## .next\prerender-manifest.js


```
self.__PRERENDER_MANIFEST="{\"version\":4,\"routes\":{},\"dynamicRoutes\":{},
\"preview\":{\"previewModeId\":\"8bff34782e6fe3263e5a3239ff5e131e\",
\"previewModeSigningKey\":
\"a32fe539184ec4aef2a83efccfe17639a46e9c30baf9bc5a1354029cef15d5ff\",
\"previewModeEncryptionKey\":
\"beeb071723885a6cb1adc9bd9e0e682699e1d85186743b0b02e63e5583cd45a5\"},
\"notFoundRoutes\":[]}"
```

## .next\server\middleware-react-loadable-manifest.js


```
self.__REACT_LOADABLE_MANIFEST='{"_app.tsx -> @/components/
GlobalErrorHandler":{"id":7292,"files":["static/
chunks/292.e86441fef50959b5.js"]},"index.tsx -> @/components/
BlockchainContext":{"id":6945,"files":[]}}';
```

## .next\server\next-font-manifest.js

```
self.__NEXT_FONT_MANIFEST='{"pages":{},"app":
{},"appUsingSizeAdjust":false,"pagesUsingSizeAdjust":false}';
```

## .next\server\pages_document.js

```
"use strict";(()=>{var
e={};e.id=660,e.ids=[660],e.modules={2785:e=>{e.exports=require("next/dist/
compiled/next-server/pages.runtime.prod.js")},6689:e=>{e.exports=require("reac
t")},1017:e=>{e.exports=require("path")}};var r=require("../webpack-
runtime.js");r.C(e);var s=e=>r(r.s=e),t=r.X(0,[388],
()=>s(5388));module.exports=t})();
```

## .next\static\chunks\main-8d6a5ee0c19b3a83.js

```
(self.webpackChunk_N_E=self.webpackChunk_N_E||[]).push([[179],{2431:function()
{}},function(n){n.O(0,[802,27,290],function(){return
n(n.s=8031)}),_N_E=n.O()}]);
```

## .next\static\chunks\pages_error-a237099c62580823.js

```
(self.webpackChunk_N_E=self.webpackChunk_N_E||[]).push([[820],
{1981:function(n,_,u){(window.__NEXT_P=window.__NEXT_P||[]).push(["/
_error",function(){return u(8435)}])}},function(n){n.O(0,[756,135,473,429,71,2
10,490,475,572,334,767,234,898,785,913,748,663,693,328,917,185,791,604,888,802
,27,290,179],function(){return n(n.s=1981)}),_N_E=n.O()}]);//#
sourceMappingURL=_error-a237099c62580823.js.map
```

## .next\static\LF_VyIMe0tJgAyMxBw2NR_buildManifest.js


```
self.__BUILD_MANIFEST={__rewrites:{afterFiles:[],beforeFiles:[],fallback:
[]},"/":["static/chunks/pages/index-6e2eda49494341ce.js"],"/_error":["static/
chunks/pages/_error-a237099c62580823.js"],"/transactions":["static/chunks/
pages/transactions-fa1f2a601fe1700c.js"],sortedPages:["/","/_app","/_error","/
transactions"]},self.__BUILD_MANIFEST_CB&&self.__BUILD_MANIFEST_CB();
```

## .next\static\LF_VyIMe0tJgAyMxBw2NR_ssgManifest.js

```
self.__SSG_MANIFEST=new Set,self.__SSG_MANIFEST_CB&&self.__SSG_MANIFEST_CB();
```

## config\network_config.js

```
module.exports = {Ð   networks: {Ð     // Local and Ethereum NetworksÐ
development: {Ð       networkId: 5777,Ð       chainId: 5777,Ð       provider:
'http://127.0.0.1:9545',Ð       type: 'local',Ð       nativeCurrency: {Ð
          name: 'Ethereum',Ð         symbol: 'ETH',Ð         decimals: 18Ð
        }Ð     },Ð     mainnet: {Ð       networkId: 1,Ð       chainId: 1,Ð
        provider: 'https://mainnet.infura.io/v3/YOUR_INFURA_PROJECT_ID',Ð
        type: 'mainnet',Ð       nativeCurrency: {Ð         name: 'Ethereum',Ð
          symbol: 'ETH',Ð         decimals: 18Ð       }Ð     },Ð     sepolia:
{Ð       networkId: 11155111,Ð       chainId: 11155111,Ð       provider:
'https://sepolia.infura.io/v3/YOUR_INFURA_PROJECT_ID',Ð       type: 'testnet',Ð
        nativeCurrency: {Ð         name: 'Sepolia ETH',Ð         symbol:
'ETH',Ð         decimals: 18Ð       }Ð     },Ð     Ð     // Polkadot and
Substrate NetworksÐ     polkadot: {Ð       networkId: 1,Ð       chainId: 1,Ð
        provider: 'wss://rpc.polkadot.io',Ð       type: 'mainnet',Ð
nativeCurrency: {Ð         name: 'DOT',Ð         symbol: 'DOT',Ð
decimals: 10Ð       }Ð     },Ð     kusama: {Ð       networkId: 2,Ð
chainId: 2,Ð       provider: 'wss://kusama-rpc.polkadot.io',Ð       type:
'mainnet',Ð       nativeCurrency: {Ð         name: 'KSM',Ð         symbol:
'KSM',Ð         decimals: 12Ð       }Ð     },Ð     Ð     // Layer 2 and
Alternative NetworksÐ     polygon: {Ð       networkId: 137,Ð       chainId:
137,Ð       provider: 'https://polygon-rpc.com',Ð       type: 'mainnet',Ð
        nativeCurrency: {Ð         name: 'Matic',Ð         symbol: 'MATIC',Ð
          decimals: 18Ð       }Ð     },Ð     arbitrum: {Ð       networkId:
42161,Ð       chainId: 42161,Ð       provider: 'https://arb1.arbitrum.io/rpc',Ð
        type: 'mainnet',Ð       nativeCurrency: {Ð         name: 'ETH',Ð
          symbol: 'ETH',Ð         decimals: 18Ð       }Ð     },Ð
optimism: {Ð       networkId: 10,Ð       chainId: 10,Ð       provider:
'https://mainnet.optimism.io',Ð       type: 'mainnet',Ð       nativeCurrency:
{Ð         name: 'ETH',Ð         symbol: 'ETH',Ð         decimals: 18Ð       }Ð
      },Ð     Ð     // Binance Smart ChainÐ     bsc: {Ð       networkId: 56,Ð
          chainId: 56,Ð       provider: 'https://bsc-dataseed.binance.org/',Ð
          type: 'mainnet',Ð       nativeCurrency: {Ð         name: 'BNB',Ð
          symbol: 'BNB',Ð         decimals: 18Ð       }Ð     }Ð   },Ð   Ð
    // Default network configurationÐ   default: 'development'Ð };
```

## constants\constants.js

```
/**truffle(develop)> const instance = await RewardDistributor.deployed()Ð
undefinedÐtruffle(develop)> instance.address */Ð*Truffle Develop started at
http://127.0.0.1:9545/Ðaccounts:Ð(0) 0xe2021b32656764d3e3fccbbf98ca77bb24ffeb8eÐ
(1) 0xad0829b177df9780886109bcf1a0afd305bbe628Ð(2)
0x14a1ca9dc964ae5e9329ae6bebd7cc3033bb4f73Ð(3)
0x81a6e5e444c66ba321f24cb208a95e33f6e1b2f1Ð(4)
0x0e7251415d77f2683004bf4f831d0265854bc7adÐ(5)
0xc06e9fd2730de26fa9f59f52b085a6156251d012Ð(6)
0xf19351d8f57576fb85fd9a44b65432e3109f15fbÐ(7)
0xa223befb3669a97efd219cb30b015cbcd9b9a041Ð(8)
0x830dc3749ba6d6b34246bc3b93f695b3b7160a75Ð(9)
0xa7a80d88260b308a73cb3a1687a460ed1e4aed68Ðrivate Keys:Ð(0)
8f9b9fb10b97336177d0064b52e14a35b9db4bc1875094e88cd417eadedf3a52Ð(1)
294fb00e4eb3ff2962cfa46094f82bc730ed6fff73e2004b6981a3bc618bc767Ð(2)
709612db00c366af9b2451fcaee77d3eec45ff843ac27af4d09cdc3cd83b9c41Ð(3)
6e69669e2117db4e5e9ee4382101efa52970f289b8169a22417036300319067dÐ(4)
0a7f022df7c539422ebbf1553d1967c3d1a66902f5940e269103325824f7d7b2Ð(5)
652c8624fe5961f40b3aad5f6940ac1779bf0c8f5597c0b783abe4ede9c287daÐ(6)
ed1cd1db05cce0c452ae1faf20b9268237601e948c5f70fab698334e74b1709bÐ(7)
aa0699e66d4d07e92c3566e17926fb7d47f49f02f67791a5dafbbe40c2d41ef7Ð(8)
11a94bce9477132329049a92487791a257454152ab958307f24509f283a79b14Ð(9)
1966fa15b8d23641c883d30be713c9786758d33d1c75b027e8ece3875096a0a0Ð/Ðconst
BlockchainRegistryBase =  '0xFB61F22BacF92C32a3Fa4ECcF6DB5c12Eb71654d';Ðconst
BlockchainRegistry = '0x71db4933bf0342b290ada075b9b40f61Cac4A88D';Ðconst
BlockchainMonitor =    '0x56e3b418E151A6a70262f337b5c4115415dB2592';Ðconst
ChaCha20Poly1305 =  '0x3dB865F0181A652EE731adf92557677D609f4705';Ðconst
MetadataParser = '0x65C7dEBC72689bEea81cC670b621fF57f97795db';Ðconst
PacechainChannel = '0xCdB1B7246Ba7dF364C773b8311e5B905B2b23386';Ðconst
SpeculativeTransactionHandler = '0xa6d512B846D3bcA0498C4178577d8c55900FDAFE';Ð
const ConfidenceScoreCalculator =
'0xF352c192e1E4aA3618Df70b8de510189BF5cdd5F';Ðconst AssetTransferProcessor =
'0x6E3372f20f931d16679bA21Ebcd63C2eAd4E1B4b';Ðconst TransactionValidator =
'0x59b92Ab6e4054f752C7E8ac5Cb846B9eCD67F5a3';Ðconst RewardDistributor =
'0x8c771E0dD72fF614dB5d3cf16c38D76BB4E62872';Ðconst RewardToken =
'0x44cedb676894FC67450D51DfE4a03e7f294605C9';Ðconst ProofOfStakeValidator =
'0xCe8e536957Cc8115d7c7FE8FB1b03b4Ed32a5b20';Ðconst   StateManager
='0xBC0EE66Cc35B3E2732D1E6301781f0A1DDb828E7';Ðconst TransactionRelay =
'0xcdA245B0b4e2006eb200c6F2505B2c914a379D74';Ð const
ReceivingBlockchainInterface = '0xAAdeB2f7B9c7285D56A06c4aA4D91Cd5d35D1030';ÐÐ
```

