decentralized ledger in an open network [1]. Open public blockchains aim to store a provably tamperproof record of data or computational operations. This is done through the use of cryptographic hashing functions that map variable length input data to a fixed-length output called a hash. Any change in the input results in an unpredictable but deterministic change to the hash, and consensus on the state of the blockchain is maintained using algorithms such as proof-of-work [1, 62]. Proof-of-work algorithmic details and the details of blockchain virtual computer operations have been discussed in detail elsewhere [1, 4]. However, in basic terms, a blockchain virtual computer could be described using the following protocol: a set of desired computations, termed “transactions,” are broadcast and executed by the computer nodes in the blockchain network including “miner” nodes. The outputs of these computations are hashed along with supplemental data by a miner node. The supplemental data includes an update to the state of the ledger that creates new tokens, or “coins,” that are under the miner node's control. The supplemental data is modified by a miner node until the numerical hash of the combined input is below a threshold (network difficulty), which is set in order to ensure that on average a certain amount of computational hashing work needs to be performed by a given miner in order for a valid combination of inputs to be found. Once a sufficiently small hash is found, the miner node that found the corresponding set of inputs broadcasts this new “block” of data to the rest of the network. Other nodes in the network then perform a much smaller number of hashing operations on these broadcast inputs in order to check that the hash is valid, and check that the other rules that the blockchain operates by have been followed. If the new block of data is accepted by the rest of the nodes in the network, then the miner receives control of the new tokens in addition to any fees that were sent to them as part of the computational work. This remunerates them for the costs associated with doing the hashing and broadcasting work. If the rules are not followed, then the new block is rejected by the other nodes, and the newly created tokens are not received in the copies of the blockchain held by the other nodes in the network. This creates an incentive for accurate and honest computational work, and disincentivizes fraudulent or erroneous work. Newly appended blocks of data contain a hash of the previous block as part of their supplemental data, which then causes revisions to old blocks of data to invalidate the hashes of the subsequent blocks, leading to them being rejected. The information in each “block” is therefore mathematically “chained” to the other blocks in the series through the sequential cryptographic hashes. All computer nodes in the network can therefore reliably maintain consensus on the state and history of the blockchain database, and on the computational work used to modify that state, creating a decentralized virtual computer. At the time of writing, the Ethereum blockchain and associated Ethereum virtual machine (EVM) is the oldest and most widely used blockchain that performs Turing complete general computation [4].

3 | BLOCKCHAIN MOLECULAR DYNAMICS

The first computational chemistry experiments performed entirely within a blockchain virtual computer have been reported in the scientific literature. These simulations involved modeling the vibrational motion of the carbon monoxide molecule on the Ethereum blockchain [60, 61]. In this case, atomic resolution molecular dynamics simulations were performed for the carbon monoxide molecule with a harmonic potential representing the energetics of the carbon–oxygen chemical bond over a 1 and 40 fs timeframe, respectively. Computer software compatible with the EVM was written in the specialized Solidity programming language, and was used to calculate a diatomic molecular dynamics trajectory with a variation

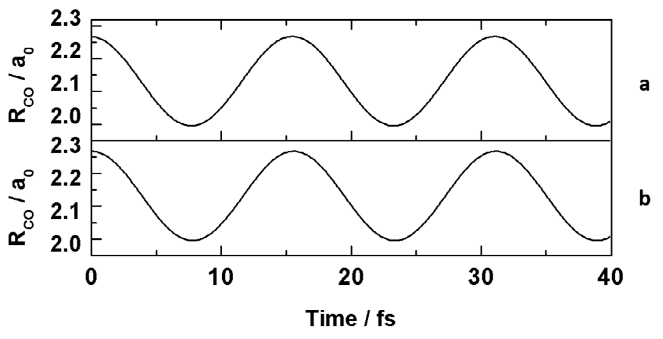


FIGURE 1 Molecular dynamics trajectories showing carbon–oxygen bond length variations for the carbon monoxide molecule when (A) coded in C# and run on a local computer, and (B) coded in solidity and run on the Ethereum blockchain. Figure reproduced from Hanson-Heine and Ashmore [60]

TABLE 1 The Ethereum blockchain transaction identification numbers (hexadecimal) for the simulations of carbon monoxide

Operation	Transaction identification number
Software upload	0x1c98dbb671dbe76b7cb4188d7585296e5adbe0f64fe8144546b4a82081089152
Preliminary molecular dynamics	0x13eb9c343f380334262706975c676aa5006ac2103b2132dfba0e1667c7dfddc6
Production molecular dynamics	0x36b510f3bdb2fa67a0d4f749899ab8630b54000ee5a211172d699d750f94f94f

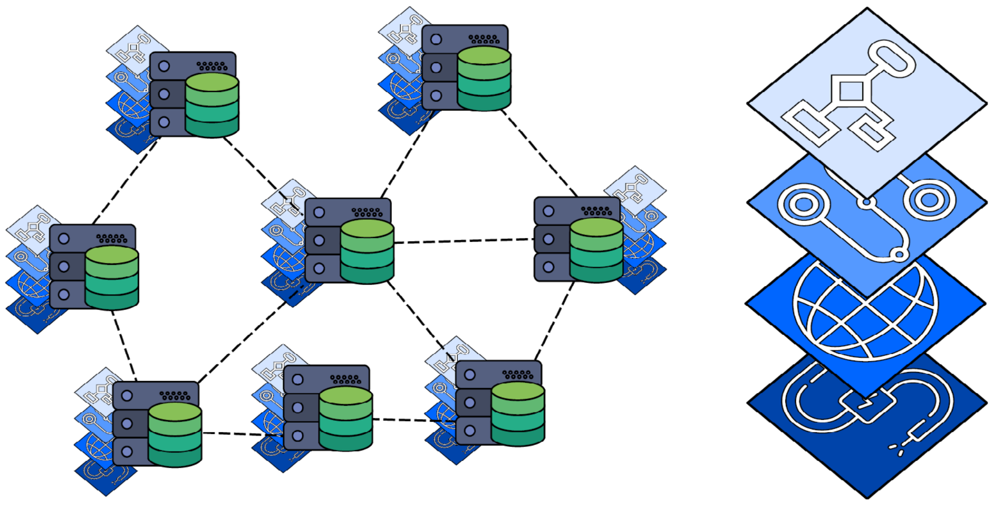


FIGURE 2 Abstract representation of computational state replication in blockchain virtual computers. Image credit: Giovanni Ciatto, University of Bologna

of the velocity Verlet algorithm by integrating Newton's equations of motion in atomic units [63]. These simulations were executed for 10 and 400 time steps of 0.1 fs each, respectively, with an initial carbon–oxygen bond length of 120 pm. This model used an equilibrium bond length parameter of 112.8 pm and a force constant of 1855 N/m, together with the masses of ^{12}C and ^{16}O assigned to the atoms. An equivalent simulation was also written using the C# programming language, and was executed on a local machine for comparison (see Figure 1). The outputs of these calculations were recorded on the Ethereum blockchain in blocks 9360161 and 9360178, respectively, and are available for the public to review. These specific transactions can be identified in the listed blocks using the hexadecimal identification numbers in Table 1.

4 | SMART CONTRACTS

Ethereum, and many other blockchains, allow for software to be deployed and executed through transactions on the blockchain. These pieces of software are known as "smart contracts," and since they are deployed as part of blocks, any contract added to the blockchain cannot be changed and any transactions interacting with the contract are also recorded permanently. This means that any deployed contract is always accessible regardless of the source code being lost, made closed source, or any hardware changes that might otherwise limit the accessibility.

Interactions with the blockchain are broadcast as messages by the nodes that form part of the blockchain network and are packaged into blocks by miner nodes as part of the proof-of-work algorithm. These messages can include instructions to deploy or execute smart contracts, and are termed "transactions." The term transaction is used in blockchain nomenclature as a holdover from the first blockchain, Bitcoin, which was initially used exclusively for financial payments, and this term is still used today when referring to the corresponding operations on blockchain networks that are capable of Turing complete computation. For example, the molecular dynamics simulation described here was carried out by executing a smart contract uploaded in Ethereum block 9360156 with the transaction identification number shown in Table 1.

5 | BLOCKCHAIN TIMESTAMPS

When similar scientific discoveries are made independently, the original discovery is often attributed to the person considered to have made the relevant observations or calculations first. One of the more famous examples of this is the controversy between Leibniz and Newton over who invented calculus; however, this can be a regular occurrence when researchers share a common knowledge base and goal set. Knowing the order in which experiments were carried out is also useful when analyzing methods of hypothesis testing and conclusion formation that can differ depending on the order in which observations happen. An important property of many blockchains is therefore the creation of an internal chronology. The regularity and sequential way in which new blocks of data are added and immutably chained to the existing blocks cryptographically, means that the position which calculations have in the blockchain acts as an automatic timestamp that can be used to verify the order in which the calculations were performed. In the case of the two carbon monoxide simulations discussed previously, the 1 fs simulation took place chronologically prior to the 10 fs simulation. Under normal circumstances this statement would be hard to prove. However, in this case the two trajectories are recorded in Ethereum blocks 9360161 and 9360178, respectively, which creates an effective timestamp proving the order in which the

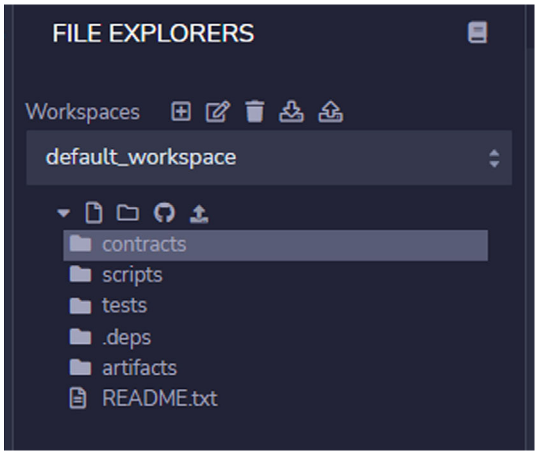
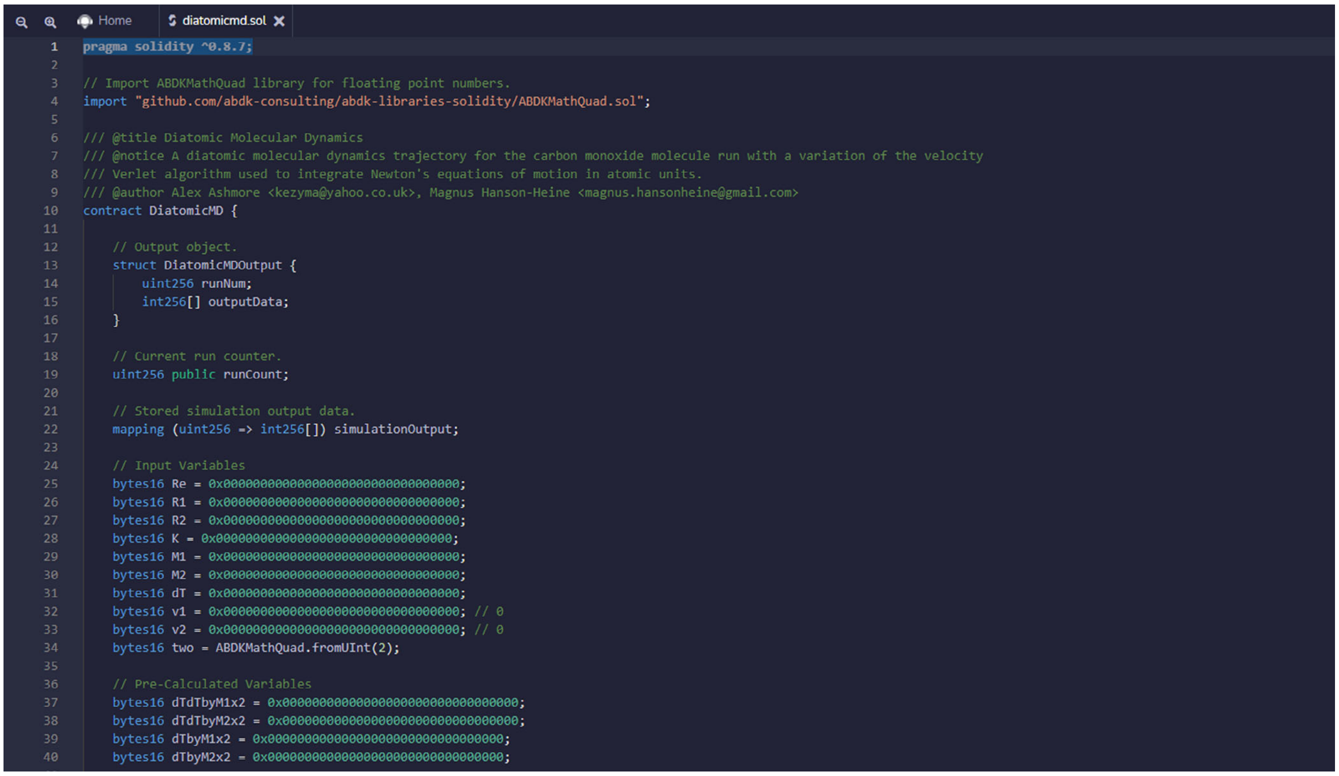