## dataanalysis\analyzed_data.js

```
// First, let's examine the structure of the dataconst fileContent = await
window.fs.readFile('egovernance.dta', { encoding: 'binary' });
// Since this is a Stata file, we'll need to parse it differently// We'll
examine key variables from the data to understand their distribution
// Let's create functions to analyze the datafunction
analyzeCorrelation(data, variables) {  // Function to analyze correlation
between variables  console.log("Correlation Analysis:");  // This would
require actual Stata file parsing} // From the document, we know the key
variables:// E-governance: EGI, EPI, OSI, HCI, TII// Institutional quality:
COC, GE, PSTAB, RQ, RL, VA// Corruption: CPI// Let's work with what we can
extract from the document                          // Based on the data we
have, let's create a sample datasetconst countries = [  "Algeria", "Benin",
"Botswana", "Burkina Faso", "Cameroon", "Cote d'Ivoire",  "Egypt",
"Ethiopia", "Ghana", "Guinea", "Kenya", "Madagascar", "Malawi",  "Mali",
"Mauritius", "Morocco", "Mozambique", "Namibia", "Niger", "Nigeria",
"Rwanda", "Senegal", "South Africa", "Tanzania", "Togo", "Tunisia", "Uganda",
  "Zambia", "Zimbabwe"];  // Creating a synthetic dataset for analysis based
on the variables mentionedconst sampleData = [];                       for
(let i = 0; i < countries.length; i++) {  for (let year = 2013; year <= 2022;
year++) {    // Sample data point    const dataPoint = {      country:
countries[i],      year: year,      CPI: Math.random() * 100, // Corruption
Perception Index (0-100)      EGI: Math.random(), // E-Government Index
EPI: Math.random(), // E-Participation Index      OSI: Math.random(), //
Online Service Index      HCI: Math.random(), // Human Capital Index
TII: Math.random(), // Telecom Infrastructure Index      COC: Math.random() *
2 - 1, // Control of Corruption (-1 to 1)      GE: Math.random() * 2 - 1, //
Government Effectiveness (-1 to 1)      PSTAB: Math.random() * 2 - 1, //
Political Stability      RQ: Math.random() * 2 - 1, // Regulatory Quality
      RL: Math.random() * 2 - 1, // Rule of Law      VA: Math.random() * 2 -
1, // Voice and Accountability      ICT: Math.random(), // ICT Score
GDPPC: Math.random() * 20000, // GDP per capita      UPG: Math.random() *
5, // Urban population growth      TNRR: Math.random() * 30 // Total natural
resources rents    };    sampleData.push(dataPoint);  }} // Calculating
average governance and institutional quality scoresfunction
calculateAverages(data) {  const countryAverages = {};    for (const item of
data) {    if (!countryAverages[item.country]) {
countryAverages[item.country] = {        CPI: 0,        eGovIndex: 0, //
Average of EGI, EPI, OSI, HCI, TII        instQuality: 0, // Average of COC,
GE, PSTAB, RQ, RL, VA        count: 0      };    }        const eGovAvg =
(item.EGI + item.EPI + item.OSI + item.HCI + item.TII) / 5;    const
instQualityAvg = (item.COC + item.GE + item.PSTAB + item.RQ + item.RL +
item.VA) / 6;        countryAverages[item.country].CPI += item.CPI;
countryAverages[item.country].eGovIndex += eGovAvg;
countryAverages[item.country].instQuality += instQualityAvg;
countryAverages[item.country].count++;  }    // Calculate final averages  for
(const country in countryAverages) {    countryAverages[country].CPI /=
countryAverages[country].count;    countryAverages[country].eGovIndex /=
countryAverages[country].count;    countryAverages[country].instQuality /=
countryAverages[country].count;  }    return countryAverages;} // Calculate
correlation between variablesfunction calculateCorrelation(x, y) {  const n =
x.length;  let sumX = 0, sumY = 0, sumXY = 0, sumX2 = 0, sumY2 = 0;    for
(let i = 0; i < n; i++) {    sumX += x[i];    sumY += y[i];    sumXY += x[i]
* y[i];    sumX2 += x[i] * x[i];    sumY2 += y[i] * y[i];  }    const
numerator = n * sumXY - sumX * sumY;  const denominator = Math.sqrt((n *
sumX2 - sumX * sumX) * (n * sumY2 - sumY * sumY));    return numerator /
denominator;} // Analyze correlations between E-governance, Institutional
Quality, and CPIfunction analyzeRelationships(data) {  const eGovScores = [];
  const instQualityScores = [];  const cpiScores = [];    for (const item of
data) {
```

```
          const eGovAvg = (item.EGI + item.EPI + item.OSI + item.HCI +
item.TII) / 5;    const instQualityAvg = (item.COC + item.GE + item.PSTAB +
item.RQ + item.RL + item.VA) / 6;        eGovScores.push(eGovAvg);
instQualityScores.push(instQualityAvg);    cpiScores.push(item.CPI);  }
const corr_eGov_CPI = calculateCorrelation(eGovScores, cpiScores);  const
corr_instQuality_CPI = calculateCorrelation(instQualityScores, cpiScores);
const corr_eGov_instQuality = calculateCorrelation(eGovScores,
instQualityScores);    return {    corr_eGov_CPI,    corr_instQuality_CPI,
corr_eGov_instQuality  };} // Perform a simple mediation analysisfunction
mediationAnalysis(data) {  // Step 1: Regress CPI on E-governance (c path)
  // Step 2: Regress Institutional Quality on E-governance (a path)  // Step
3: Regress CPI on both E-governance and Institutional Quality (b and c' paths)
    // For simplicity, we'll estimate these relationships using correlations
const relationships = analyzeRelationships(data);    console.log("Mediation
Analysis:");  console.log("Path a (E-Gov !’ Inst Quality): ",
relationships.corr_eGov_instQuality.toFixed(4));  console.log("Path b (Inst
Quality !’ CPI): ", relationships.corr_instQuality_CPI.toFixed(4));
console.log("Path c (E-Gov !’ CPI): ", relationships.corr_eGov_CPI.toFixed(4));
    // Sobel test (simplified)  const mediationEffect =
relationships.corr_eGov_instQuality * relationships.corr_instQuality_CPI;
console.log("Estimated Mediation Effect: ", mediationEffect.toFixed(4));} //
Run the analysesconst averages = calculateAverages(sampleData);
console.log("Country Averages (sample of 5 countries):");const
sampleCountries = Object.keys(averages).slice(0, 5);for (const country of
sampleCountries) {  console.log(`${country}: CPI =
${averages[country].CPI.toFixed(2)}, E-Gov =
${averages[country].eGovIndex.toFixed(2)}, Inst Quality =
${averages[country].instQuality.toFixed(2)}`);} console.log("\nCorrelation
Analysis:");const relationships = analyzeRelationships(sampleData);
console.log("Correlation between E-Governance and CPI: ",
relationships.corr_eGov_CPI.toFixed(4));console.log("Correlation between
Institutional Quality and CPI: ",
relationships.corr_instQuality_CPI.toFixed(4));console.log("Correlation
between E-Governance and Institutional Quality: ",
relationships.corr_eGov_instQuality.toFixed(4));
// Mediation analysismediationAnalysis(sampleData);
// Now let's prepare data for visualizationfunction
prepareVisualizationData() {  const visualData = {    countries: [],    eGov:
[],    instQuality: [],    CPI: []  };    for (const country in averages) {
    visualData.countries.push(country);
visualData.eGov.push(averages[country].eGovIndex);
visualData.instQuality.push(averages[country].instQuality);
visualData.CPI.push(averages[country].CPI);  }    return visualData;} const
visualizationData = prepareVisualizationData();console.log("Visualization
Data (first 5 countries):");for (let i = 0; i < 5; i++) {
console.log(`${visualizationData.countries[i]}: E-Gov =
${visualizationData.eGov[i].toFixed(2)}, Inst Quality =
${visualizationData.instQuality[i].toFixed(2)}, CPI =
${visualizationData.CPI[i].toFixed(2)}`);} // Function to simulate regression
modelsfunction simulateRegressionModels() {  console.log("\nSimulated
Regression Models:");    // Model 1: CPI = ;#  ² ²1*EGov +cp¬st beta0_m1 =
40; // Intercept  const beta1_m1 = 15; // EGov coefficient
console.log("Model 1 (CPI ~ E-Gov): CPI = " + beta0_m1 + " + " + beta1_m1 + "
* E-Gov");    // Model 2: InstQuality = ;#  ² ²1*EGov +cp¬st beta0_m2 =
-0.5; // Intercept  const beta1_m2 = 1.2; // EGov coefficient
console.log("Model 2 (Inst Quality ~ E-Gov): Inst Quality = " + beta0_m2 + "
+ " + beta1_m2 + " * E-Gov");    // Model 3: CPI = ;#  ² ²1*EGov + ;#"¤–ç7E V Æ—G’
+ ;Pconst beta0_m3 = 35; // Intercept  const beta1_m3 = 8; // EGov
coefficient (direct effect)  const beta2_m3 = 6; // InstQuality coefficient
console.log("Model 3 (CPI ~ E-Gov + Inst Quality): CPI = " + beta0_m3 + " + "
+ beta1_m3 + " * E-Gov + " + beta2_m3 + " * Inst Quality");
```

```
                                                                    // Calculate
indirect effect  const indirectEffect = beta1_m2 * beta2_m3;
console.log("Indirect effect (E-Gov !’ Inst Quality !’ CPI): " +
indirectEffect.toFixed(2));  console.log("Total effect (Direct + Indirect): "
+ (beta1_m3 + indirectEffect).toFixed(2));} simulateRegressionModels();
```