FIGURE 3 Screen capture showing the default workspace in the Remix IDE



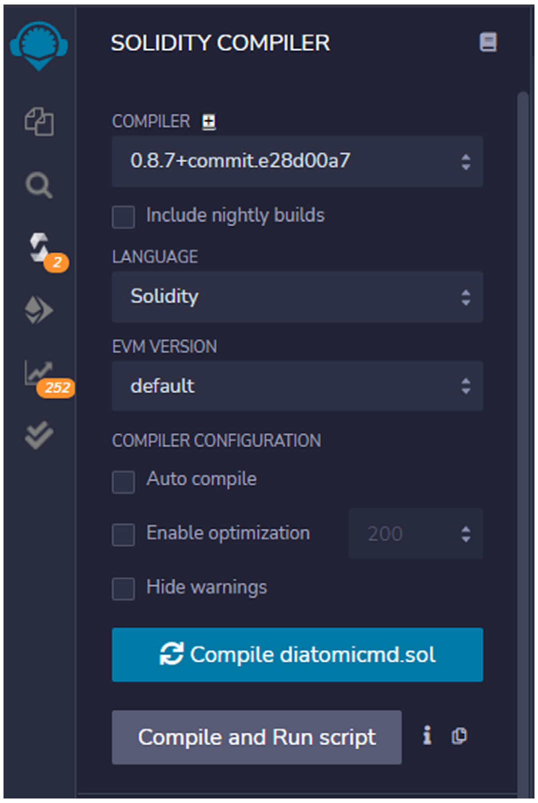
```
pragma solidity ^0.8.7;

// Import ABDKMathQuad library for floating point numbers.
import "github.com/abdk-consulting/abdk-libraries-solidity/ABDKMathQuad.sol";

/// @title Diatomic Molecular Dynamics
/// @notice A diatomic molecular dynamics trajectory for the carbon monoxide molecule run with a variation of the velocity
/// Verlet algorithm used to integrate Newton's equations of motion in atomic units.
/// @author Alex Ashmore <kezyma@yahoo.co.uk>, Magnus Hanson-Heine <magnus.hansonheine@gmail.com>
contract DiatomicMD {

    // Output object.
    struct DiatomicMDOutput {
        uint256 runNum;
        int256[] outputData;
    }

    // Current run counter.
    uint256 public runCount;

    // Stored simulation output data.
    mapping (uint256 => int256[]) simulationOutput;

    // Input Variables
    bytes16 Re = 0x00000000000000000000000000000000;
    bytes16 R1 = 0x00000000000000000000000000000000;
    bytes16 R2 = 0x00000000000000000000000000000000;
    bytes16 K = 0x00000000000000000000000000000000;
    bytes16 M1 = 0x00000000000000000000000000000000;
    bytes16 M2 = 0x00000000000000000000000000000000;
    bytes16 dT = 0x00000000000000000000000000000000;
    bytes16 v1 = 0x00000000000000000000000000000000; // 0
    bytes16 v2 = 0x00000000000000000000000000000000; // 0
    bytes16 two = ABDKMathQuad.fromUInt(2);

    // Pre-Calculated Variables
    bytes16 dTdTbyM1x2 = 0x00000000000000000000000000000000;
    bytes16 dTdTbyM2x2 = 0x00000000000000000000000000000000;
    bytes16 dTbyM1x2 = 0x00000000000000000000000000000000;
    bytes16 dTbyM2x2 = 0x00000000000000000000000000000000;
```

FIGURE 4 Screen capture showing the provided contract in the text editor of the Remix IDE

simulations were performed for as long as the Ethereum blockchain continues to operate with a coherent state and history. Similar uses of this time stamp data have been proposed in the blockchain integrated automatic experiment platform (BiaeP) protocol [23].

6 | SCIENTIFIC REPRODUCTION

Scientific reproduction, and the replication of findings, including computational work, is of critical importance for error checking and establishing consensus among scientists. The internal state of blockchain computers are reproduced by multiple independent local computer nodes (Figure 2). This makes them an ideal target for running scientific simulations in an openly verifiable and repeatable manner which can enable improved peer-review and replication in the computational sciences. Any simulation written and deployed to the blockchain can always be found in the same place on that blockchain, and the simulation can be verified by anyone running a node, and can be re-executed by anyone using the blockchain



SOLIDITY COMPILER

COMPILER: 0.8.7+commit.e28d00a7

Include nightly builds

LANGUAGE: Solidity

EVM VERSION: default

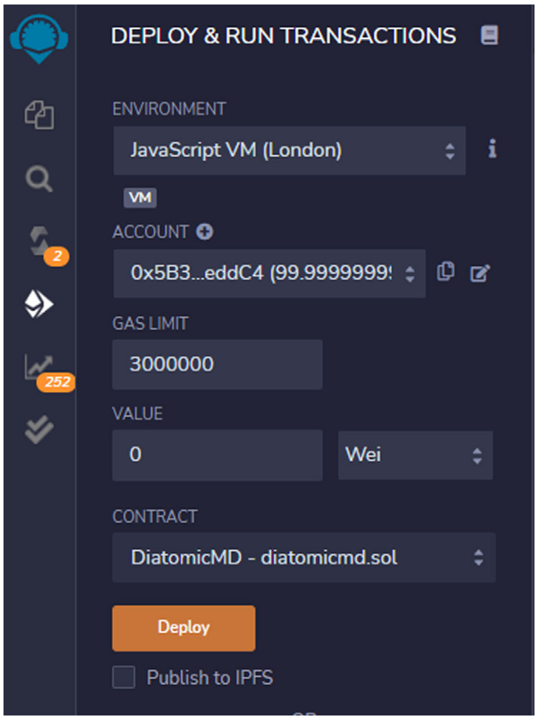
COMPILER CONFIGURATION

Auto compile

Enable optimization 200

Hide warnings

FIGURE 5 Screen capture showing the Remix IDE Solidity compiler tab



DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: JavaScript VM (London)

VM

ACCOUNT: 0x5B3...eddC4 (99.9999999!

GAS LIMIT: 3000000

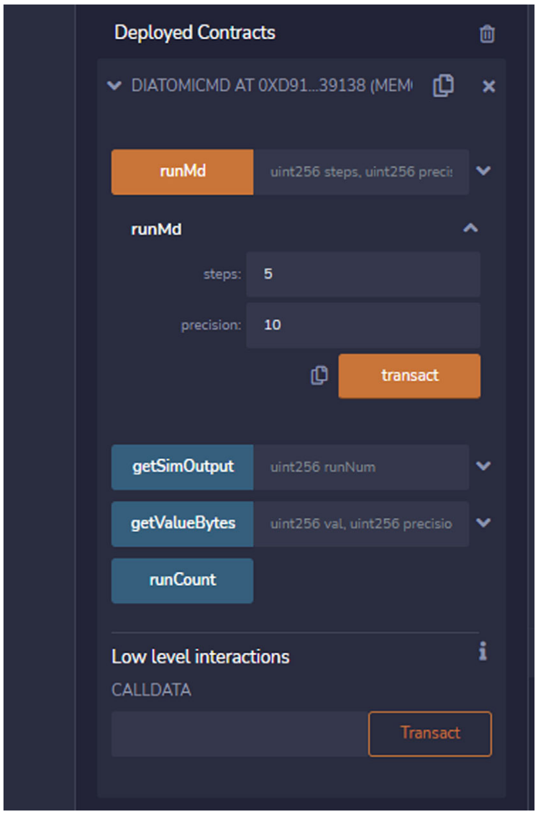
VALUE: 0 Wei

CONTRACT: DiatomicMD - diatomicmd.sol

Publish to IPFS

FIGURE 6 Screen capture showing the Remix IDE "Deploy and Run Transactions" tab

and willing to pay the transaction fees. Previous executions of smart contracts can also often be identified, and all of the parameters used to execute them can be easily verified and retrieved, ensuring that any simulation that was run can also be repeated with the computational certainty that executing the same code, with the same parameters, will result in the same output.


Deployed Contracts

DIATOMICMD AT 0XD91...39138 (MEM

runMd uint256 steps, uint256 precis

runMd

steps: 5

precision: 10

transact

getSimOutput uint256 runNum

getValueBytes uint256 val, uint256 precisio

runCount

Low level interactions

CALLDATA

Transact

FIGURE 7 Screen capture of the manual interface for the DiatomicMD contract in Remix IDE

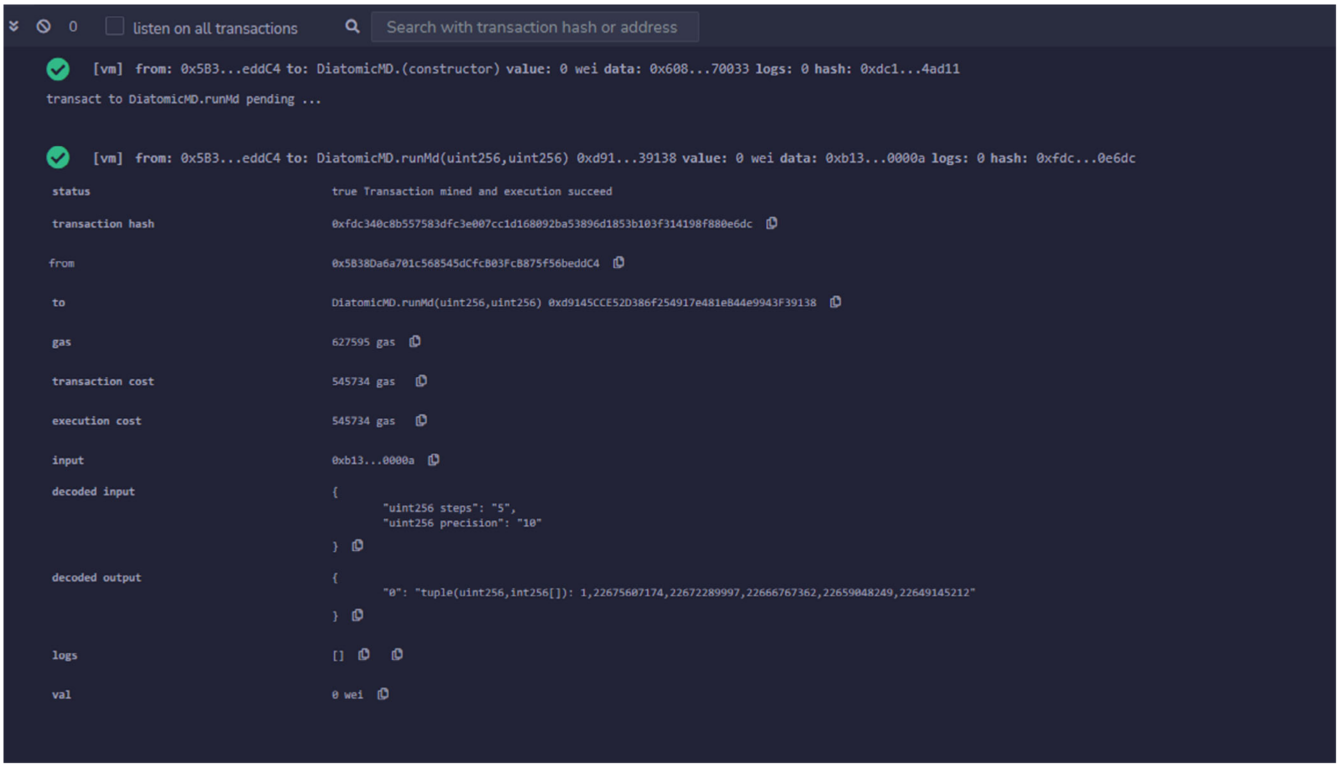
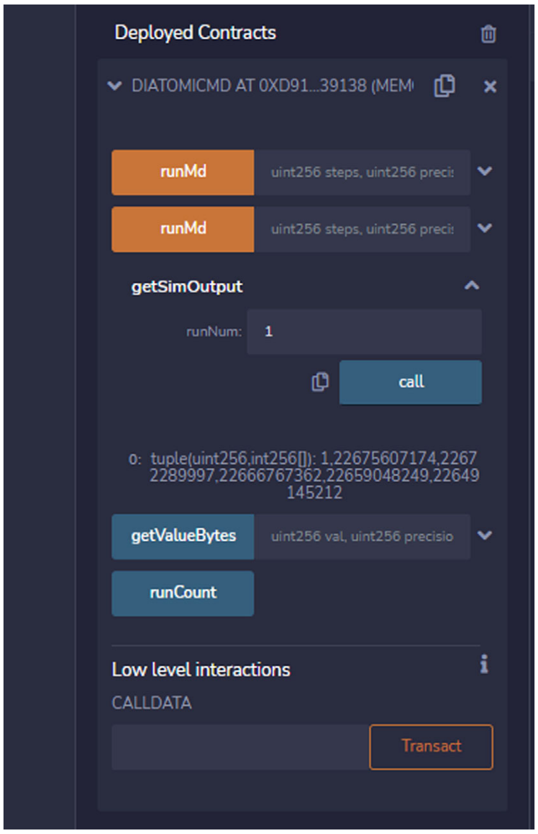

```
0   listen on all transactions   Search with transaction hash or address

[vm] from: 0x5B3...eddC4 to: DiatomicMD.(constructor) value: 0 wei data: 0x608...70033 logs: 0 hash: 0xdc1...4ad11
transact to DiatomicMD.runMd pending ...

[vm] from: 0x5B3...eddC4 to: DiatomicMD.runMd(uint256,uint256) 0xd91...39138 value: 0 wei data: 0xb13...0000a logs: 0 hash: 0xfdc...0e6dc
 status                       true Transaction mined and execution succeed
 transaction hash             0xfdc340c8b557583dfc3e007cc1d168092ba53896d1853b103f314198f880e6dc
 from                         0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
 to                           DiatomicMD.runMd(uint256,uint256) 0xd9145CCE52D386f254917e481eB44e9943F39138
 gas                          627595 gas
 transaction cost             545734 gas
 execution cost               545734 gas
 input                        0xb13...0000a
 decoded input                {
                                      "uint256 steps": "5",
                                      "uint256 precision": "10"
                              }
 decoded output               {
                                      "0": "tuple(uint256,int256[]): 1,22675607174,22672289997,22666767362,22659048249,22649145212"
                              }
 logs                         []
 val                          0 wei
```

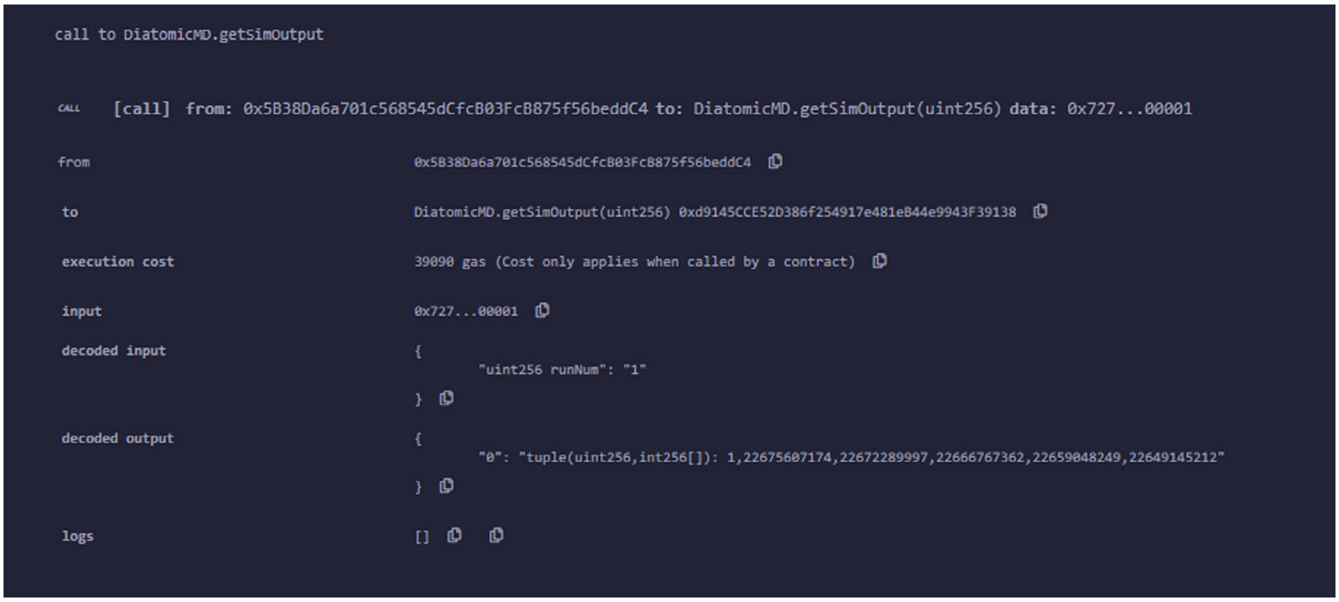
FIGURE 8 Screen capture of the Remix IDE transaction viewer, displaying the example simulation results

While this does not prevent a malicious contract, for example, one that would check the number of times it has been run and then produce a different result dependent on that number, the compiled bytecode of the contract can be retrieved from the blockchain and decompiled into code that can be manually checked for any errors or attempts at deception. This can be done regardless of whether the original source code is available, and the bytecode can also be used to confirm that the source code, if provided, is the same as the code that was deployed on the blockchain.