## deployment-script.js

```
// scripts/deploy_ethereum.jsÐconst Web3 = require('web3');Ðconst fs =
require('fs');Ðconst path = require('path');Ðconst { performance } =
require('perf_hooks');Ðasync function deployContracts() {Ð try {Ð
console.log('Starting deployment to Ethereum...');Ð   const startTime =
performance.now();Ð   // Initialize web3 with providerÐ   const web3 = new
Web3(process.env.ETHEREUM_RPC_URL);Ð   Ð   // Load ALL contract artifactsÐ
const contractArtifacts = {Ð     BlockchainRegistryBase: require('../build/
contracts/BlockchainRegistryBase.json'),Ð     BlockchainRegistry: require('../
build/contracts/BlockchainRegistry.json'),Ð     BlockchainMonitor:
require('../build/contracts/BlockchainMonitor.json'),Ð     ChaCha20Poly1305:
require('../build/contracts/ChaCha20Poly1305.json'),Ð     MetadataParser:
require('../build/contracts/MetadataParser.json'),Ð     PacechainChannel:
require('../build/contracts/PacechainChannel.json'),Ð
SpeculativeTransactionHandler: require('../build/contracts/
SpeculativeTransactionHandler.json'),Ð     ConfidenceScoreCalculator:
require('../build/contracts/ConfidenceScoreCalculator.json'),Ð
AssetTransferProcessor: require('../build/contracts/
AssetTransferProcessor.json'),Ð     ZKPVerifierBase: require('../build/
contracts/ZKPVerifierBase.json'),Ð     ProofGenerator: require('../build/
contracts/ProofGenerator.json'),Ð     TransactionValidator: require('../build/
contracts/TransactionValidator.json'),Ð     ClusterManager: require('../build/
contracts/ClusterManager.json'),Ð     ClusterCommunication: require('../build/
contracts/ClusterCommunication.json'),Ð     RewardBase: require('../build/
contracts/RewardBase.json'),Ð     RewardCalculator: require('../build/
contracts/RewardCalculator.json'),Ð     RewardDistributor: require('../build/
contracts/RewardDistributor.json'),Ð     RewardToken: require('../build/
contracts/RewardToken.json'),Ð     ProofOfStakeValidator: require('../build/
contracts/ProofOfStakeValidator.json'),Ð     StateManager: require('../build/
contracts/StateManager.json'),Ð     TransactionRelay: require('../build/
contracts/TransactionRelay.json'),Ð     RelayChain: require('../build/
contracts/RelayChain.json'),Ð     ReceivingBlockchainInterface: require('../
build/contracts/ReceivingBlockchainInterface.json')Ð   };Ð   // Deployment
object to store deployed contract addressesÐ   const deployedContracts = {};Ð
    // Deploy contracts sequentiallyÐ   for (const [contractName, artifact]
of Object.entries(contractArtifacts)) {Ð     try {Ð
console.log(`Deploying ${contractName}...`);Ð       const deployedContract =
await deployContract(web3, artifact);Ð       deployedContracts[contractName]
= deployedContract.options.address;Ð     } catch (deployError) {Ð
console.error(`Failed to deploy ${contractName}:`, deployError);Ð       //
Optionally, you can choose to continue or stop deploymentÐ       // throw
deployError; // Uncomment to stop on first deployment failureÐ     }Ð   }Ð
const endTime = performance.now();Ð   const deploymentTime = endTime -
startTime;Ð   // Save deployment informationÐ   const deploymentInfo = {Ð
networkId: await web3.eth.net.getId(),Ð     deploymentTime,Ð     contracts:
deployedContracts,Ð     timestamp: new Date().toISOString()Ð   };Ð   // Save
deployment info to fileÐ   const deploymentPath = path.join(__dirname, '../
config/deployment.json');Ð   fs.writeFileSync(deploymentPath,
JSON.stringify(deploymentInfo, null, 2));Ð   console.log('Deployment
completed successfully!');Ð   console.log(`Total deployment time:
${deploymentTime}ms`);Ð   console.log('Deployment info saved to:',
deploymentPath);Ð   return deploymentInfo;Ð } catch (error) {Ð
console.error('Deployment failed:', error);Ð   process.exit(1);Ð }ÐÐasync
function deployContract(web3, artifact) {Ð // Ensure the artifact has the
required propertiesÐ if (!artifact.abi || !artifact.bytecode) {Ð   throw new
Error('Invalid contract artifact');Ð }Ð const accounts = await
web3.eth.getAccounts();Ð const account = accounts[0];Ð Ð const contract = new
web3.eth.Contract(artifact.abi);Ð const deploy = contract.deploy({Ð   data:
artifact.bytecode,Ð   arguments: [] // Add constructor arguments if neededÐ
  });Ð
```

```
    Ð const gasEstimate = await deploy.estimateGas();Ð const gasPrice =
await web3.eth.getGasPrice();Ð Ð const deployed = await deploy.send({Ð
from: account,Ð   gas: gasEstimate,Ð   gasPrice: gasPriceÐ });Ð
console.log(`Contract deployed at: ${deployed.options.address}`);Ð return
deployed;ÐÐ}ÐÐif (require.main === module) {Ð deployContracts()Ð   .then(() =>
process.exit(0))Ð   .catch(error => {Ð     console.error(error);Ð
process.exit(1);Ð   });ÐÐmodule.exports = deployContracts;Ð }
```

## migrations\10_initial_migration.js

```
const UncertaintyAnalytics = artifacts.require("UncertaintyAnalytics");Ðconst
RequestManager = artifacts.require("RequestManager");ÐÐmodule.exports =
function(deployer) {Ð deployer.deploy(RequestManager,
UncertaintyAnalytics.address);Ð};
```

## migrations\11_initial_migration.js

```
const CityRegister = artifacts.require("CityRegister");Ðconst CompanyRegister
= artifacts.require("CompanyRegister");Ðconst CityEmissions =
artifacts.require("CityEmissionsContract");Ðconst RenewalTheory =
artifacts.require("RenewalTheoryContract");Ðconst CityHealth =
artifacts.require("CityHealthCalculator");Ðconst TemperatureRenewal =
artifacts.require("TemperatureRenewalContract");Ðconst ClimateReduction =
artifacts.require("ClimateReductionContract");Ðconst Mitigation =
artifacts.require("MitigationContract");Ðconst CarbonCreditMarket =
artifacts.require("CarbonCreditMarket");Ðmodule.exports = async
function(deployer, network, accounts) {Ð try {Ð   // Network-specific
configurationÐ   const networkConfig = {Ð     development: {Ð
uniswapRouter: accounts[8],Ð       carbonToken: accounts[7],Ð       usdToken:
accounts[6],Ð       carbonFeed: accounts[4],Ð       tempFeed: accounts[3]Ð
      },Ð     mainnet: {Ð       // Replace with actual mainnet addressesÐ
        uniswapRouter: '0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D', //
Uniswap V2 RouterÐ       carbonToken:
'0x0000000000000000000000000000000000000000',Ð       usdToken:
'0x0000000000000000000000000000000000000000',Ð       carbonFeed:
'0x0000000000000000000000000000000000000000',Ð       tempFeed:
'0x0000000000000000000000000000000000000000'Ð     },Ð     ropsten: {Ð
        // Replace with Ropsten testnet addresses if differentÐ
uniswapRouter: '0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D',Ð
carbonToken: '0x0000000000000000000000000000000000000000',Ð       usdToken:
'0x0000000000000000000000000000000000000000',Ð       carbonFeed:
'0x0000000000000000000000000000000000000000',Ð       tempFeed:
'0x0000000000000000000000000000000000000000'Ð     }Ð   };Ð   // Select
network configuration, default to development if not specifiedÐ   const
config = networkConfig[network] || networkConfig.development;Ð   // Fallback
addresses if network config is incompleteÐ   const getFallbackAddress =
(address, fallbackIndex) => Ð     address && address !==
'0x0000000000000000000000000000000000000000' Ð       ? address Ð       :
accounts[fallbackIndex];Ð   // Deploy CityRegister with explicit constructor
parametersÐ   const cityRegister = await CityRegister.new("RPSTOKENS", "RPS");Ð
    console.log('CityRegister deployed at:', cityRegister.address);Ð   //
Deploy CompanyRegister with explicit constructor parametersÐ   const
companyRegister = await CompanyRegister.new("RPSTOKENS", "RPS");Ð
console.log('CompanyRegister deployed at:', companyRegister.address);Ð   //
Deploy CityEmissionsÐ   const cityEmissions = await CityEmissions.new();Ð
console.log('CityEmissions deployed at:', cityEmissions.address);Ð   //
Deploy RenewalTheoryÐ   const renewalTheory = await RenewalTheory.new();Ð
console.log('RenewalTheory deployed at:', renewalTheory.address);Ð   //
Prepare addresses with fallbackÐ   const uniswapRouter =
getFallbackAddress(config.uniswapRouter, 8);Ð   const carbonToken =
getFallbackAddress(config.carbonToken, 7);Ð   const usdToken =
getFallbackAddress(config.usdToken, 6);Ð   // Deploy CarbonCreditMarketÐ
const carbonCreditMarket = await CarbonCreditMarket.new(Ð     uniswapRouter,Ð
      carbonToken,Ð     usdTokenÐ   );Ð   console.log('CarbonCreditMarket
deployed at:', carbonCreditMarket.address);Ð   // Deploy CityHealthÐ   const
cityHealth = await CityHealth.new();Ð   console.log('CityHealth deployed
at:', cityHealth.address);Ð   // Deploy TemperatureRenewalÐ   const
temperatureRenewal = await TemperatureRenewal.new();Ð
console.log('TemperatureRenewal deployed at:', temperatureRenewal.address);Ð
    // Deploy ClimateReductionÐ   const climateReduction = await
ClimateReduction.new(carbonCreditMarket.address);Ð
console.log('ClimateReduction deployed at:', climateReduction.address);Ð   //
Prepare Chainlink feed addresses with fallbackÐ   const carbonFeed =
getFallbackAddress(config.carbonFeed, 4);Ð   const tempFeed =
getFallbackAddress(config.tempFeed, 3);Ð   // Deploy MitigationÐ   const
mitigation = await Mitigation.new(carbonFeed, tempFeed);Ð
```

```
console.log('Mitigation deployed at:', mitigation.address);Ð   // Optional: Log network being deployed toÐ   console.log(`Deployment completed for network: ${network}`);Ð } catch (error) {Ð   console.error('Deployment error:', error);Ð   throw error;Ð }Þ;
```

## migrations\12_initial_migration.js

```
const BCADN = artifacts.require("BCADN");const ProactiveDefenseMechanism =
artifacts.require("ProactiveDefenseMechanism");
// Helper function to add delayconst delay = (ms) => new Promise(resolve =>
setTimeout(resolve, ms));                          module.exports = async
function(deployer, network, accounts) {  try {    // Deploy BCADN contract
    await deployer.deploy(BCADN);    await delay(5000);  // Add a 5-second
delay between deployments        // Deploy Proactive Defense Mechanism
await deployer.deploy(      ProactiveDefenseMechanism,       30,    //
anomalyThreshold      86400  // baselineUpdateInterval (1 day in seconds)
    );        console.log(`Deployment completed for network: ${network}`);  }
catch (error) {    console.error('Deployment error:', error);    throw error;
  }};  // Etherscan https://sepolia.etherscan.io/
tx/0x3326b4c25c1ae0b7d6db118e86c095f30c9caf052430043ed62955b74abc49c6//
https://sepolia.etherscan.io/block/8118449#consensusinfo
```

## migrations\13_initial_migration.js

```
const NIDRegistry = artifacts.require("NIDRegistry");const NIASRegistry =
artifacts.require("NIASRegistry");const ABATLTranslation =
artifacts.require("ABATLTranslation");const SequencePathRouter =
artifacts.require("SequencePathRouter");const ClusteringContract =
artifacts.require("ClusteringContract");const fs = require('fs');const path =
require('path');                module.exports = async function(deployer,
network, accounts) {  console.log("Deploying N2N routing system
contracts...");    // Deploy NIDRegistry  console.log("Deploying
NIDRegistry...");  await deployer.deploy(NIDRegistry);  const nidRegistry =
await NIDRegistry.deployed();  console.log("NIDRegistry deployed at:",
nidRegistry.address);    // Deploy NIASRegistry  console.log("Deploying
NIASRegistry...");  await deployer.deploy(NIASRegistry);  const niasRegistry
= await NIASRegistry.deployed();  console.log("NIASRegistry deployed at:",
niasRegistry.address);    // Deploy ABATLTranslation  console.log("Deploying
ABATLTranslation...");  await deployer.deploy(ABATLTranslation,
nidRegistry.address, niasRegistry.address);  const abatlTranslation = await
ABATLTranslation.deployed();  console.log("ABATLTranslation deployed at:",
abatlTranslation.address);    // Deploy SequencePathRouter
console.log("Deploying SequencePathRouter...");  await deployer.deploy(
SequencePathRouter,     nidRegistry.address,     niasRegistry.address,
abatlTranslation.address  );  const sequencePathRouter = await
SequencePathRouter.deployed();  console.log("SequencePathRouter deployed
at:", sequencePathRouter.address);    // Deploy ClusteringContract
console.log("Deploying ClusteringContract...");  await deployer.deploy(
ClusteringContract,    nidRegistry.address,    niasRegistry.address,
abatlTranslation.address  );  const clusteringContract = await
ClusteringContract.deployed();  console.log("ClusteringContract deployed
at:", clusteringContract.address);    // Save deployed contract addresses to
file  const addresses = {    NIDRegistry: nidRegistry.address,
NIASRegistry: niasRegistry.address,    ABATLTranslation:
abatlTranslation.address,    SequencePathRouter: sequencePathRouter.address,
    ClusteringContract: clusteringContract.address  };    const addressesFile
= path.join(__dirname, '../contract_addresses.json');
fs.writeFileSync(addressesFile, JSON.stringify(addresses, null, 2));
console.log("Contract addresses saved to:", addressesFile);
console.log("All contracts deployed successfully!");};
```