Deployed Contracts
DIATOMICMD AT 0XD91...39138 (MEM
runMd
uint256 steps, uint256 precis
runMd
uint256 steps, uint256 precis
getSimOutput
runNum: 1
call
getValueBytes
uint256 val, uint256 precisio
runCount
Low level interactions
CALLDATA
Transact

```
0: tuple(uint256,int256[]): 1,22675607174,22672289997,22666767362,22659048249,22649145212
```

FIGURE 9 Screen capture showing the manual interface for the DiatomicMD contract with the getSimOutput call expanded



```
call to DiatomicMD.getSimOutput

CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: DiatomicMD.getSimOutput(uint256) data: 0x727...00001

from                 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

to                   DiatomicMD.getSimOutput(uint256) 0xd9145CCE52D386f254917e481eB44e9943F39138

execution cost       39090 gas (Cost only applies when called by a contract)

input                0x727...00001

decoded input        {
                             "uint256 runNum": "1"
                     }

decoded output       {
                             "0": "tuple(uint256,int256[]): 1,22675607174,22672289997,22666767362,22659048249,22649145212"
                     }

logs                 []
```

FIGURE 10 Screen capture showing the transaction viewer displaying results retrieved with a call to getSimOutput

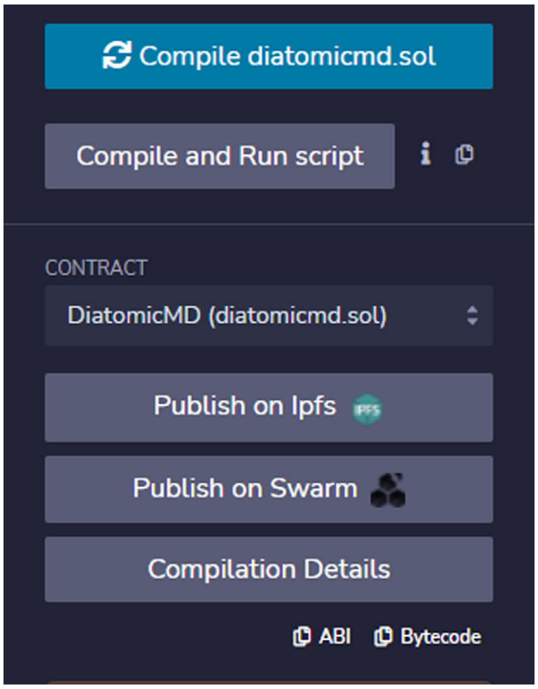
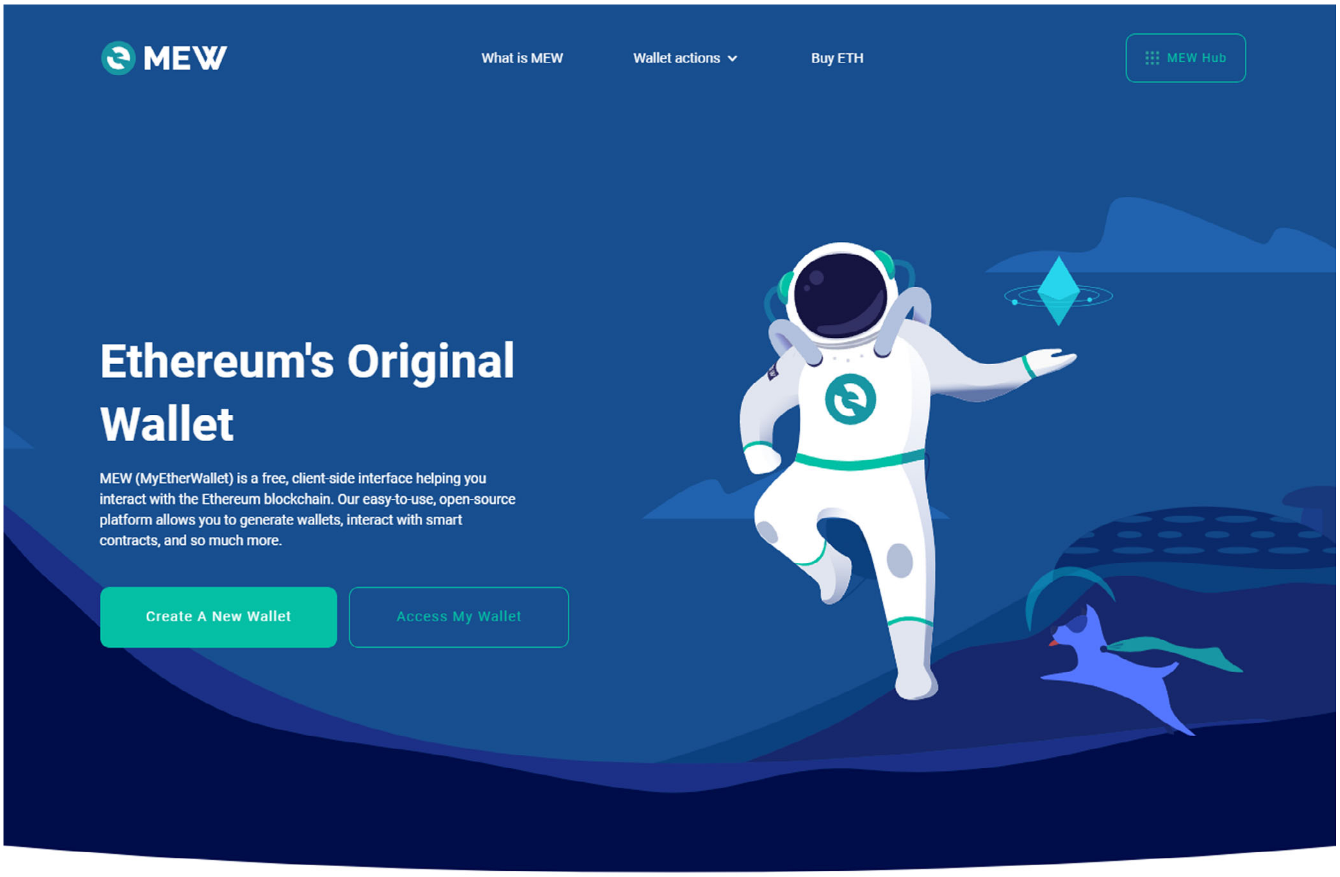


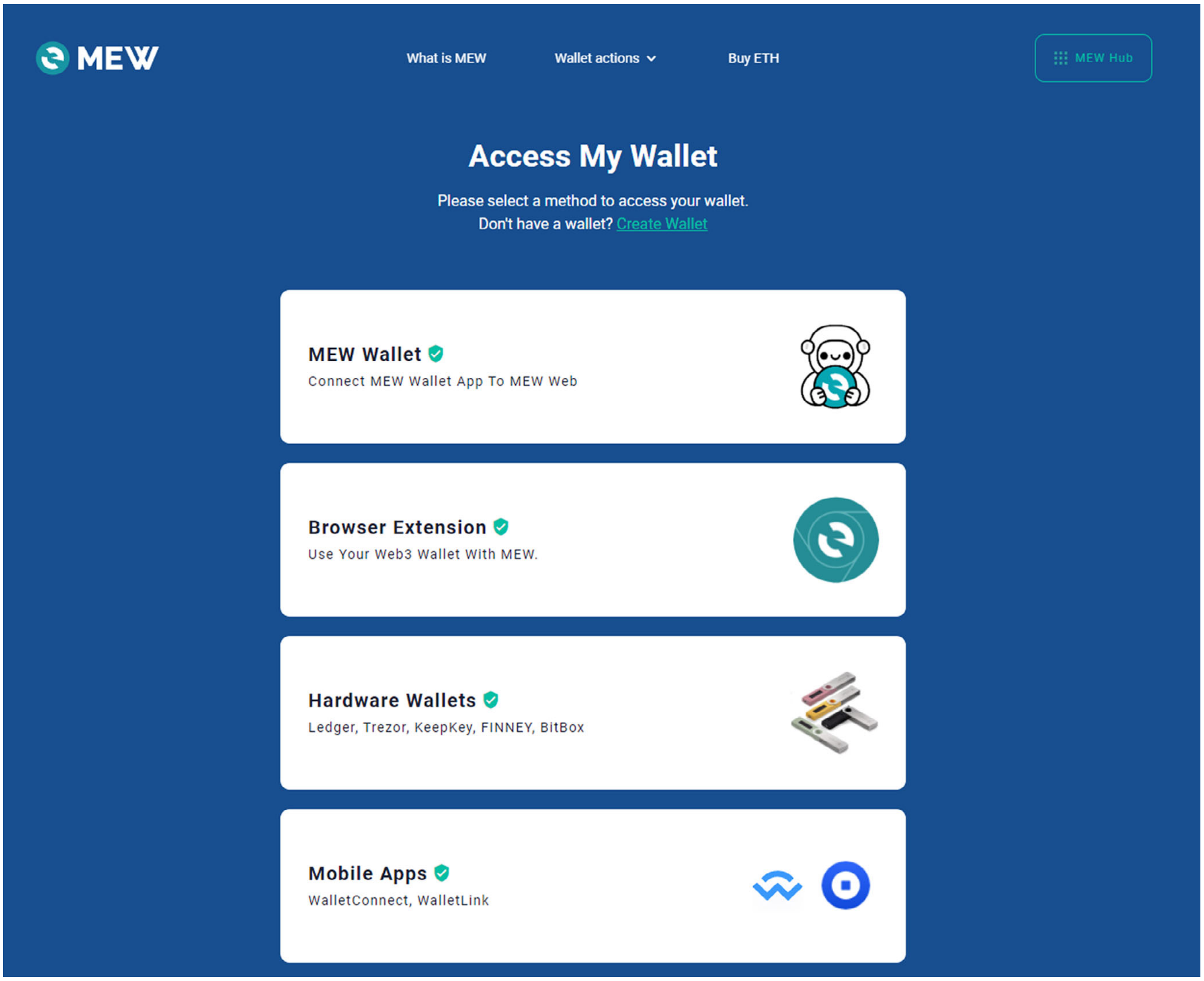
FIGURE 11 Screen capture of the "Deploy and Run Transactions" tab after compiling the contract



Ethereum's Original Wallet

MEW (MyEtherWallet) is a free, client-side interface helping you interact with the Ethereum blockchain. Our easy-to-use, open-source platform allows you to generate wallets, interact with smart contracts, and so much more.

FIGURE 12 Screen capture of the MyEtherWallet website homepage



Access My Wallet

Please select a method to access your wallet.
Don't have a wallet? Create Wallet

MEW Wallet
Connect MEW Wallet App To MEW Web

Browser Extension
Use Your Web3 Wallet With MEW.

Hardware Wallets
Ledger, Trezor, KeepKey, FINNEY, BitBox

Mobile Apps
WalletConnect, WalletLink

FIGURE 13 Screen capture of the MyEtherWallet website wallet access menu

Smart contracts can make records and retrieve data from blocks, which can serve as an optional additional record of a simulation. While it is generally more computationally expensive to record data on the blockchain itself, if that data is to be accessed later multiple times, this could prove more cost effective, as well as making the simulation easier to verify, since, any output stored will have been confirmed and recorded in a way that makes tampering almost impossible when the blockchain is operating as intended.

7 | CENSORSHIP RESISTANT COMPUTATION

Calculations on open public blockchains can be made available to anyone for review. Previous blockchain executions can be identified, verified, and executed again using the same code to produce the same output, and new pieces of software can be uploaded and executed with pseudo-anonymity for the researchers involved. Once the transactions are broadcast and the correct fees are paid, a given computation cannot be stopped by any central authority so long as the blockchain operates normally. Blockchains can therefore be useful for scientists working under conditions where concerns about fraud and censorship become meaningful considerations. Blockchain consensus algorithms can create permanent and practically immutable or tamperproof records of computational data. While relatively rare, in extreme cases, state actors have been known to block or edit data contained in peer-reviewed journals [64, 65].

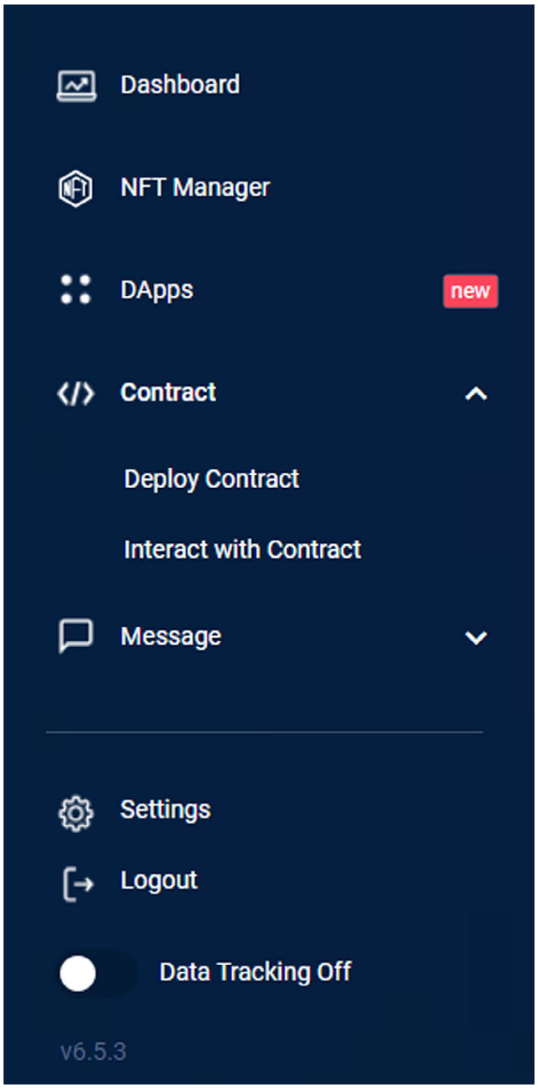
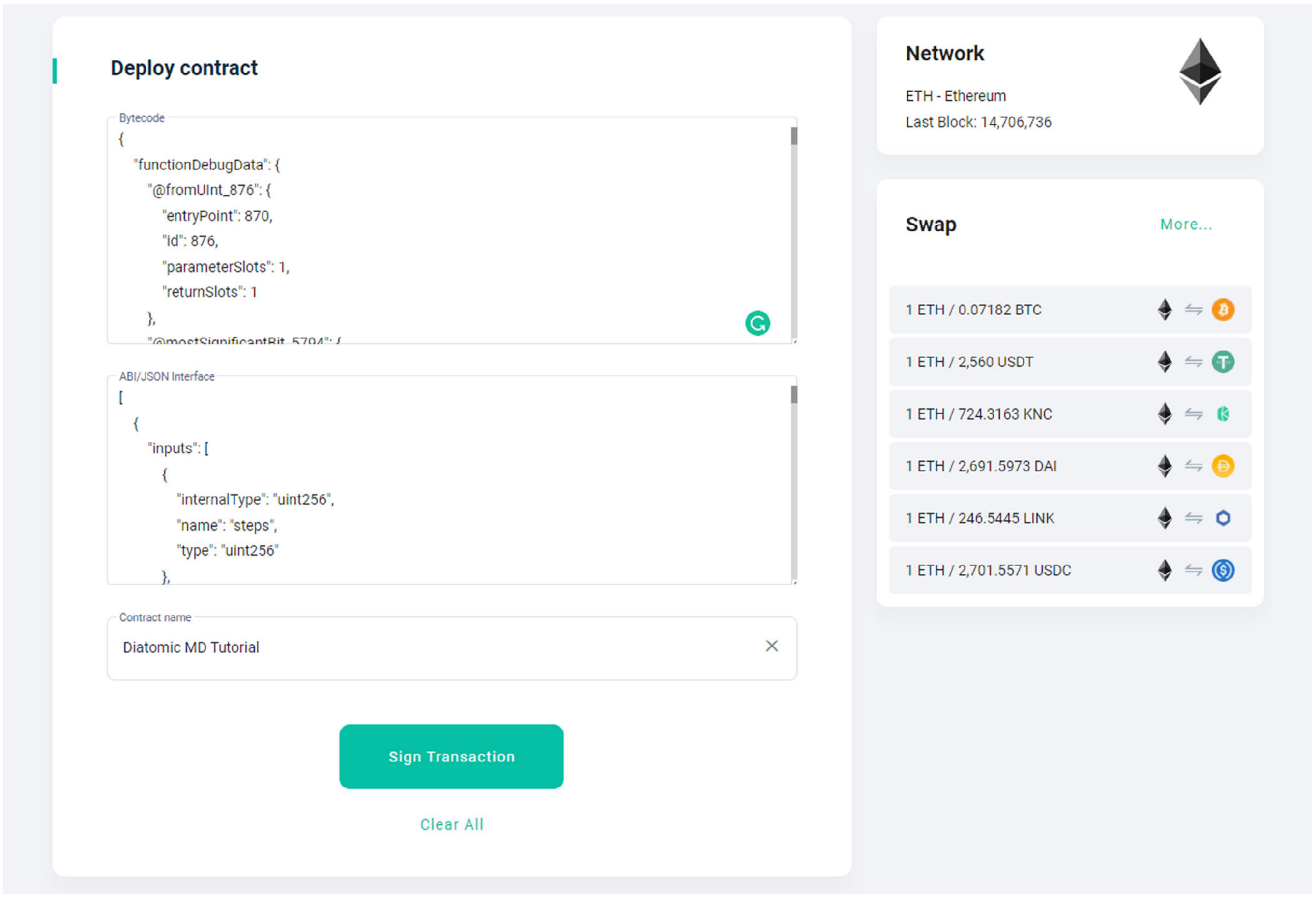


FIGURE 14 Screen capture showing the MyEtherWallet "Deploy Contract" menu option

8 | BLOCKCHAIN LIMITATIONS

The requirement that multiple computer nodes reproduce the computational work in each block in order to reach and maintain internal consensus currently makes the computational rate of blockchains significantly slower than conventional computers. More efficient architectures are currently under development; however, this remains a significant limitation. For example, the production molecular dynamics simulation of carbon monoxide previously mentioned in this tutorial review was included in Ethereum block 9360178, and this block was mined in ~11 s, compared to a ~135 ms execution time for an equivalent C# simulation executed on a standalone desktop machine running with a i7-4790k CPU and 32GB of 1333 MHz DDR3 RAM.

The relative inefficiency of blockchain computation compared with trusted and centralized computational work, and the electricity needed to perform the hashing operations in proof-of-work, has led to open questions over the ecological impact of blockchain computation [66]. Consensus algorithms such as proof-of-stake and even green-proof-of-work have been proposed as energy efficient alternatives [67]. The mining competition in proof-of-work means that miners to use energy to reach consensus even when they fail to find the correct hash before their competitors, which could be considered wasteful when compared with alternative methods. However, this process also incentivizes miners to use energy in an efficient manner in order to maximize their chance of being the first to generate a new block by using electricity in areas and at times when it is cheaper, and would otherwise be wasted or used less efficiently. This makes direct watt-for-watt comparisons between blockchain proof-of-work mining and other types of computational energy usage potentially misleading as a direct measure of their relative environmental impact. Blockchain computers are currently many orders of magnitude less efficient than centralized or trusted alternatives; however, they also provide certain computational advantages that these other systems cannot replicate.


Deploy contract

Bytecode
```
{
  "functionDebugData": {
    "@fromUInt_876": {
      "entryPoint": 870,
      "id": 876,
      "parameterSlots": 1,
      "returnSlots": 1
    },
    "@mostSignificantBit_5794": {
```

ABI/JSON Interface
```
[
  {
    "inputs": [
      {
        "internalType": "uint256",
        "name": "steps",
        "type": "uint256"
      },
```

Contract name: Diatomic MD Tutorial

Network

ETH - Ethereum

Last Block: 14,706,736

Swap

1 ETH / 0.07182 BTC

1 ETH / 2,560 USDT

1 ETH / 724.3163 KNC

1 ETH / 2,691.5973 DAI

1 ETH / 246.5445 LINK

1 ETH / 2,701.5571 USDC

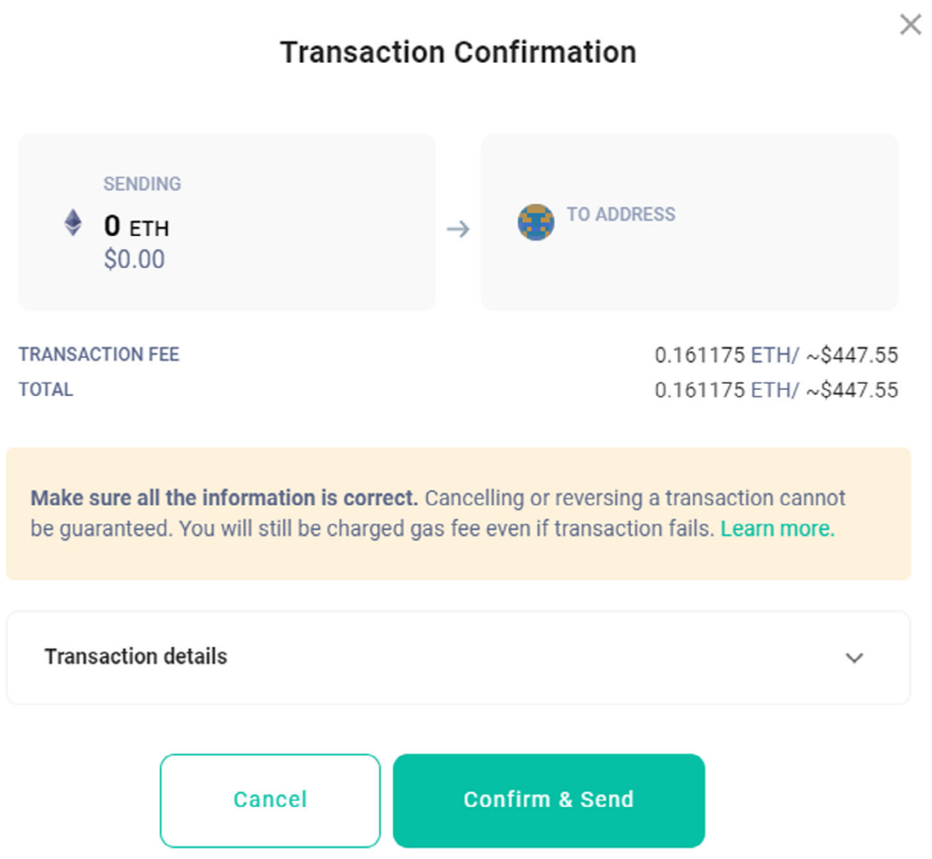
FIGURE 15 Screen capture of MyEtherWallet, showing the contract deployment section for the tutorial program

Ethereum also has a limit to the maximum amount of computation that can be performed as part of generating a single block. This is measured in units known as "gas," and exists in part to make sure that individual programs do not halt the blockchain and prevent it from regularly forming new blocks. A detailed description of the relationship between gas and specific computational operations can be found in the Ethereum yellow paper "Ethereum: A Secure Decentralized Generalized Transaction Ledger EIP-150 Revision" by Gavin Wood and currently hosted on the website gavwood.com/paper.pdf at the time of writing. The hard limit on the maximum amount of computation that can be performed as part of generating a single block in the case of the Ethereum blockchain is currently set to 30 000 000 gas per-block, with a target of 15 000 000. The example molecular dynamics transaction carried out in this work made use of 545 720 gas. However, if the entire block limit were used for a simulation it would then be stopped by the Ethereum network at that point even if instructions and additional funds were available for a longer run time. In principle both the results and the current state of a simulation after one transaction can be stored and used to send a second transaction in a future block in order to continue the simulation from where it left off, but this is not currently possible without action from outside the Ethereum blockchain in order to trigger the continuation of the simulation.

Furthermore, since each node needs to re-execute every transaction and reproduce the same results, the programming languages used for smart contracts are either entirely deterministic, or, in the cases where they use pre-existing languages, are limited to ensure that certain features are not used. As an example, this means that smart contracts cannot use random number generators that generate (pseudo)random numbers within a contract in a way which might lead to different computers generating different values, they also cannot contain any kind of variable that involves an estimation which can vary across different programming languages and hardware, and they cannot contain any calculation involving the current date, time, or location specific data that is determined by the local computer node at the point that the code is executed. It is possible to implement pseudo-random number generators and provide seed data as an input parameter to generate pseudo-randomness in a deterministic manner that allows for consensus on the output between nodes in the blockchain.

9 | BLOCKCHAIN SIMULATION TUTORIAL

The smart contract used in this tutorial is the source code and set of execution instructions for a piece of molecular dynamics software that has been designed to simulate the general vibration of diatomic molecules using a variation of the velocity Verlet algorithm [63] with a harmonic potential


Transaction Confirmation

SENDING
0 ETH
$0.00

TO ADDRESS

TRANSACTION FEE 0.161175 ETH/ ~$447.55
TOTAL 0.161175 ETH/ ~$447.55

Make sure all the information is correct. Cancelling or reversing a transaction cannot be guaranteed. You will still be charged gas fee even if transaction fails. Learn more.