## migrations\1_initial_migration.js

```
//const TransactionValidator = artifacts.require("TransactionValidator");Ð
const BlockchainRegistryBase = artifacts.require("BlockchainRegistryBase");Ð
const BlockchainRegistry = artifacts.require("BlockchainRegistry");Ðconst
BlockchainMonitor = artifacts.require("BlockchainMonitor");Ð// Add other
contract imports as neededÐmodule.exports = function(deployer) {Ð // Deploy
contracts in orderÐ deployer.deploy(BlockchainRegistryBase);Ð
deployer.deploy(BlockchainRegistry);Ð deployer.deploy(BlockchainMonitor);Ð};
```

## migrations\2_deploy_contracts.js

```
const ChaCha20Poly1305 = artifacts.require("ChaCha20Poly1305");Ð// Add other
contract imports as neededÐmodule.exports = function(deployer) {Ð // Deploy
contracts in orderÐ deployer.deploy(ChaCha20Poly1305);Ð};
```

## migrations\3_initial_migration.js

```
const MetadataParser = artifacts.require("MetadataParser");Ð // Add other
contract imports as neededÐmodule.exports = function(deployer) {Ð // Deploy
contracts in orderÐ deployer.deploy(MetadataParser);Ð};
```

## migrations\4_initial_migration.js

```
const PacechainChannel = artifacts.require("PacechainChannel");Ð
SpeculativeTransactionHandler =
artifacts.require("SpeculativeTransactionHandler");Ð
ConfidenceScoreCalculator = artifacts.require("ConfidenceScoreCalculator");Ð
const AssetTransferProcessor = artifacts.require("AssetTransferProcessor");Ð//
Add other contract imports as neededÐmodule.exports = function(deployer) {Ð //
Deploy contracts in orderÐ deployer.deploy(PacechainChannel);Ð
deployer.deploy(SpeculativeTransactionHandler);Ð
deployer.deploy(ConfidenceScoreCalculator);Ð
deployer.deploy(AssetTransferProcessor);Ð};
```

## migrations\5_initial_migration.js

```
Ðconst TransactionValidator = artifacts.require("TransactionValidator");Ð// Add
other contract imports as neededÐmodule.exports = function(deployer) {Ð //
Deploy contracts in orderÐ deployer.deploy(TransactionValidator);Ð};
```

## migrations\6_initial_migration.js

```
const RewardDistributor = artifacts.require("RewardDistributor");Ð
RewardToken = artifacts.require("RewardToken");Ðmodule.exports =
function(deployer, network, accounts) {Ð // Deploy the RewardToken firstÐ
deployer.deploy(RewardToken).then(() => {Ð   // Then deploy the
RewardDistributor with the RewardToken addressÐ   return
deployer.deploy(RewardDistributor, RewardToken.address);Ð });Ð};
```

## migrations\7_initial_migration.js

```
const ProofOfStakeValidator = artifacts.require("ProofOfStakeValidator");Ð
const StateManager = artifacts.require("StateManager");Ðconst TransactionRelay
= artifacts.require("TransactionRelay");Ðconst ReceivingBlockchainInterface =
artifacts.require("ReceivingBlockchainInterface");Ð// Add other contract
imports as neededÐmodule.exports = function(deployer) {Ð // Deploy contracts
in orderÐ deployer.deploy(ProofOfStakeValidator);Ð
deployer.deploy(StateManager);Ð deployer.deploy(TransactionRelay);Ð
deployer.deploy(ReceivingBlockchainInterface);Ð};
```

## migrations\8_initial_migration.js

```
const UncertaintyAnalytics = artifacts.require("UncertaintyAnalytics");Ð
module.exports = function(deployer) {Ð deployer.deploy(UncertaintyAnalytics);Ð
};
```

## migrations\9_initial_migration.js

```
const UncertaintyAnalytics = artifacts.require("UncertaintyAnalytics");Ðconst
ResponseManager = artifacts.require("ResponseManager");Ðmodule.exports =
function(deployer) {Ð deployer.deploy(ResponseManager,
UncertaintyAnalytics.address);Ð};
```

## next.config.js

```
/** @type {import('next').NextConfig} */Ðconst nextConfig = {Ð
reactStrictMode: true,Ð Ð // Build optimizationÐ swcMinify: true,Ð
optimizeFonts: true,Ð productionBrowserSourceMaps: false,Ð // TypeScript
configurationÐ typescript: {Ð   ignoreBuildErrors: false,Ð },Ð // Webpack
configurationÐ webpack: (config, { isServer }) => {Ð   // Optimize source
mappingÐ   config.devtool = isServer ? false : 'cheap-source-map';Ð   //
Fallback configurations for browser compatibilityÐ   if (!isServer) {Ð
config.resolve.fallback = {Ð       ...config.resolve.fallback,Ð       fs:
false,Ð       net: false,Ð       tls: false,Ð       crypto:
require.resolve('crypto-browserify'),Ð       stream: require.resolve('stream-
browserify'),Ð       http: require.resolve('stream-http'),Ð       https:
require.resolve('https-browserify'),Ð       os: require.resolve('os-
browserify/browser'),Ð       zlib: require.resolve('browserify-zlib'),Ð     };Ð
    }Ð   // Code splitting and optimizationÐ   config.optimization = {Ð
      ...config.optimization,Ð     splitChunks: {Ð       chunks: 'all',Ð
        minSize: 20000,Ð       maxSize: 250000,Ð       minChunks: 1,Ð
maxAsyncRequests: 30,Ð       maxInitialRequests: 30,Ð
automaticNameDelimiter: '~',Ð       cacheGroups: {Ð         defaultVendors: {Ð
            test: /[\\/]node_modules[\\/]/,Ð           priority: -10,Ð
            reuseExistingChunk: true,Ð         },Ð         default: {Ð
            minChunks: 2,Ð           priority: -20,Ð
reuseExistingChunk: true,Ð         },Ð       },Ð     },Ð   };Ð   // Reduce
build noiseÐ   config.stats = 'minimal';Ð   // Bundle analyzer configurationÐ
    if (process.env.ANALYZE) {Ð     const { BundleAnalyzerPlugin } =
require('@next/bundle-analyzer');Ð     config.plugins.push(Ð       new
BundleAnalyzerPlugin({Ð         analyzerMode: 'server',Ð
analyzerPort: isServer ? 8888 : 8889,Ð         openAnalyzer: true,Ð       })Ð
      );Ð   }Ð   return config;Ð },Ð // Experimental featuresÐ experimental: {Ð
    optimizePackageImports: ['react', 'react-dom', 'next'],Ð
optimisticClientCache: true,Ð },Ð // Performance monitoringÐ onDemandEntries:
{Ð   maxInactiveAge: 30 * 1000,Ð   pagesBufferLength: 2,Ð },Ð // Image
optimizationÐ images: {Ð   deviceSizes: [640, 750, 828, 1080, 1200, 1920,
2048, 3840],Ð   imageSizes: [16, 32, 48, 64, 96, 128, 256, 384],Ð   domains:
[],Ð   formats: ['image/avif', 'image/webp'],Ð },Ð // Compiler options for
productionÐ compiler: {Ð   removeConsole: process.env.NODE_ENV ===
'production',Ð   reactRemoveProperties: process.env.NODE_ENV === 'production',Ð
  },Ð  // Add environment variablesÐ  env: {Ð   ETHEREUM_PROVIDER_URL:
process.env.ETHEREUM_PROVIDER_URL,Ð   POLKADOT_PROVIDER_URL:
process.env.POLKADOT_PROVIDER_URL,Ð   NEXT_PUBLIC_TATUM_API_KEY:
process.env.NEXT_PUBLIC_TATUM_API_KEY,Ð   METADATA_PARSER_ADDRESS:
process.env.METADATA_PARSER_ADDRESS,Ð   PACECHAIN_CHANNEL_ADDRESS:
process.env.PACECHAIN_CHANNEL_ADDRESS,Ð   RELAY_CHAIN_ADDRESS:
process.env.RELAY_CHAIN_ADDRESSÐ },Ð};Ðmodule.exports = nextConfig;
```

## scripts\deploy_ethereum.js

```
require('dotenv').config({ path: '.env.local' });Ðconst Web3 = require('web3');Ð
const axios = require('axios');Ðconst fs = require('fs');Ðconst path =
require('path');Ðconst glob = require('glob');ÐÐ// Environment variablesÐconst
NEXT_PUBLIC_TATUM_API_KEY = process.env.NEXT_PUBLIC_TATUM_API_KEY;Ðconst
ETHERSCAN_API_KEY = process.env.ETHERSCAN_API_KEY;Ðconst PRIVATE_KEY =
process.env.PRIVATE_KEY;Ðconst INFURA_URL = process.env.ETHEREUM_PROVIDER_URL;Ð
const CHAIN_ID = 11155111; // Sepolia chain IDÐconst CHAIN = 'ethereum-
sepolia'; // This corresponds to the {CHAIN} part in the URLÐÐ// Providers
configurationÐconst providers = [Ð {Ð   url: INFURA_URL,Ð   name: 'Infura',Ð
chainId: CHAIN_ID,Ð   minBalance: '0.1',Ð   retryAttempts: 3,Ð   timeout:
30000,Ð },Ð {Ð   url: `https://x-api-key:${NEXT_PUBLIC_TATUM_API_KEY}
@${CHAIN}.gateway.tatum.io`,Ð   name: 'Tatum',Ð   chainId: CHAIN_ID,Ð
minBalance: '0.1',Ð   retryAttempts: 3,Ð   timeout: 30000,Ð },Ð];ÐÐ// Test Tatum
connectionÐasync function testTatumConnection() {Ð try {Ð   const response =
await axios.post(Ð     `https://x-api-key:${NEXT_PUBLIC_TATUM_API_KEY}
@${CHAIN}.gateway.tatum.io`,Ð     {Ð       jsonrpc: '2.0',Ð       method:
'eth_blockNumber',Ð       params: [],Ð       id: 1,Ð     },Ð     {Ð
headers: {Ð         'Content-Type': 'application/json',Ð       },Ð     }Ð   );Ð
    if (response.data && response.data.result) {Ð     console.log('Tatum
connection successful. Latest block number:', parseInt(response.data.result,
16));Ð     return true;Ð   } else {Ð     console.log('Tatum connection
failed. Unexpected response:', response.data);Ð     return false;Ð   }Ð }
catch (error) {Ð   console.error('Tatum connection test failed:',
error.message);Ð   return false;Ð }ÐÐ// Get Web3 ProviderÐasync function
getWeb3Provider() {Ð if (!PRIVATE_KEY) {Ð   throw new Error('Private key not
found in environment variables');Ð }Ð for (const provider of providers) {Ð
try {Ð     console.log(`Attempting to connect to ${provider.name} with URL:
${provider.url}`);Ð     let web3Provider;Ð     if (provider.name === 'Tatum')
{Ð       const tatumConnected = await testTatumConnection();Ð       if (!
tatumConnected) {Ð         console.log('Skipping Tatum due to failed
connection test');Ð         continue;Ð       }Ð     }Ð     web3Provider = new
Web3.providers.HttpProvider(provider.url);Ð     const web3 = new
Web3(web3Provider);Ð     // Add the account using the private keyÐ     const
account = web3.eth.accounts.privateKeyToAccount(PRIVATE_KEY);Ð
web3.eth.accounts.wallet.add(account);Ð     web3.eth.defaultAccount =
account.address;Ð     // Test provider connectivityÐ     await
web3.eth.getBlockNumber();Ð     const balance = await
web3.eth.getBalance(account.address);Ð     const balanceInEth =
web3.utils.fromWei(balance, 'ether');Ð     console.log(`Connected to
${provider.name}. Account balance: ${balanceInEth} ETH`);Ð     if
(parseFloat(balanceInEth) >= parseFloat(provider.minBalance)) {Ð       return
{ web3, provider: web3Provider };Ð     }Ð     console.log(`Insufficient
balance on ${provider.name}. Trying next provider...`);Ð   } catch (error) {Ð
      console.error(`Failed to connect to ${provider.name}:`, error.message);Ð
    }Ð }Ð throw new Error('No viable provider found');ÐÐasync function
retryOperation(operation, maxRetries = 3, delay = 2000) {Ð let lastError;Ð
for (let i = 0; i < maxRetries; i++) {Ð   try {Ð     return await operation();Ð
    } catch (error) {Ð     lastError = error;Ð     console.log(`Attempt ${i +
1} failed: ${error.message}`);Ð     if (i === maxRetries - 1) {Ð
console.error(`All ${maxRetries} attempts failed. Last error:`, error);Ð
        throw error;Ð     }Ð     console.log(`Retrying in ${delay / 1000}
seconds...`);Ð     await new Promise((resolve) => setTimeout(resolve, delay));Ð
    }Ð }ÐÐasync function verifyContract(contractAddress, contractName,
contractSource, compilerVersion = 'v0.8.19') {Ð if (!ETHERSCAN_API_KEY) {Ð
throw new Error('ETHERSCAN_API_KEY not found in environment variables');Ð }Ð
try {Ð   const response = await axios.post('https://api-sepolia.etherscan.io/
api', null, {Ð     params: {Ð       module: 'contract',Ð       action:
'verifysourcecode',Ð       apikey: ETHERSCAN_API_KEY,Ð       contractaddress:
contractAddress,Ð
```

```
                        sourceCode: contractSource,Ð       codeformat:
'solidity-single-file',Ð       contractname: contractName,Ð
compilerversion: compilerVersion,Ð       optimizationUsed: 1,Ð     },Ð   });Ð
    if (response.data.status !== '1') {Ð     throw new Error(`Contract
verification failed: ${response.data.result}`);Ð   }Ð   return response.data;Ð
  } catch (error) {Ð   console.error(`Verification error for
${contractName}:`, error);Ð   throw error;Ð }}Ðasync function
deployContract(contractName, abi, bytecode) {Ð return retryOperation(async ()
=> {Ð   const { web3, provider } = await getWeb3Provider();Ð   const accounts
= await web3.eth.getAccounts();Ð   try {Ð     const contract = new
web3.eth.Contract(abi);Ð     console.log(`Estimating gas for
${contractName}...`);Ð     const gasEstimate = await contract.deploy({ data:
bytecode }).estimateGas({ from: accounts[0] });Ð     const gasPrice = await
web3.eth.getGasPrice();Ð     const totalCost =
web3.utils.fromWei((BigInt(gasEstimate) * BigInt(gasPrice)).toString(),
'ether');Ð     console.log(`Estimated deployment cost for ${contractName}:
${totalCost} ETH`);Ð     console.log(`Deploying ${contractName}...`);Ð
const deployedContract = await contract.deploy({ data:
bytecode }).send({ from: accounts[0], gas: Math.floor(gasEstimate * 1.2) });Ð
      console.log(`${contractName} deployed to:`,
deployedContract.options.address);Ð     return { address:
deployedContract.options.address, transactionHash:
deployedContract.transactionHash };Ð   } catch (error) {Ð
console.error(`Deployment error for ${contractName}:`, error);Ð     throw
error;Ð   } finally {Ð     if (provider && provider.engine) {Ð       await
new Promise((resolve) => {Ð         provider.engine.stop();Ð
resolve();Ð       });Ð     }Ð   }Ð }, 3, 5000);}Ðasync function
deployAllContracts() {Ð console.log('Starting contract deployment...');Ð
const contractsDir = path.join(__dirname, '../contracts');Ð const
buildContractsDir = path.join(__dirname, '../build/contracts');Ð const
addressesPath = path.join(__dirname, '../config/contract_addresses.json');Ð
  // Ensure directories existÐ if (!fs.existsSync(buildContractsDir)) {Ð
throw new Error('Build directory not found. Please compile contracts first.');Ð
  }Ð // Read or create addresses fileÐ let existingAddresses = {};Ð try {Ð
existingAddresses = JSON.parse(fs.readFileSync(addressesPath, 'utf8'));Ð }
catch (error) {Ð   console.log('No existing addresses file found. Creating
new one.');Ð }Ð const deployedContracts = {};Ð const contractFiles =
fs.readdirSync(buildContractsDir).filter((f) => f.endsWith('.json') && !
f.includes('Metadata'));Ð for (const file of contractFiles) {Ð   const
contractName = path.basename(file, '.json');Ð   console.log(`Processing
${contractName}...`);Ð   try {Ð     const contractPath =
path.join(buildContractsDir, file);Ð     const contractData =
JSON.parse(fs.readFileSync(contractPath, 'utf8'));Ð     const existingAddress
= existingAddresses[contractName]?.address;Ð     const deploymentResult =
await deployContract(contractName, contractData.abi, contractData.bytecode);Ð
      const sourceCodePath = path.join(contractsDir, `**/
${contractName}.sol`);Ð     const sourceCodes = glob.sync(sourceCodePath);Ð
      if (sourceCodes.length > 0) {Ð       const sourceCode =
fs.readFileSync(sourceCodes[0], 'utf8');Ð       console.log(`Verifying
${contractName}...`);Ð       const verificationResult = await
verifyContract(deploymentResult.address, contractName, sourceCode);Ð
deployedContracts[contractName] = {Ð         address:
deploymentResult.address,Ð         abi: `${contractName}.json`,Ð
transactionHash: deploymentResult.transactionHash,Ð
verificationStatus: verificationResult,Ð         previousAddress:
existingAddress,Ð       };Ð       console.log(`${contractName} deployed and
verified successfully`);Ð     }Ð   } catch (error) {Ð
console.error(`Failed to deploy and verify ${contractName}:`, error);Ð     //
Continue with other contracts even if one failsÐ   }Ð }Ð const
updatedAddresses = {Ð
```

```
                        ...existingAddresses,Ð   ...Object.fromEntries(Ð
Object.entries(deployedContracts).map(([name, contract]) => [Ð       name,Ð
        { address: contract.address, abi: contract.abi, previousAddress:
contract.previousAddress },     ]),Ð     ),Ð   };Ð   // Ensure config
directory existsÐ   const configDir = path.dirname(addressesPath);Ð   if (!
fs.existsSync(configDir)) {Ð     fs.mkdirSync(configDir, { recursive: true });Ð
    }Ð   fs.writeFileSync(addressesPath, JSON.stringify(updatedAddresses,
null, 2));Ð   console.log('Contracts deployed and addresses updated:',
deployedContracts);Ð   return deployedContracts;Ð }Ð Ð // Main executionÐ if
(require.main === module) {Ð   // Process-level unhandled rejection handlerÐ
    process.on('unhandledRejection', (error) => {Ð
console.error('Unhandled promise rejection:', error);Ð     process.exit(1);Ð
    });Ð Ð   // Validate required environment variablesÐ   const
requiredEnvVars = {Ð     'NEXT_PUBLIC_TATUM_API_KEY':
NEXT_PUBLIC_TATUM_API_KEY,Ð     'PRIVATE_KEY': PRIVATE_KEY,Ð
'ETHERSCAN_API_KEY': ETHERSCAN_API_KEY,Ð   };Ð   const missingEnvVars =
Object.entries(requiredEnvVars).filter(([_, value]) => !value).map(([key]) =>
key);Ð   if (missingEnvVars.length > 0) {Ð     console.error('Missing
required environment variables:', missingEnvVars.join(', '));Ð
process.exit(1);Ð   }Ð Ð   // Add timeout for the entire deployment processÐ
    const deploymentTimeout = setTimeout(() => {Ð
console.error('Deployment timed out after 5 minutes');Ð     process.exit(1);Ð
    }, 300000); // 5 minutesÐ Ð   testTatumConnection()Ð     .then((result)
=> {Ð       if (result) {Ð         console.log('Tatum connection test passed.
Proceeding with deployment...');Ð         return deployAllContracts();Ð
        } else {Ð         console.log('Tatum connection test failed. Please
check your configuration.');Ð         process.exit(1);Ð       }Ð     })Ð
      .then(() => {Ð       clearTimeout(deploymentTimeout);Ð
console.log('Deployment completed successfully');Ð       process.exit(0);Ð
      })Ð     .catch((error) => {Ð       clearTimeout(deploymentTimeout);Ð
        console.error('Deployment failed:', { message: error.message, stack:
error.stack, details: error.details || 'No additional details' });Ð
process.exit(1);Ð     });Ð }Ð Ð module.exports = {Ð   deployContract,Ð
deployAllContracts,Ð   verifyContract,Ð   getWeb3Provider,Ð
testTatumConnection,Ð };
```