FIGURE 16 Screen capture showing the MyEtherWallet smart contract deployment transaction confirmation

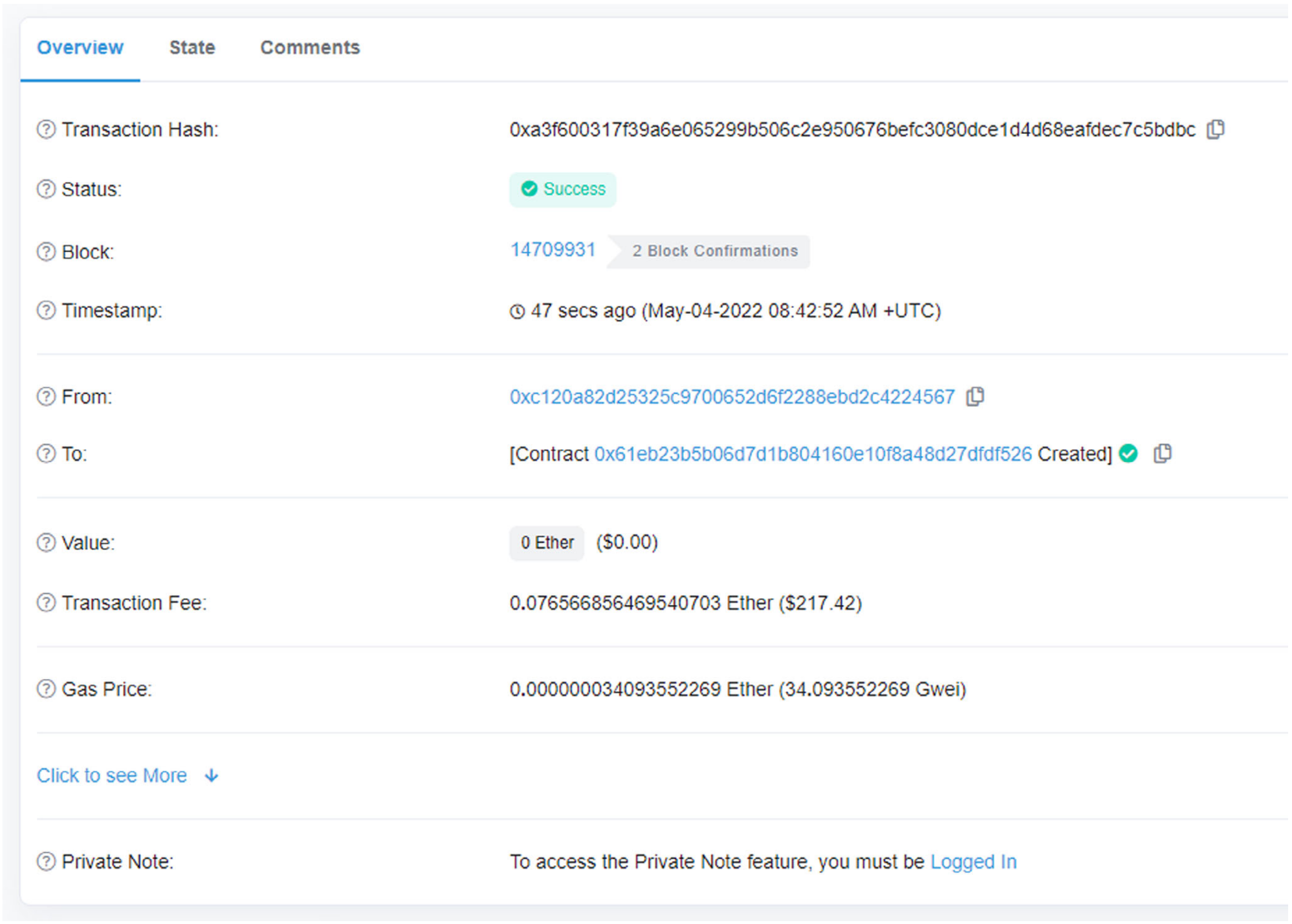
representing the energetics of chemical bonding. This contract is similar to the contract used to carry out the original simulations of the carbon monoxide molecule mentioned previously, but it has been written to allow for a greater number of user-defined input parameters in order to facilitate the general simulation of any diatomic molecule and to improve its usefulness as a potential teaching aid. The source code for this smart contract can be found in the Supporting Information and on the Ethereum blockchain (vide infra), and will be referenced explicitly here. This molecular dynamics software has two modes of execution. The first mode reproduces the original molecular dynamics of carbon monoxide using the parameters used during the simulations carried out in Ethereum blocks 9 360 161 and 9 360 178, with a user specified output precision and number of time steps. The second mode also allows for user defined input values to be set for the masses of the two atoms individually, the harmonic bond force constant, the time step size, the initial bond length, and the equilibrium bond length. The details of these functions can be found in the smart contract and for simplicity the first mode will be used for the demonstration in this tutorial. However, any custom built simulation software that is written in Solidity can also be substituted for the contract we have used here in order to run any other physically meaningful simulation or any other computational science application of the users choice.

10 | SUPPORTING SOFTWARE AND TOOLS

To follow this tutorial, some third-party software will be required. To simplify this process, minimize hardware requirements, and to reduce the chances of encountering version differences, web applications have been primarily chosen here. There are however, many different options for each of these categories and this is only one of many ways to deploy a smart contract.

10.1 | Ethereum wallet

A wallet is a piece of software (or hardware) that stores the private key of a cryptographic private and public key pair that is used to access and sign transactions for an account on a blockchain. Wallets will often contain further features, ranging from running



Transaction Hash: 0xa3f600317f39a6e065299b506c2e950676befc3080dce1d4d68eafdec7c5bdbc

Status: Success

Block: 14709931 2 Block Confirmations

Timestamp: 47 secs ago (May-04-2022 08:42:52 AM +UTC)

From: 0xc120a82d25325c9700652d6f2288ebd2c4224567

To: [Contract 0x61eb23b5b06d7d1b804160e10f8a48d27dfdf526 Created]

Value: 0 Ether ($0.00)

Transaction Fee: 0.076566856469540703 Ether ($217.42)

Gas Price: 0.000000034093552269 Ether (34.093552269 Gwei)

Click to see More

Private Note: To access the Private Note feature, you must be Logged In

FIGURE 17 Screen capture showing the smart contract deployment on the Ethereum blockchain using the Etherscan blockchain block explorer website

entire nodes or miner nodes, to lite-wallets that interact with third-party nodes in order to broadcast transactions and query the blockchain.

For this tutorial, a browser based lite-wallet was chosen. These are wallets that are integrated into a web browser and allow for more convenient access to web applications that utilize blockchains. While this tutorial is written with these wallets in mind, any wallet that is supported by MyEtherWallet (MEW) can be used here, which includes a variety of physical hardware wallets and mobile applications.

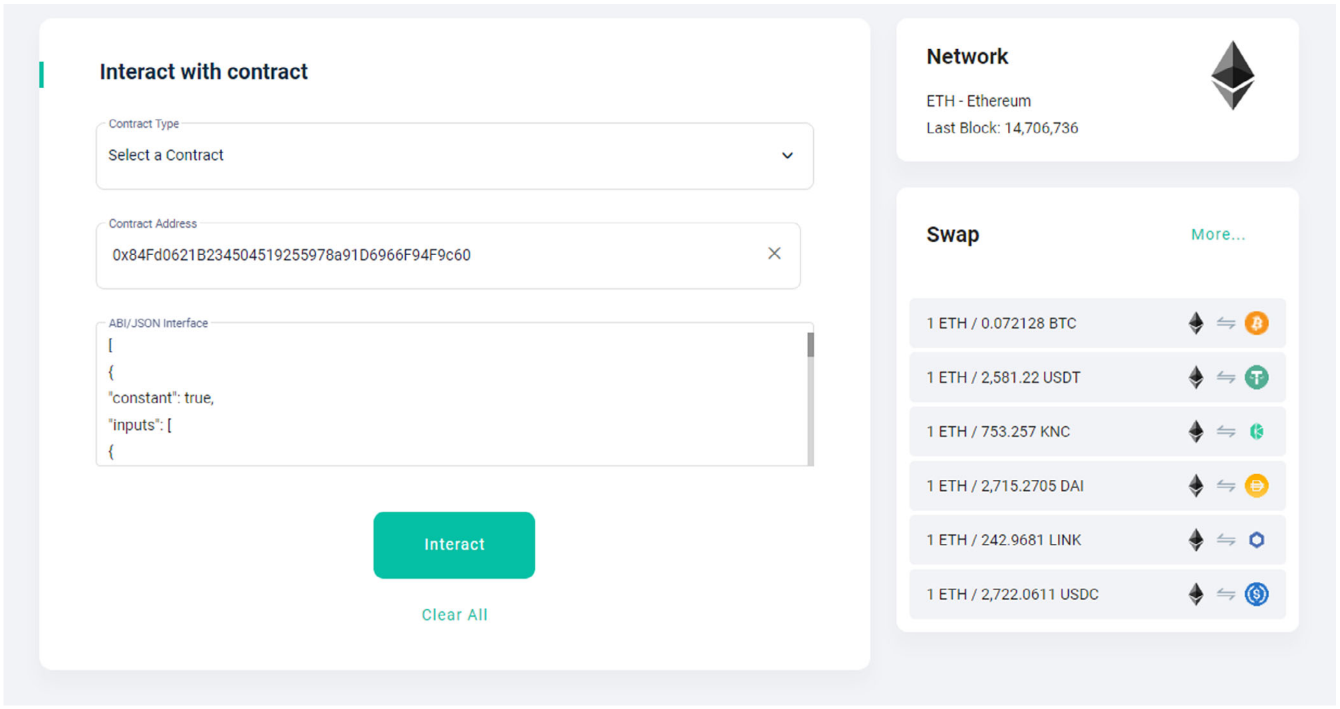
To complete the deployment and execution process, a sum of the Ether token (ETH), or the native token associated with the blockchain being used to compute the simulation, will be required in order to pay for any associated transaction fees (Appendix). This sum can vary greatly depending on the relative exchange value of the native token, and the congestion of the network.

Two notable Ethereum browser wallets that function very similarly are the "Brave Wallet" and "Metamask" software. The former is a wallet integrated with the Brave browser, while the latter is a plug-in that can be installed on most mainstream browsers such as Chrome and Firefox. Brave can be downloaded from brave.com and Metamask can be downloaded from *metamask.io*.

10.2 | Integrated development environment

An integrated development environment (IDE) is an application designed to fulfill many or all of the requirements of software development. This typically incorporates a text editor for working with source code, alongside other features such as file management, a code compiler, and/or debugging tools.

For this tutorial, an IDE for Solidity, the native programming language of Ethereum, is required. The Remix IDE is the one used here, and it has the capability to run smart contracts locally for debugging purposes. Remix can be accessed through the website remix.ethereum.org.



Interact with contract

Contract Type: Select a Contract

Contract Address: 0x84Fd0621B234504519255978a91D6966F94F9c60

ABI/JSON Interface:

```
[
{
"constant": true,
"inputs": [
{
```

Network

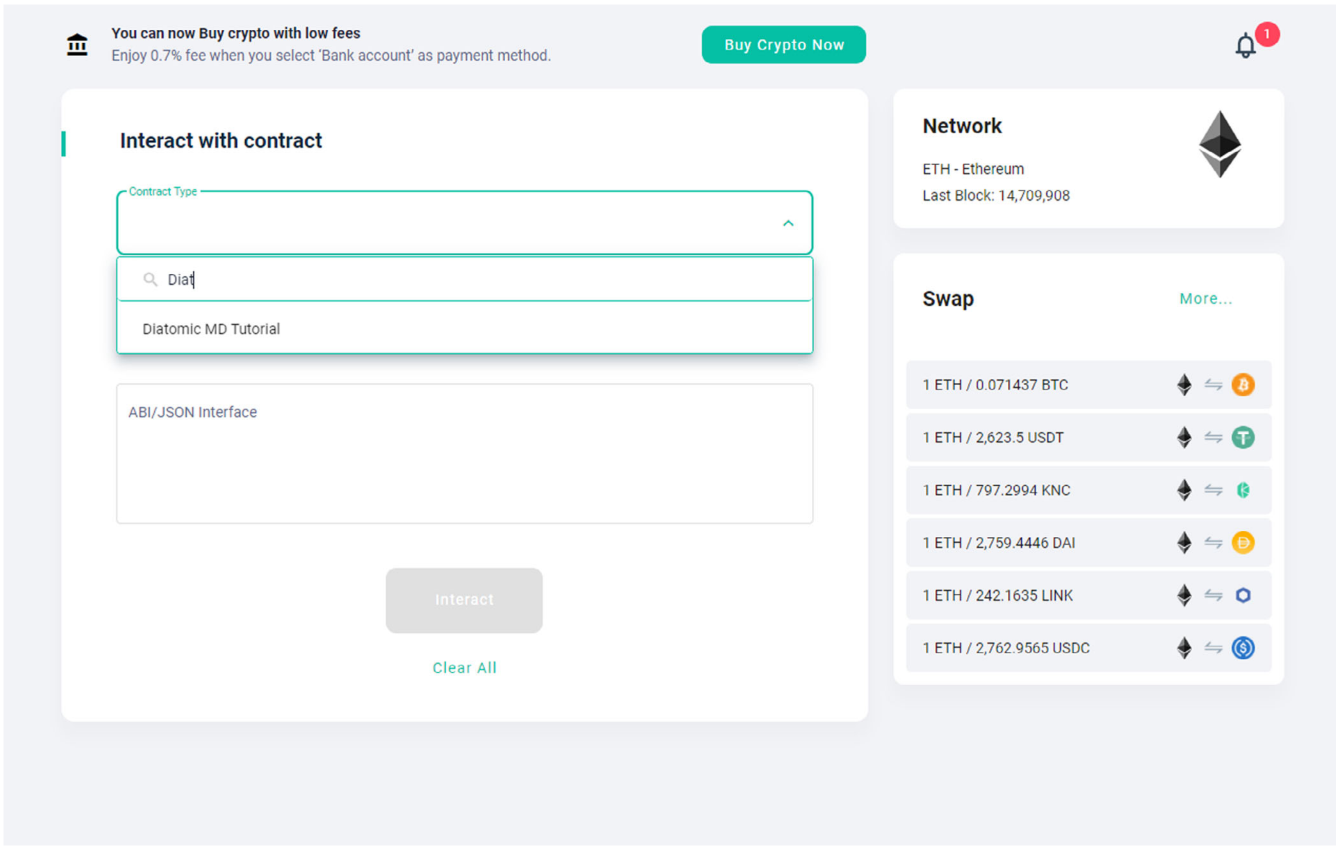
ETH - Ethereum

Last Block: 14,706,736

Swap

1 ETH / 0.072128 BTC
1 ETH / 2,581.22 USDT
1 ETH / 753.257 KNC
1 ETH / 2,715.2705 DAI
1 ETH / 242.9681 LINK
1 ETH / 2,722.0611 USDC

FIGURE 18 Screen capture showing the "interact with contract" option for the deployment smart contract in MyEtherWallet



You can now Buy crypto with low fees

Enjoy 0.7% fee when you select 'Bank account' as payment method.

Interact with contract

Contract Type

Diat

Diatomic MD Tutorial

ABI/JSON Interface

Network

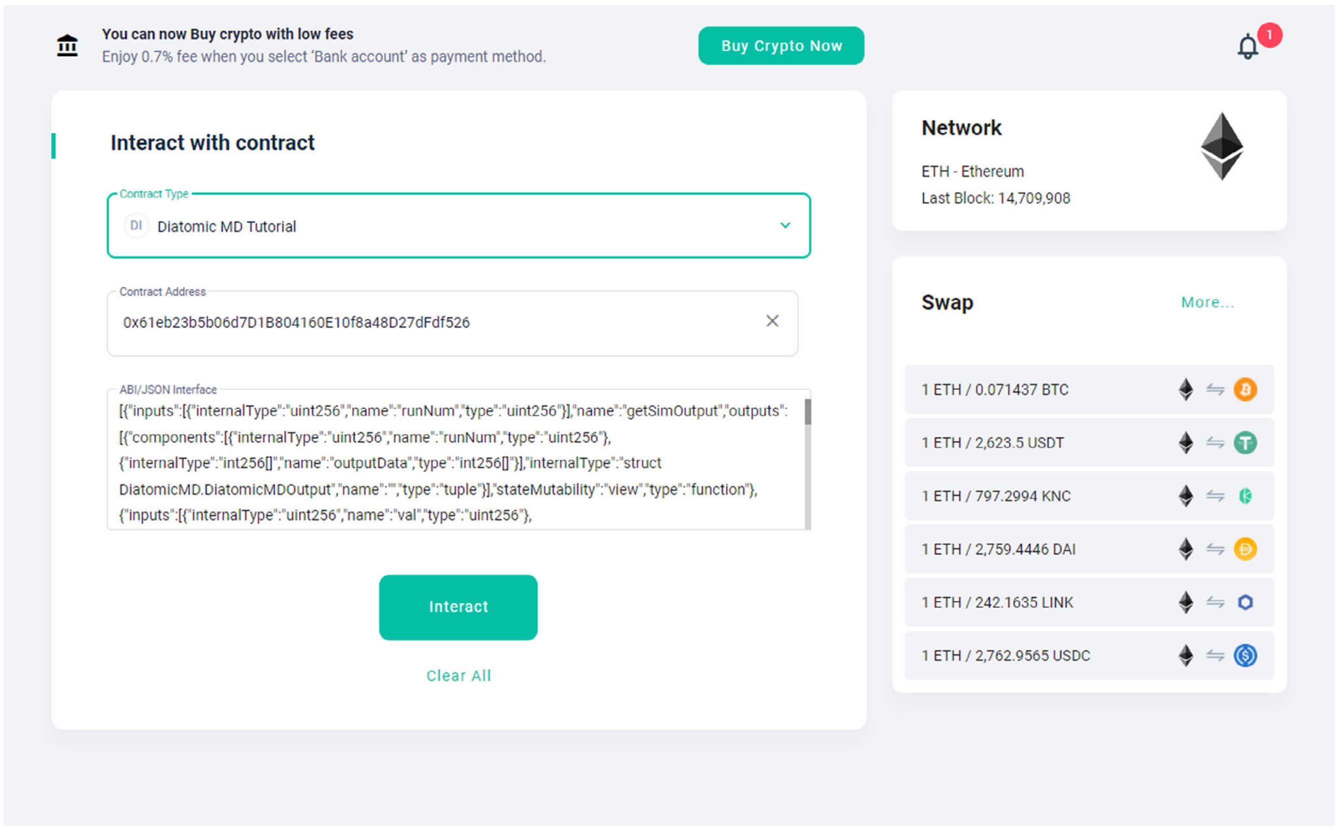
ETH - Ethereum

Last Block: 14,709,908

Swap

1 ETH / 0.071437 BTC
1 ETH / 2,623.5 USDT
1 ETH / 797.2994 KNC
1 ETH / 2,759.4446 DAI
1 ETH / 242.1635 LINK
1 ETH / 2,762.9565 USDC

FIGURE 19 Screen capture of the contract listed in MyEtherWallet (MEW)'s search after deployment



You can now Buy crypto with low fees
Enjoy 0.7% fee when you select 'Bank account' as payment method.

Interact with contract

Contract Type: Diatomic MD Tutorial

Contract Address: 0x61eb23b5b06d7D1B804160E10f8a48D27dFdf526

ABI/JSON Interface:

```
[{"inputs":[{"internalType":"uint256","name":"runNum","type":"uint256"}],"name":"getSimOutput","outputs":
[{"components":[{"internalType":"uint256","name":"runNum","type":"uint256"},
{"internalType":"int256[]","name":"outputData","type":"int256[]"}],"internalType":"struct
DiatomicMD.DiatomicMDOutput","name":"","type":"tuple"}],"stateMutability":"view","type":"function"},
{"inputs":[{"internalType":"uint256","name":"val","type":"uint256"},
```

Network
ETH - Ethereum
Last Block: 14,709,908

Swap

1 ETH / 0.071437 BTC
1 ETH / 2,623.5 USDT
1 ETH / 797.2994 KNC
1 ETH / 2,759.4446 DAI
1 ETH / 242.1635 LINK
1 ETH / 2,762.9565 USDC

FIGURE 20 Screen capture of the contract once selected, with address and ABI fields populated

10.3 | MyEtherWallet

MEW is a website that allows users to view and manage their account on the Ethereum blockchain using a variety of different wallets. It offers a simple graphical user interface (GUI) for both deploying and interacting with smart contracts, and it supports both of the browser wallets suggested for this tutorial alongside numerous other options. While it is possible to deploy a contract in many other ways, MEW's GUI has been deemed to be a particularly user friendly option. MEW can be accessed through myetherwallet.com.

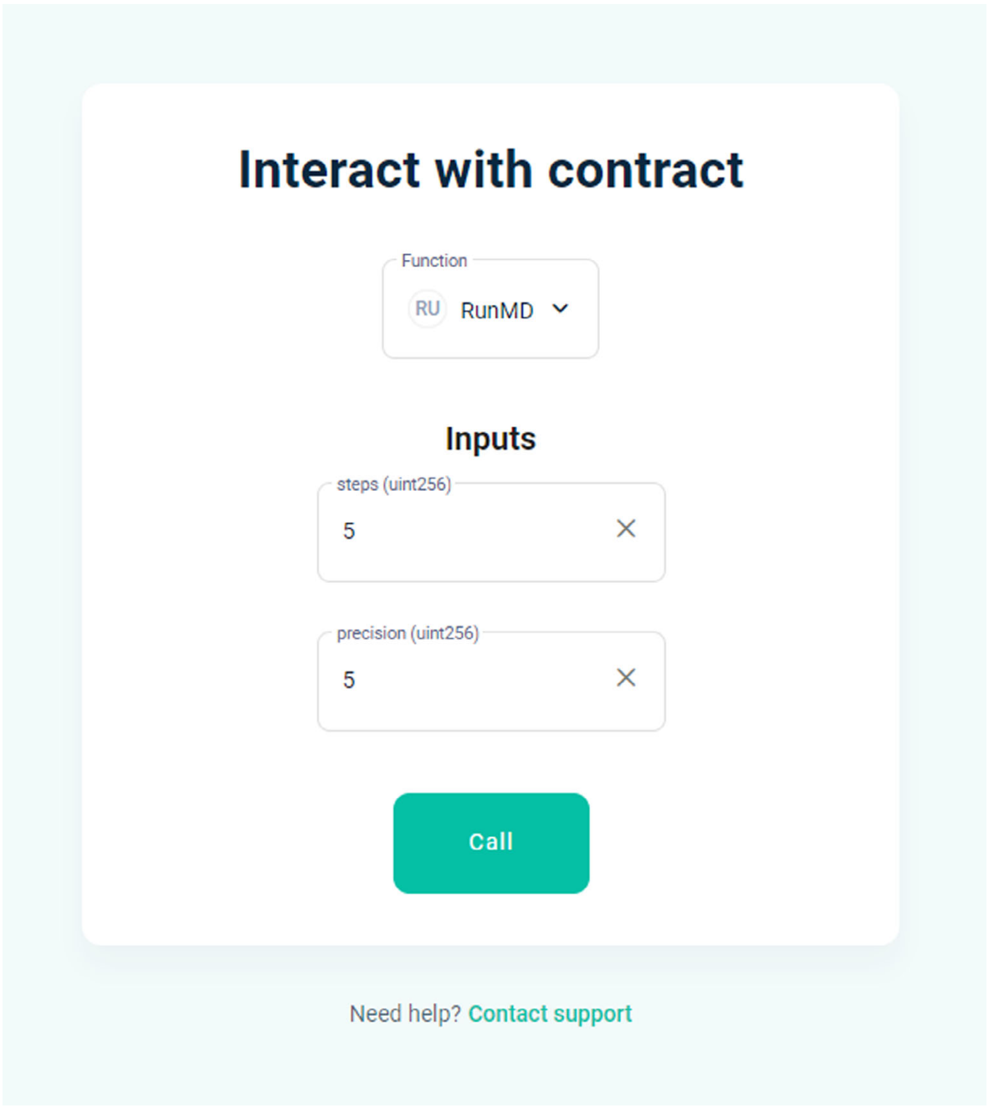
10.4 | Off-chain smart contract testing

Before deploying a contract, it first needs to be compiled into bytecode that can be understood by the EVM. During this stage, it is also important to test the contract to ensure that it functions as intended, since, once deployed, the contract cannot be altered or removed, and each new deployment incurs an additional cost in transaction fees.

To test the contract, the first step is to open Remix IDE. By default, Remix will provide an example workspace with sample code. A new workspace is not required and the existing sample code can be safely ignored as it will not be used. To begin, a new file must be created by clicking the document icon located below the workspace name as shown in Figure 3. In this instance the file is named "diatomicmd.sol."

Once created, the new diatomicmd.sol file must be opened and the contents of the provided contract pasted in, ensuring that the entire contract is present and that no lines have been missed. An example of this is shown in Figure 4. The contract begins with a "pragma solidity" statement denoting the version number of the compiler to use and ends with a closing curly brace "}" on the final line.

Once the contract code has been entered into Remix's text editor, it must be compiled before it can be tested or deployed. On the toolbar to the left, clicking the third item listed will open the Solidity Compiler tab. Once opened, a button will be displayed with the text "Compile diatomicmd.sol," shown in Figure 5, which after being clicked, will have prepared the bytecode and application binary interface (ABI) for this contract. The ABI is an interface that describes the functions of a contract, which makes it possible for other applications to understand how to interact with the deployed contract itself. A single compiler warning should also appear below this panel, but can be safely ignored for the purposes of this tutorial.


Interact with contract

Function: RunMD

Inputs

steps (uint256): 5

precision (uint256): 5

FIGURE 21 Screen capture showing MyEtherWallet's contract interaction panel with the RunMD function selected

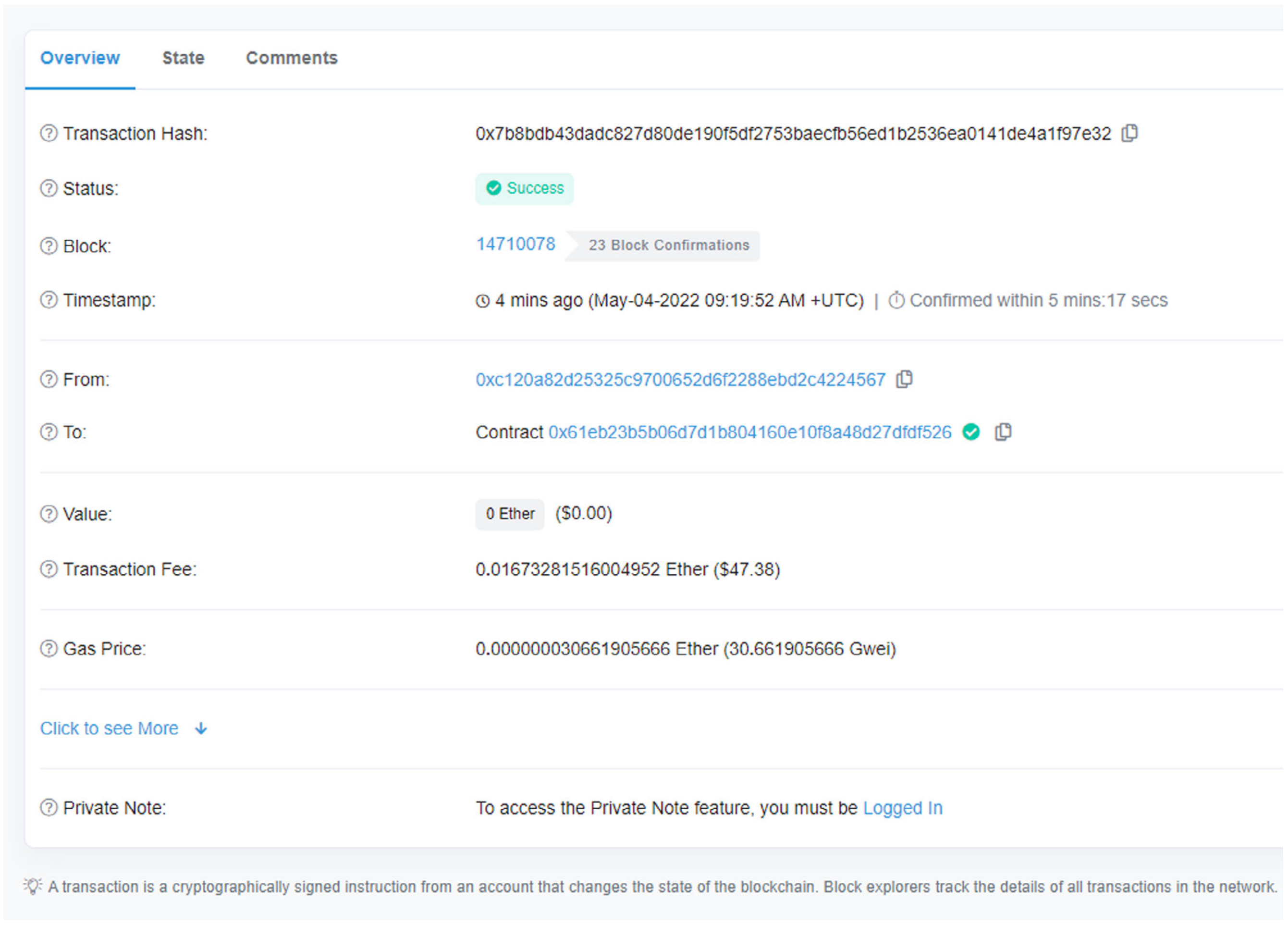
With the contract compiled, it needs to then be deployed to an environment for testing. Remix IDE supports running an EVM inside the local web browser using JavaScript so that contracts can be tested without being deployed to a live or public environment. Clicking the fourth item in the toolbar on the left will open the "Deploy & Run Transactions" tab.