## scripts\deploy_polkadot.js

```
// deploy_polkadot.jsÐrequire('dotenv').config();Ðconst { ApiPromise,
WsProvider, Keyring } = require('@polkadot/api');Ðconst { cryptoWaitReady } =
require('@polkadot/util-crypto');Ðconst fs = require('fs');Ðconst path =
require('path');ÐÐ// Polkadot Network ConfigurationÐconst POLKADOT_PROVIDER_URL
= process.env.POLKADOT_PROVIDER_URL || 'wss://rpc.polkadot.io';Ðconst MNEMONIC
= process.env.POLKADOT_MNEMONIC;Ðasync function connectToPolkadotNetwork() {Ð
try {Ð   // Wait for crypto libraries to be readyÐ   await cryptoWaitReady();Ð
    // Create a new WS providerÐ   const wsProvider = new
WsProvider(POLKADOT_PROVIDER_URL);Ð   Ð   // Create API instanceÐ   const api
= await ApiPromise.create({ provider: wsProvider });Ð   Ð   // Verify network
connectionÐ   const [chain, nodeName, nodeVersion] = await Promise.all([Ð
api.rpc.system.chain(),Ð     api.rpc.system.name(),Ð
api.rpc.system.version()Ð   ]);Ð   console.log(`Connected to chain: ${chain},
Node: ${nodeName} v${nodeVersion}`);Ð   return { api, wsProvider };Ð } catch
(error) {Ð   console.error('Polkadot network connection failed:', error);Ð
throw error;Ð }ÐÐasync function setupPolkadotAccount() {Ð const keyring = new
Keyring({ type: 'sr25519' });Ð Ð if (!MNEMONIC) {Ð   throw new
Error('Polkadot mnemonic not found in environment variables');Ð }Ð // Create
account from mnemonicÐ const account = keyring.addFromMnemonic(MNEMONIC);Ð Ð
return { keyring, account };ÐÐasync function deployRelayChainContracts(api,
account) {Ð try {Ð   // Load existing Ethereum contract addressesÐ   const
contractAddressesPath = path.join(__dirname, '../config/
contract_addresses.json');Ð   const contractAddresses =
JSON.parse(fs.readFileSync(contractAddressesPath, 'utf8'));Ð   // Prepare
transactionÐ   const tx = api.tx.system.remark(Ð     JSON.stringify({Ð
sourceChain: 'Ethereum',Ð       sourceContractAddresses: contractAddresses,Ð
        timestamp: Date.now()Ð     })Ð   );Ð   // Sign and send transactionÐ
    const hash = await tx.signAndSend(account);Ð   Ð   console.log('Relay
Chain Transaction Hash:', hash.toHex());Ð   Ð   return hash;Ð } catch (error)
{Ð   console.error('Relay Chain contract deployment failed:', error);Ð
throw error;Ð }ÐÐasync function performCrossChainRegistration() {Ð let
connection = null;Ð Ð try {Ð   // Connect to Polkadot NetworkÐ   connection =
await connectToPolkadotNetwork();Ð   Ð   // Setup accountÐ   const
{ account } = await setupPolkadotAccount();Ð   Ð   // Deploy Relay Chain
ContractsÐ   const relayChainHash = await deployRelayChainContracts(Ð
connection.api, Ð     accountÐ   );Ð   Ð   // Track performance metricsÐ
const performanceMetrics = {Ð     networkName: 'Polkadot',Ð
connectionTime: Date.now(),Ð     relayChainTransactionHash:
relayChainHash.toHex(),Ð     status: 'Successful'Ð   };Ð   Ð   // Save
performance metricsÐ   const metricsPath = path.join(__dirname, '../metrics/
polkadot_deployment.json');Ð   fs.writeFileSync(metricsPath,
JSON.stringify(performanceMetrics, null, 2));Ð   Ð   console.log('Cross-Chain
Registration Complete');Ð   return performanceMetrics;Ð } catch (error) {Ð
console.error('Cross-Chain Registration Failed:', error);Ð   throw error;Ð }
finally {Ð   // Close WebSocket connectionÐ   if (connection &&
connection.wsProvider) {Ð     connection.wsProvider.disconnect();Ð   }Ð }ÐÐ//
Main executionÐif (require.main === module) {Ð performCrossChainRegistration()Ð
    .then(metrics => {Ð     console.log('Deployment Metrics:', metrics);Ð
process.exit(0);Ð   })Ð   .catch(error => {Ð     console.error('Deployment
Error:', error);Ð     process.exit(1);Ð   });ÐÐmodule.exports = {Ð
connectToPolkadotNetwork,Ð setupPolkadotAccount,Ð deployRelayChainContracts,Ð
performCrossChainRegistrationÐ};
```

## test\cllimateintergration_test.js

```
// test/Integration.test.jsÐconst CityRegister =
artifacts.require("CityRegister");Ðconst CityEmissions =
artifacts.require("CityEmissionsContract");Ðconst RenewalTheory =
artifacts.require("RenewalTheoryContract");Ðontract("Integration Tests",
accounts => {Ð let cityRegister, cityEmissions, renewalTheory;Ð const owner =
accounts[0];Ð const city = "Melbourne";Ð const sector = "Aviation";Ð
beforeEach(async () => {Ð   cityRegister = await CityRegister.new({ from:
owner });Ð   cityEmissions = await CityEmissions.new({ from: owner });Ð
renewalTheory = await RenewalTheory.new({ from: owner });Ð });Ð it("should
process emissions data and calculate renewal metrics", async () => {Ð   //
Register cityÐ   await cityRegister.registerCity(city, { from: owner });Ð
    // Add emissions dataÐ   const timestamp = Math.floor(Date.now() / 1000);Ð
    const emissionValue = web3.utils.toWei("0.000736959", "ether");Ð   Ð
await cityEmissions.addEmissionData(Ð     city,Ð     timestamp,Ð     sector,Ð
      emissionValue,Ð     80, // AQIÐ     { from: owner }Ð   );Ð   // Process
renewal theory calculationsÐ   await renewalTheory.processRenewal(city,
sector);Ð   // Verify resultsÐ   const renewalData = await
renewalTheory.getSectorStats(city, sector);Ð
assert.notEqual(renewalData.totalRecordings.toNumber(), 0, "No recordings
found");Ð });Ð});
```

## test\uncertainty_analytics_test.js

```
const UncertaintyAnalytics = artifacts.require("UncertaintyAnalytics");Ðconst
RequestManager = artifacts.require("RequestManager");Ðconst ResponseManager =
artifacts.require("ResponseManager");Ðontract("UncertaintyAnalytics",
function(accounts) {Ð let analytics;Ð let requestManager;Ð let
responseManager;Ð const [owner, requester, responder] = accounts;Ð const
measureTime = (fn) => {Ð   console.log(">>> Measuring time for function");Ð
const start = Date.now();Ð   return fn().then(result => {Ð     const end =
Date.now();Ð     console.log(`>>> Time measurement: ${end - start} ms`);Ð
return { result, duration: end - start };Ð   });Ð };Ð const getGasCost = (tx)
=> {Ð   console.log(">>> Calculating Gas Cost");Ð   return
web3.eth.getTransactionReceipt(tx.tx)Ð     .then(receipt => {Ð
console.log("Transaction Receipt:", receipt);Ð       const gasCost = web3.util
s.toBN(receipt.gasUsed).mul(web3.utils.toBN(tx.receipt.effectiveGasPrice));Ð
        console.log("Gas Cost:", gasCost.toString());Ð       return gasCost;Ð
      });Ð };Ð const advanceTime = () => {Ð   console.log(`>>> Advancing time
by 86401 seconds`);Ð   return new Promise((resolve, reject) => {Ð
web3.currentProvider.send({Ð       jsonrpc: "2.0",Ð       method:
"evm_increaseTime",Ð       params: [86401],Ð       id: new Date().getTime(),Ð
      }, (err1) => {Ð       if (err1) return reject(err1);Ð
web3.currentProvider.send({Ð         jsonrpc: "2.0",Ð         method:
"evm_mine",Ð         params: [],Ð         id: new Date().getTime()Ð       },
(err2) => {Ð         if (err2) return reject(err2);Ð         console.log(">>>
Time advanced successfully");Ð         resolve();Ð       });Ð     });Ð   });Ð
  };Ð beforeEach(function(done) {Ð   this.timeout(0);Ð
UncertaintyAnalytics.new()Ð     .then(analyticsContract => {Ð       analytics
= analyticsContract;Ð       return Promise.all([Ð
RequestManager.new(analytics.address),Ð
ResponseManager.new(analytics.address)Ð       ]);Ð     })Ð
      .then(([requestManagerContract, responseManagerContract]) => {Ð
requestManager = requestManagerContract;Ð       responseManager =
responseManagerContract;Ð       done();Ð     })Ð     .catch(done);Ð });Ð
it("Should analyze full request-response cycle with metrics", function(done) {Ð
    this.timeout(0);Ð   const metrics = {Ð     confirmationTimes: [],Ð
executionTimes: [],Ð     gasCosts: [],Ð     failedTransactions: 0,Ð
invalidAddressAttempts: 0,Ð     invalidRequirementAttempts: 0Ð   };Ð
console.log("\n=== Starting 10 Request-Response Cycles ===");Ð   // Create a
promise chain for request-response cyclesÐ   const requestCycles =
Array(10).fill().reduce((chain, _, i) => {Ð     return chain.then(() => Ð
        measureTime(() => Ð         requestManager.submitRequest({Ð
from: requester,Ð           value: web3.utils.toWei("0.001", "ether")Ð
          })Ð       ).then(({ duration: confirmTime, result: tx }) => {Ð
          metrics.confirmationTimes.push(confirmTime);Ð         return
getGasCost(tx).then(gasCost => {Ð           metrics.gasCosts.push(gasCost);Ð
            return measureTime(() => Ð
responseManager.submitResponse(i + 1, { from: responder })Ð
            ).then(({ duration: execTime }) => {Ð
metrics.executionTimes.push(execTime);Ð             Ð             if (i % 3
=== 0) {Ð               return Promise.all([Ð
analytics.updateDisruptionLevel(i + 1, { from: owner }),Ð
analytics.updateEscalationLevel(Math.floor(i / 2), { from: owner })Ð
               ]);Ð             }Ð           });Ð         });Ð       })Ð
      );Ð   }, Promise.resolve());Ð   // Chain invalid scenarios and final
metrics evaluationÐ   requestCyclesÐ     .then(() => {Ð
console.log("\n=== Testing Invalid Scenarios ===");Ð       Ð       return
requestManager.submitRequest({ Ð         from:
"0x0000000000000000000000000000000000000000",Ð         value:
web3.utils.toWei("0.001", "ether")Ð       }).catch(() => {Ð
metrics.invalidAddressAttempts++;Ð       });Ð     })Ð     .then(() => {Ð
        return requestManager.submitRequest({Ð
```

```
                                                    from: requester,Ð
          value: web3.utils.toWei("0.0001", "ether")Ð       }).catch(() => {Ð
          metrics.invalidRequirementAttempts++;Ð       });Ð     })Ð
      .then(() => advanceTime())Ð     .then(() =>
analytics.calculateUnavailabilityCost(1))Ð     .then(() => Promise.all([Ð
        analytics.getMetrics(),Ð       analytics.unavailabilityCost(),Ð
analytics.disruptionLevel(),Ð       analytics.escalationLevel()Ð     ]))Ð
      .then(([analyticsMetrics, unavailabilityCost, disruptionLevel,
escalationLevel]) => {Ð       console.log("\n=== Performance Metrics ===");Ð
        console.log("Average Confirmation Time:", Ð
metrics.confirmationTimes.reduce((a, b) => a + b, 0) /
metrics.confirmationTimes.length, "ms");Ð       console.log("Average
Execution Time:", Ð         metrics.executionTimes.reduce((a, b) => a + b,
0) / metrics.executionTimes.length, "ms");Ð       console.log("Average Gas
Cost:", Ð         web3.utils.fromWei(Ð           metrics.gasCosts.reduce((a,
b) => a.add(b), web3.utils.toBN(0))Ð
            .div(web3.utils.toBN(metrics.gasCosts.length))Ð         ), "ETH");Ð
        console.log("\n=== Uncertainty Metrics ===");Ð
console.log("Success Rate:", analyticsMetrics.successRate.toString(), "%");Ð
        console.log("Average Processing Time:",
analyticsMetrics.avgProcessingTime.toString(), "seconds");Ð
console.log("Total Transaction Cost:",
web3.utils.fromWei(analyticsMetrics.totalCost), "ETH");Ð
console.log("Unavailability Cost:", web3.utils.fromWei(unavailabilityCost),
"ETH");Ð       console.log("Disruption Level:", disruptionLevel.toString());Ð
        console.log("Escalation Level:", escalationLevel.toString());Ð
console.log("\n=== Error Metrics ===");Ð       console.log("Failed
Transactions:", metrics.failedTransactions);Ð       console.log("Invalid
Address Attempts:", metrics.invalidAddressAttempts);Ð
console.log("Invalid Requirement Attempts:",
metrics.invalidRequirementAttempts);Ð       console.log("Total Disruptions:",
analyticsMetrics.disruptionCount.toString());Ð
assert.isTrue(analyticsMetrics.successRate.gt(web3.utils.toBN(0)), "Success
rate should be positive");Ð
assert.isTrue(unavailabilityCost.gt(web3.utils.toBN(0)), "Should have
unavailability cost");Ð       assert.equal(metrics.invalidAddressAttempts +
metrics.invalidRequirementAttempts, 2, "Should have caught invalid
scenarios");Ð       Ð       done();Ð     })Ð     .catch(done);Ð });Ð
it("Should track data holding costs accurately", function(done) {Ð
this.timeout(0);Ð   const holdingCost = web3.utils.toWei("0.0005", "ether");Ð
    Ð   analytics.updateDataHoldingCost(holdingCost, { from: owner })Ð
      .then(() => analytics.dataHoldingCost())Ð     .then((dataHoldingCost)
=> {Ð       console.log("\n=== Storage Metrics ===");Ð
console.log("Data Holding Cost:", web3.utils.fromWei(dataHoldingCost), "ETH");Ð
        Ð       assert.equal(dataHoldingCost.toString(), holdingCost, "Data
holding cost should match");Ð       Ð       done();Ð     })Ð     .catch(done);Ð
  });Ð});
```