The environment can then be set to the JavaScript VM option, and with the contract set to the DiatomicMD contract that was compiled, the "Deploy" button shown in Figure 6 can then be clicked to deploy the contract for testing outside of the blockchain.

The contract functions will be presented at the bottom of the tab and can be run in the browser using the JavaScript VM. There will be two functions named "runMD," one accepting two parameters and one accepting eight. The two-parameter version will, when called, run the simulation using pre-configured parameters taken from the first simulation of carbon monoxide run on the EVM [60, 61].

The eight-parameter version can be used to run a custom simulation, however, each input variable will first need to be converted to bytes using the "getValueBytes" function, providing an integer representation of the input value and an integer indicating the number of decimal places, the output of which will then be used as the input for calling the "runMd" function. For example, if an input value was meant to be 1.25, you would call "getValueBytes" using 125 as the "val" parameter and 2 as the "precision" parameter.

Expand the second listed two-parameter "runMd" function and provide a number of steps and precision. In this case it has been carried out using five steps, with a precision of ten decimal places, shown in Figure 7. When ready, click "transact" to run the test simulation. Due to limitations running a copy of the EVM in a web browser with JavaScript, and the fact that software has not yet been designed with these types of simulation in mind, it is likely that longer running functions will cause the browser to stop responding or crash, while this is not an issue on a full Ethereum node, it is recommended to use a smaller number of steps to mitigate this during testing.



Overview State Comments

Transaction Hash:	0x7b8bdb43dadc827d80de190f5df2753baecfb56ed1b2536ea0141de4a1f97e32
Status:	Success
Block:	14710078 23 Block Confirmations
Timestamp:	4 mins ago (May-04-2022 09:19:52 AM +UTC) | Confirmed within 5 mins:17 secs
From:	0xc120a82d25325c9700652d6f2288ebd2c4224567
To:	Contract 0x61eb23b5b06d7d1b804160e10f8a48d27dfdf526
Value:	0 Ether ($0.00)
Transaction Fee:	0.01673281516004952 Ether ($47.38)
Gas Price:	0.000000030661905666 Ether (30.661905666 Gwei)
Click to see More	
Private Note:	To access the Private Note feature, you must be Logged In

A transaction is a cryptographically signed instruction from an account that changes the state of the blockchain. Block explorers track the details of all transactions in the network.

FIGURE 22 Screen capture showing the Etherscan block explorer data associated with this new molecular dynamics trajectory of carbon monoxide

At the bottom of the debug window, two transactions will now be listed (shown in Figure 8) and can be expanded. The first transaction is the deployment of the contract itself, while the second is the test execution of the simulation. Expanding the second transaction and looking at the "decoded input" line will list the parameters that were entered previously, while the "decoded output" line will list the results of the simulation as a comma separated list.

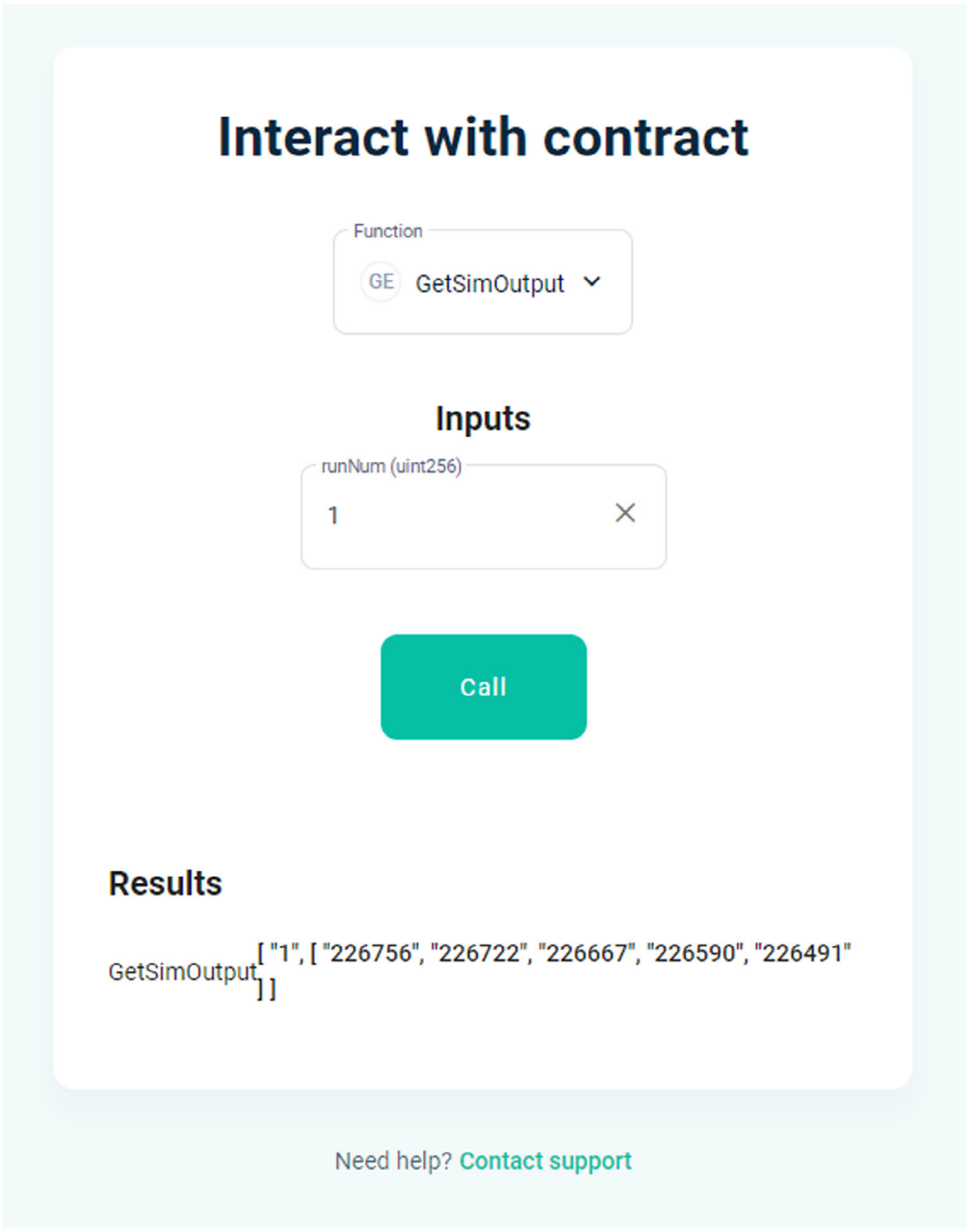
The first value in this list is the run number, which can be used later to retrieve these results, with the following values being the raw simulation output which must then be divided by $10^{\text{precision}}$ to produce the diatomic internuclear separation distances for each time step in atomic units.

With the simulation having been run, it is now possible to confirm that the results have been recorded by using the "getSimOutput" function. Expanding the "getSimOutput" menu in Figure 9, entering the run number of the test, and clicking "call," will add a further item below the previous two transactions shown in Figure 10. The "decoded input" line will list the run number that was entered and the "decoded output" will display the same results as the previous transaction.

10.5 | Blockchain contract deployment

Once satisfied that the contract is working as intended, the next step is to obtain the bytecode and ABI for the contract. The bytecode is the compiled contract in a form that the EVM can understand that will actually be deployed on the blockchain network.

Switching back to the "Deploy and Run Transactions" tab there should be two buttons to copy both the bytecode and ABI. These buttons are located at the bottom of Figure 11. These will be needed in the next steps, and should each be copied and recorded where they can be easily retrieved during later steps.



```
GetSimOutput [ "1", [ "226756", "226722", "226667", "226590", "226491" ] ]
```

FIGURE 23 Screen capture showing the results of this molecular dynamics trajectory

With the ABI and bytecode prepared, the next step is to switch to MEW and access MetaMask or the Brave wallet from within the Brave browser. In order to do this, navigate to the MEW website and click the “Access My Wallet” button (Figure 12), followed by the Browser extension button (Figure 13).

If using a different wallet from those suggested but still using MEW, select the appropriate option on the MEW site and follow the instructions provided. The remaining steps will be identical to using the suggested wallets with the only exception being the process of signing transactions.

When the wallet is connected, expand the “Contract” section and navigate to the “Deploy Contract” page in Figure 14 using the menu on the left side of the webpage.

Once the webpage loads, paste the bytecode and ABI into the two text areas, and enter a name for the smart contract. Only the bytecode is deployed to the blockchain, while the ABI and smart contract name are recorded by the website to simplify accessing the contract in the future. While the contract name is unimportant, the ABI should be retained as there is no guarantee that MEW will maintain a record of it and without a recognized standard for contracts of this type, interacting with the contract without the ABI may be difficult.

TABLE 2 The carbon monoxide bond lengths calculated using the Ethereum virtual computer in blocks 14710078 and 9360178 [60, 61]

Time (fs)	R_{CO} (a_0) Block 14710078	R_{CO} (a_0) Block 9360178
0.1	2.2676	2.2676
0.2	2.2672	2.2672
0.3	2.2667	2.2667
0.4	2.2659	2.2659
0.5	2.2649	2.2649

TABLE 3 Decimal block numbers, hexadecimal block header hashes, and hexadecimal transaction identification numbers for the simulation in this work

Contract deployment	
Block number	14709931
Transaction ID	0xa3f600317f39a6e065299b506c2e950676befc3080dce1d4d68eafdec7c5bdbc
Block header hash	0x03eefc07fcefc9d06f43c877986d7394deb065e445a5a5234e039bb659b262fb
Molecular dynamics simulation	
Block number	14710078
Transaction ID	0x7b8bdb43dadc827d80de190f5df2753baecfb56ed1b2536ea0141de4a1f97e32
Block header hash	0x440c0aec7ed4ceff351575fd9f72693e4c61f232e68fddf70fd9155d11c14ff7

With the contract data filled in, the “Sign Transaction” button in Figure 15 can be clicked to deploy the contract onto the blockchain. Once a smart contact is deployed it can no longer be changed, and modifying any software contained in the smart contract will therefore require that an updated contract be deployed separately, so a user may wish to take a moment to ensure that everything is filled out correctly at this point.

After clicking the “Confirm & Send” button shown in Figure 16, the wallet will prompt for the transaction to be cryptographically signed, at which point it will be broadcast to the Ethereum network. MEW will also update to confirm that the transaction has been initiated. The transaction can be viewed, and progress of the broadcast can be monitored, by selecting the “View on Etherscan” option.

Recommended practice is to copy and save the contract address from Etherscan for future reference. This may not be needed on MEW initially, but can save time in the future when calling the contract and looking up transactions. The contract address is listed in the “To:” section on the Etherscan block explorer (see Figure 17).

Should you lose or forget to record the contract address, it can always be retrieved from the transaction history of the account used to deploy the contract.

With the contract now deployed, in this case in block 14709931 of the Ethereum blockchain with the transaction identification number 0xa3f600317f39a6e065299b506c2e950676befc3080dce1d4d68eafdec7c5bdbc, return to MEW and switch to the “Interact with Contract” section in the menu (Figure 18).

Select the contract in the contract dropdown menu (Figure 19). If the contract is not listed by name, copy the contract address from earlier into the appropriate field and add the ABI (see Figure 20).

MEW does not, at time of writing, support calling overloaded functions (having two functions with the same name but a different set of parameters). When running the simulation with custom inputs this will not be an issue, however, it will not be possible to call the simplified function used in testing, that is, to recreate the original simulations of carbon monoxide, without an altered ABI on MEW. For simplicity, this tutorial uses a slightly altered ABI which is provided in the Supporting Information. This is a limitation of MEW specifically, and not a limitation of Ethereum itself.

With the address and ABI filled in, click “Interact” and then the “RunMD” function from the list provided. For demonstration purposes we have opted for five steps with a precision of five (Figure 21). Click “Call” and sign the transaction just as before when deploying the contract, this will likely take longer to complete and can again be monitored using Etherscan or an equivalent block explorer (see Figure 22).

The simulation has now taken place on the Ethereum blockchain. In this case the simulation of carbon monoxide was recorded in Ethereum block 14710078, with a transaction ID of 0x7b8bdb43dadc827d80de190f5df2753baecfb56ed1b2536ea0141de4a1f97e32 (Table 3). Once completed, return to MEW and select the “GetSimOutput” function, enter “1” as the “runNum” parameter and click “Call.” This function is call-only and has no cost, nor will it be recorded on the blockchain. If everything has worked correctly, the output of the simulation will be retrieved and displayed below in the “Results” section, starting with the run number and followed by an array of values. These values will need to be multiplied by $10^{-\text{precision}}$ in order to produce the final intended output for this contract, and should again be quoted to one fewer decimal places than the precision specification. The results for this simulation of carbon monoxide are shown in unprocessed form in Figure 23, and compared with the original outputs from simulation of carbon monoxide in block 9360178 in Table 2. Finally, the block header hashes for the specific blocks that contained the simulation data can be taken either directly from the blockchain or using block explorer software like Etherscan to provide extra information about the blocks reproducibility. The block header hashes for the blocks in which the software deployment and molecular dynamics run took place are given with the block numbers and transaction ID numbers in Table 3.

10.6 | Conclusions and perspective

This tutorial provides a template for researchers looking to perform scientific computation using blockchain computers. Blockchains can provide clear advantages over conventional computers in terms of error recording, computational reproducibility, provenance and determining calculation order, and censorship resistance in scientific work. The use of blockchain cryptography can also aid in federated calculations that involve data sharing. The high costs relative to centralized architectures currently make blockchains less useful for routine calculations where these properties carry less of a premium. However, it is our view that blockchain calculations are likely to have an increasing impact as blockchain throughputs continue to scale. Hybrid calculations that use blockchains for certain computational steps and conventional off-chain computers for other steps may also allow more common use of these methods at a significantly reduced cost. In practical terms, while speculative, more developed blockchain calculations could prevent future uncertainties like the "war over supercooled water" between the research groups of Chandler and Debenedetti [68]. We look forward to seeing how this field develops over time.

AUTHOR CONTRIBUTIONS

Magnus W. D. Hanson-Heine: Conceptualization; data curation; project administration; supervision; writing – original draft. **Alexander P. Ashmore:** Software; writing – original draft.

CONFLICT OF INTEREST

The authors declare no competing financial interest. It has come to our attention that the Ethereum blockchain has begun a hard fork between the proof-of-work and proof-of-stake consensus algorithms while the present article was under peer review.