## truffle-config.js

```
require('dotenv').config();Ðconst HDWalletProvider = require('@truffle/
hdwallet-provider');Ðmodule.exports = {Ð networks: {Ð   development: {Ð
host: "127.0.0.1",Ð     port: 7545,Ð     network_id: "*",Ð     gas: 6721975,Ð
      gasPrice: 20000000000Ð   },Ð   sepolia: {Ð     provider: () => new
HDWalletProvider(Ð       process.env.MNEMONIC,Ð       `https://
sepolia.infura.io/v3/${process.env.INFURA_PROJECT_ID}`Ð     ),Ð
network_id: 11155111, // '  Correct for SepoliaÐ   gas: 5500000,Ð
confirmations: 2,Ð     timeoutBlocks: 200,Ð     skipDryRun: trueÐ   },Ð
tatum_testnet: {Ð     provider: () => new HDWalletProvider({Ð
privateKeys: [process.env.PRIVATE_KEY],Ð       providerOrUrl: 'https://
ethereum-sepolia.gateway.tatum.io/',Ð       headers: {Ð         'x-api-key':
process.env.NEXT_PUBLIC_TATUM_API_KEYÐ       }Ð     }),Ð     network_id:
11155111,Ð     gas: 5500000,Ð     confirmations: 2,Ð     timeoutBlocks: 200Ð
    }Ð },Ð compilers: {Ð   solc: {Ð     version: "0.8.20",Ð     settings: {Ð
        optimizer: {Ð         enabled: true,Ð         runs: 200Ð       }Ð
      }Ð   }Ð },Ð contracts_directory: './contracts',Ð
contracts_build_directory: './build/contracts',Ð migrations_directory: './
migrations',Ð db: {Ð   enabled: falseÐ }};Ð
```

## .eslintrc.json

```
{Đ   "extends": [Đ     "next/core-web-vitals"Đ   ],Đ   "rules": {Đ     "no-
console": "off"Đ   }Đ }
```

## .next\build-manifest.json


```
{  "polyfillFiles": [    "static/chunks/polyfills-c67a75d1b6f99dc8.js"  ],
"devFiles": [],  "ampDevFiles": [],  "lowPriorityFiles": [    "static/
LF_VyIMe0tJgAyMxBw2NR/_buildManifest.js",    "static/LF_VyIMe0tJgAyMxBw2NR/
_ssgManifest.js"  ],  "rootMainFiles": [],  "pages": {    "/": [      "static/
chunks/webpack-76c7cf389ff1afa5.js",      "static/chunks/802-
c35ff74cd7ce9687.js",      "static/chunks/27-7bd7da71ab7a6b14.js",
"static/chunks/290-611e7f86e1008d4b.js",      "static/chunks/
main-8d6a5ee0c19b3a83.js",      "static/chunks/pages/
index-6e2eda49494341ce.js"    ],    "/_app": [      "static/chunks/
webpack-76c7cf389ff1afa5.js",      "static/chunks/802-c35ff74cd7ce9687.js",
      "static/chunks/27-7bd7da71ab7a6b14.js",      "static/
chunks/290-611e7f86e1008d4b.js",      "static/chunks/
main-8d6a5ee0c19b3a83.js",      "static/chunks/756-73cc2dd7f8675213.js",
"static/chunks/135-e3be18176966c22b.js",      "static/chunks/473-
d1f9c97882f52d8e.js",      "static/chunks/429-2d0a17b32146e275.js",
"static/chunks/71-2a48985b276386a1.js",      "static/
chunks/210-82b94d0ffb674aa3.js",      "static/chunks/490-4a28cc6a8b78205d.js",
      "static/chunks/475-1007e2e6e0425bd1.js",      "static/chunks/572-
e660b6f037033ad0.js",      "static/chunks/334-679c2f2e1697e890.js",
"static/chunks/767-4505cbbfe8197d72.js",      "static/chunks/234-
d68fc770e3e761f2.js",      "static/chunks/898-82355bc33e7acb9f.js",
"static/chunks/785-51a0d4a99205c763.js",      "static/
chunks/913-7acde5eadc060759.js",      "static/chunks/748-ea8a8945ac7205d1.js",
      "static/chunks/663-6c4c8e3c610cc960.js",      "static/chunks/693-
ed9e6896aee8e2ae.js",      "static/chunks/328-0d3a2bc78117f9d2.js",
"static/chunks/917-f8a8c7d48bef753c.js",      "static/
chunks/185-948dd8374c8e9ef3.js",      "static/chunks/791-fd703dda90b10a34.js",
      "static/chunks/604-bd66eae5823cd811.js",      "static/chunks/pages/
_app-42a474e1f653941c.js"    ],    "/_error": [      "static/chunks/
webpack-76c7cf389ff1afa5.js",      "static/chunks/802-c35ff74cd7ce9687.js",
      "static/chunks/27-7bd7da71ab7a6b14.js",      "static/
chunks/290-611e7f86e1008d4b.js",      "static/chunks/
main-8d6a5ee0c19b3a83.js",      "static/chunks/pages/_error-
a237099c62580823.js"    ],    "/transactions": [      "static/chunks/
webpack-76c7cf389ff1afa5.js",      "static/chunks/802-c35ff74cd7ce9687.js",
      "static/chunks/27-7bd7da71ab7a6b14.js",      "static/
chunks/290-611e7f86e1008d4b.js",      "static/chunks/
main-8d6a5ee0c19b3a83.js",      "static/chunks/pages/transactions-
fa1f2a601fe1700c.js"    ]  },  "ampFirstPages": []}
```

## .next\export-marker.json

{"version":1,"hasExportPathMap":false,"exportTrailingSlash":false,"isNextImage Imported":false}

## .next\images-manifest.json


```
{"version":1,"images":{"deviceSizes":
[640,750,828,1080,1200,1920,2048,3840],"imageSizes":
[16,32,48,64,96,128,256,384],"path":"/_next/
image","loader":"default","loaderFile":"","domains":
[],"disableStaticImages":false,"minimumCacheTTL":60,"formats":["image/
avif","image/
webp"],"dangerouslyAllowSVG":false,"contentSecurityPolicy":"script-src
'none'; frame-src 'none';
sandbox;","contentDispositionType":"inline","remotePatterns":
[],"unoptimized":false,"sizes":
[640,750,828,1080,1200,1920,2048,3840,16,32,48,64,96,128,256,384]}}
```

## .next\package.json

```
{"type": "commonjs"}
```

## .next\prerender-manifest.json

{"version":4,"routes":{},"dynamicRoutes":{},"preview":{"previewModeId":"8bff34
782e6fe3263e5a3239ff5e131e","previewModeSigningKey":"a32fe539184ec4aef2a83efcc
fe17639a46e9c30baf9bc5a1354029cef15d5ff","previewModeEncryptionKey":"beeb07172
3885a6cb1adc9bd9e0e682699e1d85186743b0b02e63e5583cd45a5"},"notFoundRoutes":[]}

## .next\react-loadable-manifest.json


```
{  "_app.tsx -> @/components/GlobalErrorHandler": {    "id": 7292,
"files": [      "static/chunks/292.e86441fef50959b5.js"    ]  },  "index.tsx -
> @/components/BlockchainContext": {    "id": 6945,    "files": []  }}
```

## .next\routes-manifest.json

{"version":3,"pages404":true,"caseSensitive":false,"basePath":"","redirects":[
{"source":"/:path+/","destination":"/:path+","internal":true,"statusCode":308,
"regex":"^(?:/((?:[^/]+?)(?:/(?:[^/]+?))*))/$"}],"headers":[],"dynamicRoutes":
[],"staticRoutes":[{"page":"/","regex":"^/(?:/)?$","routeKeys":
{},"namedRegex":"^/(?:/)?$"},{"page":"/transactions","regex":"^/
transactions(?:/)?$","routeKeys":{},"namedRegex":"^/transactions(?:/)?
$"}],"dataRoutes":[],"rsc":{"header":"RSC","varyHeader":"RSC, Next-Router-
State-Tree, Next-Router-Prefetch, Next-Url","prefetchHeader":"Next-Router-
Prefetch","didPostponeHeader":"x-nextjs-postponed","contentTypeHeader":"text/
x-component","suffix":".rsc","prefetchSuffix":".prefetch.rsc"},"rewrites":[]}

## .next\server\chunks\font-manifest.json

```
[]
```

## .next\server\font-manifest.json

```
[]
```

## .next\server\middleware-manifest.json


```
{  "sortedMiddleware": [],  "middleware": {},  "functions": {},  "version": 2}
```

## .next\server\next-font-manifest.json

{"pages":{},"app":{},"appUsingSizeAdjust":false,"pagesUsingSizeAdjust":false}

## .next\server\pages-manifest.json

{"/_app":"pages/_app.js","/":"pages/index.html","/_error":"pages/_error.js","/transactions":"pages/transactions.html","/_document":"pages/_document.js","/404":"pages/404.html"}