DATA AVAILABILITY STATEMENT

The data generated during this study is available in this article, and is also available on the Ethereum blockchain in the referenced blocks.

ORCID

Magnus W. D. Hanson-Heine https://orcid.org/0000-0002-6709-297X
Alexander P. Ashmore https://orcid.org/0000-0001-7498-8873

REFERENCES

[1] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008 Bitcoin.org, https://bitcoin.org/bitcoin.pdf (accessed: June 22 2019).
[2] D. R. Wong, S. Bhattacharya, A. J. Butte, *Nat. Commun.* **2019**, *10*(1), 917.
[3] M. Andoni, V. Robu, D. Flynn, S. Abram, D. Geach, D. Jenkins, P. McCallum, A. Peacock, *Renew. Sustain. Energy Rev.* **2019**, *100*, 143.
[4] V. Buterin, *A Next-Generation Smart Contract and Decentralized Application Platform*, 2014 (Cryptorating.eu) https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf. (accessed November 13 2019).
[5] T.-T. Kuo, X. Jiang, H. Tang, X. F. Wang, T. Bath, D. Bu, L. Wang, A. Harmanci, S. Zhang, D. Zhi, H. J. Sofia, L. Ohno-Machado, *BMC Med. Genom.* **2020**, *13*(7), 98.
[6] D. Grishin, K. Obbad, G. M. Church, *Nat. Biotechnol.* **2019**, *37*(10), 1115.
[7] G. Gürsoy, C. M. Brannon, M. Gerstein, *BMC Med. Genom.* **2020**, *13*(1), 74.
[8] N. D. Pattengale, C. M. Hudson, *BMC Med. Genom.* **2020**, *13*(7), 102.
[9] T.-T. Kuo, T. Bath, S. Ma, N. Pattengale, M. Yang, Y. Cao, C. M. Hudson, J. Kim, K. Post, L. Xiong, L. Ohno-Machado, *Int. J. Med. Inform.* **2021**, *154*, 104559.
[10] S. Ma, Y. Cao, L. Xiong, *BMC Med. Genom.* **2020**, *13*(7), 91.
[11] H. I. Ozercan, A. M. Ileri, E. Ayday, C. Alkan, *Genome Res.* **2018**, *28*(9), 1255.
[12] M. S. Ozdayi, M. Kantarcioglu, B. Malin, *BMC Med. Genom.* **2020**, *13*(7), 82.
[13] G. Gürsoy, R. Bjornson, M. E. Green, M. Gerstein, *BMC Med. Genom.* **2020**, *13*(7), 78.
[14] T.-T. Kuo, R. A. Gabriel, L. Ohno-Machado, *J. Am. Med. Inform. Assoc.* **2019**, *26*(5), 392.
[15] B. Adanur Dedeturk, A. Soran, B. Bakir-Gungor, *PeerJ* **2021**, *9*, e12130.
[16] F. Carlini, R. Carlini, S. Dalla Palma, R. Pareschi, F. Zappone, *Front. Blockchain* **2020**, *3*(57), 1.
[17] V. Racine, *Sci. Eng. Ethics* **2021**, *27*(3), 35.
[18] X.-L. Jin, M. Zhang, Z. Zhou, X. Yu, *J. Med. Internet Res.* **2019**, *21*(9), e13587.
[19] B. S. Glicksberg, S. Burns, R. Currie, A. Griffin, Z. J. Wang, D. Haussler, T. Goldstein, E. Collisson, *J. Med. Internet Res.* **2020**, *22*(3), e16810.

[20] M. Shabani, *J. Am. Med. Inform. Assoc.* **2019**, *26*(1), 76.
[21] M. Kang, V. Lemieux, *Ledger* **2021**, *6*, 126.
[22] D. R. Ortega, C. M. Oikonomou, H. J. Ding, P. Rees-Lee, Alexandria, G. J. Jensen, *PLoS One* **2019**, *14*(4), e0215531.
[23] Y. Xu, R. Liu, J. Li, Y. Xu, X. Zhu, *J. Phys. Chem. Lett.* **2020**, *11*(23), 9995.
[24] H. Kim, S. Kim, J. Y. Hwang, C. Seo, *IEEE Access* **2019**, *7*, 136481.
[25] F. Chen, H. Wan, H. Cai, G. Cheng, *Can. J. Stat.* **2021**, *49*(4), 1364.
[26] D. Li, D. Han, T.-H. Weng, Z. Zheng, H. Li, H. Liu, A. Castiglione, K.-C. Li, *Soft Comput.* **2021**, *26*, 4423.
[27] M. Imran, U. Zaman, Imran, J. Imtiaz, M. Fayaz, J. Gwak, *Electronics* **2021**, *10*(20), 2501.
[28] F. Zerka, V. Urovi, A. Vaidyanathan, S. Barakat, R. T. H. Leijenaar, S. Walsh, H. Gabrani-Juma, B. Miraglio, H. C. Woodruff, M. Dumontier, P. Lambin, *IEEE Access* **2020**, *8*, 183939.
[29] V. Drungilas, E. Vaičiukynas, M. Jurgelaitis, R. Butkienė, L. Čeponienė, *Appl. Sci.* **2021**, *11*(3), 1010.
[30] E. A. Mantey, C. Zhou, J. H. Anajemba, I. M. Okpalaoguchi, O. D.-M. Chiadika, *Front. Public Health.* **2021**, *9*(1256), 737269.
[31] Y. Liu, Y. Qu, C. Xu, Z. Hao, B. Gu, *Sensors* **2021**, *21*(10), 3335.
[32] R. Zhang, M. Song, T. Li, Z. Yu, Y. Dai, X. Liu, G. Wang, *J. Syst. Archit.* **2021**, *118*, 102205.
[33] J. Weng, J. Zhang, M. Li, Y. Zhang, W. Luo, *IEEE Trans. Dependable Secure Comput.* **2021**, *18*(5), 2438.
[34] H. Jin, X. Dai, J. Xiao, B. Li, H. Li, Y. Zhang, *IEEE Internet Things J.* **2021**, *8*(21), 15776.
[35] M. Ali, H. Karimipour, M. Tariq, *Comput. Secur.* **2021**, *108*, 102355.
[36] D. Unal, M. Hammoudeh, M. A. Khan, A. Abuarqoub, G. Epiphaniou, R. Hamila, *Comput. Secur.* **2021**, *109*, 102393.
[37] M. Shen, H. Wang, B. Zhang, L. Zhu, K. Xu, Q. Li, X. du, *IEEE Internet Things J.* **2021**, *8*(4), 2265.
[38] Z. Mahmood, V. Jusas, *Symmetry* **2021**, *13*(7), 1116.
[39] C. Hickman, H. Alshubbar, J. Chambost, C. Jacques, C.-A. Pena, A. Drakeley, T. Freour, *Fertil. Steril.* **2020**, *114*(5), 927.
[40] U. Majeed, L. U. Khan, A. Yousafzai, Z. Han, B. J. Park, C. S. Hong, *IEEE Access* **2021**, *9*, 155634.
[41] C. Jiang, C. Xu, Y. Zhang, *Inf. Sci.* **2021**, *576*, 288.
[42] E. O. Kiktenko, N. O. Pozhar, M. N. Anufriev, A. S. Trushechkin, R. R. Yunusov, Y. V. Kurochkin, A. I. Lvovsky, A. K. Fedorov, *Quantum Sci. Technol.* **2018**, *3*(3), 035004.
[43] I. Stewart, D. Ilie, A. Zamyatin, S. Werner, M. F. Torshizi, W. J. Knottenbelt, *R. Soc. Open Sci.* **2018**, *5*(6), 180410.
[44] M. Edwards, A. Mashatan, S. Ghose, *Quantum Inf. Process.* **2020**, *19*(6), 184.
[45] Y.-L. Gao, X. B. Chen, G. Xu, K. G. Yuan, W. Liu, Y. X. Yang, *Quantum Inf. Process.* **2020**, *19*(12), 420.
[46] S. Singh, N. K. Rajput, V. K. Rathi, H. M. Pandey, A. K. Jaiswal, P. Tiwari, *Neural Process. Lett* **2020**, 1.
[47] C. Li, X. Chen, Y. Chen, Y. Hou, J. Li, *IEEE Access* **2019**, *7*, 2026.
[48] G. Iovane, *IEEE Access* **2021**, *9*, 39827.
[49] S. Banerjee, A. Mukherjee, P. K. Panigrahi, *Phys. Rev. Res.* **2020**, *2*(1), 013322.
[50] Q. Li, M. Luo, C. Hsu, L. Wang, D. He, *IEEE Syst. J.* **2022**, *16*, 4816.
[51] P. Zhang, L. Wang, W. Wang, K. Fu, J. Wang, *Secur. Commun. Netw.* **2021**, *2021*, 6671648.
[52] Z. Cai, J. Qu, P. Liu, J. Yu, *IEEE Access* **2019**, *7*, 138657.
[53] J. Nicola, *Nature* **2021**, *594*(7864), 481.
[54] Z. C. Kennedy, D. E. Stephenson, J. F. Christ, T. R. Pope, B. W. Arey, C. A. Barrett, M. G. Warner, *J. Mater. Chem. C* **2017**, *5*(37), 9570.
[55] J. J. Sikorski, J. Haughton, M. Kraft, *Appl. Energy* **2017**, *195*, 234.
[56] J. Leng, P. Jiang, K. Xu, Q. Liu, J. L. Zhao, Y. Bian, R. Shi, *J. Cleaner Prod.* **2019**, *234*, 767.
[57] S. S. Takhar, K. Liyanage, *Int. J. Internet Technol. Secur. Trans.* **2021**, *11*(1), 75.
[58] K. Bhubalan, A. M. Tamothran, S. H. Kee, S. Y. Foong, S. S. Lam, K. Ganeson, S. Vigneswari, A.-A. Amirul, S. Ramakrishna, *Environ. Res.* **2022**, *213*, 113631.
[59] H. Lu, K. Huang, M. Azimi, L. Guo, *IEEE Access* **2019**, *7*, 41426.
[60] M. W. D. Hanson-Heine, A. P. Ashmore, *Chem. Sci.* **2020**, *11*(18), 4644.
[61] M. W. D. Hanson-Heine, A. P. Ashmore, *J. Phys. Chem. Lett.* **2020**, *11*(16), 6618.
[62] C. Dwork, M. Naor, Pricing via processing or combatting junk mail. in *Advances in Cryptology—CRYPTO '92* (Ed: E. F. Brickell), Springer, Berlin Heidelberg **1993**, p. 139.
[63] W. C. Swope, H. C. Andersen, P. H. Berens, K. R. Wilson, *J. Chem. Phys.* **1982**, *76*(1), 637.
[64] W. D. Carey, *Science* **1983**, *219*(4587), 911.
[65] U. Hoßfeld, L. Olsson, *Nature* **2006**, *443*(7109), 271.
[66] C. Schinckus, *Renew. Sustain. Energy Rev.* **2021**, *152*, 111682.
[67] N. Lasla, L. Al-Sahan, M. Abdallah, M. Younis, *Comput. Netw.* **2022**, *214*, 109118.
[68] G. S. Ashley, *Phys. Today* **2018**, 1945. https://doi.org/10.1063/PT.6.1.20180822a

AUTHOR BIOGRAPHIES

Magnus Hanson-Heine is a researcher in Computational and Quantum Chemistry. He received an MCHEM masters degree in Chemistry in 2010 and was awarded a Ph.D. in 2014 from the University of Nottingham. He has been a contributor to the Q-CHEM quantum chemical modelling software since 2013. His research interests include molecular quantum mechanics using density functional theory and molecular vibrational theories, and since 2020 he has been researching the interface between blockchains and physical simulation.

Alexander Ashmore studied computing and IT through the Open University from 2014 through to 2020 when he received his BSc and started his research into blockchain applications. He has worked as a software developer helping to produce, maintain and support business and educational web applications since 2015.

SUPPORTING INFORMATION

Additional supporting information can be found online in the Supporting Information section at the end of this article.

How to cite this article:
M. W. D. Hanson-Heine, A. P. Ashmore, *Int. J. Quantum Chem.* **2022**, e27035. https://doi.org/10.1002/qua.27035

APPENDIX

This appendix details the steps of how to setup a blockchain wallet on the Ethereum blockchain and purchase Ether tokens to run a simulation of the kind outlined in this publication. The appendix assumes the use of the Chrome web browser and MetaMask wallet plug-in for this purpose at the time of writing.

1. Go to the Chrome web store and install the MetaMask plug-in. A copy of this plug-in can also be found through the MetaMask website (which is currently metamask.io).
2. Click the “Add to Chrome” button to install the plug-in.
3. Once installed, a new tab titled “Welcome to MetaMask” will open. Click the “Get started” button.
4. Click either the “No thanks” or “I agree” button to share diagnostic data with MetaMask based on personal preferences. Neither will directly affect any further steps in this process.
5. Click the “Create a wallet” button.
6. Enter a password of your choice and click “Create” to generate a new “wallet.”
7. The next page will allow you to watch a video explaining what a “recovery phrase” is in the context of cryptographic key pairs, as well as presenting some basic information on how to keep this phrase secure.
8. The next page will allow you to click to view your recovery phrase. You should record this somewhere secure, however, if you do not, it is possible to retrieve it from the MetaMask plug-in at a later date. Should this phrase be lost however, there is no method to retrieve it, and any tokens associated with the wallet will be lost.
9. Click either “Remind me later” or “Next.” If you click “Next,” you will be prompted to enter the phrase to confirm you have backed it up. Once done, click “Confirm.”
10. The wallet is now set up and ready to use. To obtain some Ether (ETH), which is the native token of the Ethereum blockchain, click the “Buy” button.
11. The option to select multiple payment gateways will then be presented where you can choose to purchase ETH with fiat currencies such as Dollars ($), Euros (€), or Pounds Sterling (£), and have it directly deposited into the wallet you have just set up. The payment gateway to select will depend on the preferred method to pay for the ETH (e.g., debit card, credit card, bank transfer, apple pay, etc).
12. Once installed, MetaMask is available from the top-right plug-in bar in the Chrome browser.
13. Should this wallet need to be used on a separate machine, repeat the same steps until the “Create a wallet” option is presented and instead click “Recover a wallet,” at which point you should enter the recovery phrase from the original setup to restore that wallet instead.