## .vscode\settings.json

```
ĐĐ   "python.pythonPath": "~/AppData/Local/Programs/Python/Python313/
python.exe"Þ Đ
```

## build\contracts\TransactionTypes.json


```
{  "contractName": "TransactionTypes",  "abi": [],  "metadata":
"{\"compiler\":{\"version\":\"0.8.20+commit.a1b79de6\"},\"language\":
\"Solidity\",\"output\":{\"abi\":[],\"devdoc\":{\"kind\":\"dev\",\"methods\":
{},\"version\":1},\"userdoc\":{\"kind\":\"user\",\"methods\":{},
\"version\":1}},\"settings\":{\"compilationTarget\":{\"project:/library/
TransactionTypes.sol\":\"TransactionTypes\"},\"evmVersion\":\"shanghai\",
\"libraries\":{},\"metadata\":{\"bytecodeHash\":\"ipfs\"},\"optimizer\":
{\"enabled\":true,\"runs\":200},\"remappings\":[]},\"sources\":{\"project:/
library/TransactionTypes.sol\":{\"keccak256\":
\"0x9099d7f9d5aabd02d53f72ca725bca1e7c190022f67940a6538992ea841884b0\",
\"license\":\"MIT\",\"urls\":[\"bzz-
raw://42c857e8874473e4c61b155dd8b1f5408e4dce9190fe8bfcac2bdcb137cd488c\",
\"dweb:/ipfs/QmUnaaAb9NcJGE17YL55q48dJbi5tpW9b3VnRyNDKvDBKQ\"]}},
\"version\":1}",  "bytecode": "0x60556032600b8282823980515f1a607314602657634e4
87b7160e01b5f525f60045260245ffd5b305f52607381538281f3fe730000000000000000000000
0000000000000000000030146080604052 5f80fdfea2646970667 35822122026ef393c8d8b8db52
2a676f416a9386f3c7d3fad1a61725a393341bf35511c0564736f6c63430008140033",
"deployedBytecode": "0x730000000000000000000000000000000000000000000030146080604052
5f80fdfea2646970667 35822122026ef393c8d8b8db522a676f416a9386f3c7d3fad1a61725a3
93341bf35511c0564736f6c63430008140033",  "immutableReferences": {},
"generatedSources": [],  "deployedGeneratedSources": [],  "sourceMap":
"61:831:47:-:0;;;;;;;;;;;;;;;;-1:-1:-1;;;61:831:47;;;;;;;;;;;;;;;;;",
"deployedSourceMap": "61:831:47:-:0;;;;;;;;",  "source": "// SPDX-License-
Identifier: MIT\r\npragma solidity ^0.8.17;\r\n\r\nlibrary TransactionTypes
{\r\n    struct SpeculativeTx {\r\n        bytes32 id;\r\n        address
sender;\r\n        address receiver;\r\n        uint256 anticipatedTime;
\r\n        bytes32 dataHash;\r\n        bool isAssetTransfer;\r\n
uint256 interpolationTime;\r\n        bytes rbfParams;\r\n
mapping(bytes32 => bool) validationProofs;\r\n    }\r\n\r\n    struct
ConfirmableTx {\r\n        bytes32 id;\r\n        address sender;\r\n
address receiver;\r\n        uint256 confirmationTime;\r\n        bytes32
dataHash;\r\n        bool isAssetTransfer;\r\n        bytes32 speculativeTxId;
\r\n        mapping(bytes32 => bool) zkProofs;\r\n    }\r\n\r\n    struct
Channel {\r\n        bytes32 id;\r\n        address sourceBridge;\r\n
address targetBridge;\r\n        uint256 creationTime;\r\n        bool
isActive;\r\n        uint256 confidenceThreshold;\r\n    }\r\n}\r\n",
"sourcePath": "C:\\Users\\Bonsu\\Documents\\Blockchain Multi Industrial
Service Center\\Blockchain MultiIndustrial Projects\
\BlockchainMultiIndustryRepo\\library\\TransactionTypes.sol",  "ast": {
"absolutePath": "project:/library/TransactionTypes.sol",
"exportedSymbols": {      "TransactionTypes": [        20693      ]    },
"id": 20694,    "license": "MIT",    "nodeType": "SourceUnit",    "nodes": [
      {        "id": 20639,        "literals": [          "solidity",
          "^",          "0.8",          ".17"        ],        "nodeType":
"PragmaDirective",        "src": "33:24:47"      },      {        "abstract":
false,        "baseContracts": [],        "canonicalName": "TransactionTypes",
        "contractDependencies": [],        "contractKind": "library",
"fullyImplemented": true,        "id": 20693,
"linearizedBaseContracts": [          20693        ],        "name":
"TransactionTypes",        "nameLocation": "69:16:47",        "nodeType":
"ContractDefinition",        "nodes": [          {
"canonicalName": "TransactionTypes.SpeculativeTx",            "id": 20660,
            "members": [              {                "constant": false,
                "id": 20641,                "mutability": "mutable",
                "name": "id",                "nameLocation": "133:2:47",
                "nodeType": "VariableDeclaration",                "scope":
20660,                "src": "125:10:47",                "stateVariable":
false,                "storageLocation": "default",
"typeDescriptions": {
```

```
                                    "typeIdentifier": "t_bytes32",
                "typeString": "bytes32"                },
"typeName": {                 "id": 20640,                 "name":
"bytes32",                 "nodeType": "ElementaryTypeName",
                "src": "125:7:47",                 "typeDescriptions": {
                  "typeIdentifier": "t_bytes32",
"typeString": "bytes32"                 }              },
"visibility": "internal"              },              {
"constant": false,              "id": 20643,                 "mutability":
"mutable",                "name": "sender",                "nameLocation":
"154:6:47",                "nodeType": "VariableDeclaration",
"scope": 20660,                "src": "146:14:47",
"stateVariable": false,                "storageLocation": "default",
                "typeDescriptions": {                  "typeIdentifier":
"t_address",                "typeString": "address"              },
                "typeName": {                "id": 20642,
"name": "address",                "nodeType": "ElementaryTypeName",
                "src": "146:7:47",                 "stateMutability":
"nonpayable",                "typeDescriptions": {
"typeIdentifier": "t_address",                  "typeString": "address"
                }              },                "visibility": "internal"
              },              {                "constant": false,
                "id": 20645,                "mutability": "mutable",
                "name": "receiver",                "nameLocation": "179:8:47",
                "nodeType": "VariableDeclaration",                "scope":
20660,                "src": "171:16:47",                "stateVariable":
false,                "storageLocation": "default",
"typeDescriptions": {                "typeIdentifier": "t_address",
                "typeString": "address"                },
"typeName": {                "id": 20644,                 "name":
"address",                "nodeType": "ElementaryTypeName",
                "src": "171:7:47",                 "stateMutability":
"nonpayable",                "typeDescriptions": {
"typeIdentifier": "t_address",                  "typeString": "address"
                }              },                "visibility": "internal"
              },              {                "constant": false,
                "id": 20647,                "mutability": "mutable",
                "name": "anticipatedTime",                "nameLocation":
"206:15:47",                "nodeType": "VariableDeclaration",
"scope": 20660,                "src": "198:23:47",
"stateVariable": false,                "storageLocation": "default",
                "typeDescriptions": {                  "typeIdentifier":
"t_uint256",                "typeString": "uint256"              },
                "typeName": {                "id": 20646,
"name": "uint256",                "nodeType": "ElementaryTypeName",
                "src": "198:7:47",                 "typeDescriptions": {
                  "typeIdentifier": "t_uint256",
"typeString": "uint256"                 }              },
"visibility": "internal"              },              {
"constant": false,              "id": 20649,                 "mutability":
"mutable",                "name": "dataHash",                "nameLocation":
"240:8:47",                "nodeType": "VariableDeclaration",
"scope": 20660,                "src": "232:16:47",
"stateVariable": false,                "storageLocation": "default",
                "typeDescriptions": {                  "typeIdentifier":
"t_bytes32",                "typeString": "bytes32"              },
                "typeName": {                "id": 20648,
"name": "bytes32",                "nodeType": "ElementaryTypeName",
                "src": "232:7:47",                 "typeDescriptions": {
                  "typeIdentifier": "t_bytes32",
```

```
"typeString": "bytes32"                  }                },
"visibility": "internal"              },              {
"constant": false,               "id": 20651,                "mutability":
"mutable",               "name": "isAssetTransfer",
"nameLocation": "264:15:47",                "nodeType": "VariableDeclaration",
               "scope": 20660,                   "src": "259:20:47",
               "stateVariable": false,                "storageLocation":
"default",               "typeDescriptions": {
"typeIdentifier": "t_bool",                "typeString": "bool"
               },               "typeName": {                   "id": 20650,
                 "name": "bool",                "nodeType":
"ElementaryTypeName",                   "src": "259:4:47",
"typeDescriptions": {                   "typeIdentifier": "t_bool",
                   "typeString": "bool"                  }              },
               "visibility": "internal"              },              {
               "constant": false,                "id": 20653,
"mutability": "mutable",               "name": "interpolationTime",
               "nameLocation": "298:17:47",                "nodeType":
"VariableDeclaration",               "scope": 20660,                "src":
"290:25:47",               "stateVariable": false,
"storageLocation": "default",                "typeDescriptions": {
                 "typeIdentifier": "t_uint256",
"typeString": "uint256"                },                "typeName": {
                 "id": 20652,                 "name": "uint256",
                 "nodeType": "ElementaryTypeName",                "src":
"290:7:47",                 "typeDescriptions": {
"typeIdentifier": "t_uint256",                   "typeString": "uint256"
                 }              },                "visibility": "internal"
             },              {                "constant": false,
                 "id": 20655,                "mutability": "mutable",
                 "name": "rbfParams",                "nameLocation":
"332:9:47",                 "nodeType": "VariableDeclaration",
"scope": 20660,                   "src": "326:15:47",
"stateVariable": false,                "storageLocation": "default",
               "typeDescriptions": {                  "typeIdentifier":
"t_bytes_storage_ptr",                  "typeString": "bytes"
               },               "typeName": {                  "id": 20654,
                 "name": "bytes",                "nodeType":
"ElementaryTypeName",                   "src": "326:5:47",
"typeDescriptions": {                     "typeIdentifier":
"t_bytes_storage_ptr",                     "typeString": "bytes"
                 }              },                "visibility": "internal"
             },              {                "constant": false,
                 "id": 20659,                "mutability": "mutable",
                 "name": "validationProofs",                "nameLocation":
"377:16:47",                 "nodeType": "VariableDeclaration",
"scope": 20660,                   "src": "352:41:47",
"stateVariable": false,                "storageLocation": "default",
               "typeDescriptions": {                  "typeIdentifier":
"t_mapping$_t_bytes32_$_t_bool_$",                "typeString":
"mapping(bytes32 => bool)"               },                "typeName": {
                 "id": 20658,                 "keyName": "",
                 "keyNameLocation": "-1:-1:-1",                "keyType": {
                   "id": 20656,                   "name": "bytes32",
                   "nodeType": "ElementaryTypeName",
"src": "360:7:47",                   "typeDescriptions": {
                     "typeIdentifier": "t_bytes32",
"typeString": "bytes32"                   }                 },
                 "nodeType": "Mapping",
```

```
                                                    "src": "352:24:47",
                "typeDescriptions": {                    "typeIdentifier":
"t_mapping$_t_bytes32_$_t_bool_$",                "typeString":
"mapping(bytes32 => bool)"                },                "valueName":
"",                "valueNameLocation": "-1:-1:-1",
"valueType": {                    "id": 20657,                    "name":
"bool",                "nodeType": "ElementaryTypeName",
                "src": "371:4:47",                "typeDescriptions":
{                  "typeIdentifier": "t_bool",
"typeString": "bool"                }              }            },
              "visibility": "internal"            }          ],
            "name": "SpeculativeTx",            "nameLocation": "100:13:47",
            "nodeType": "StructDefinition",            "scope": 20693,
            "src": "93:308:47",            "visibility": "public"          },
          {            "canonicalName": "TransactionTypes.ConfirmableTx",
            "id": 20679,            "members": [            {
              "constant": false,              "id": 20662,
"mutability": "mutable",              "name": "id",
"nameLocation": "449:2:47",              "nodeType": "VariableDeclaration",
              "scope": 20679,              "src": "441:10:47",
              "stateVariable": false,              "storageLocation":
"default",              "typeDescriptions": {
"typeIdentifier": "t_bytes32",              "typeString": "bytes32"
              },              "typeName": {              "id": 20661,
                "name": "bytes32",              "nodeType":
"ElementaryTypeName",              "src": "441:7:47",
"typeDescriptions": {                "typeIdentifier": "t_bytes32",
                "typeString": "bytes32"              }
              },              "visibility": "internal"            },
            {              "constant": false,              "id": 20664,
              "mutability": "mutable",              "name": "sender",
              "nameLocation": "470:6:47",              "nodeType":
"VariableDeclaration",              "scope": 20679,              "src":
"462:14:47",              "stateVariable": false,
"storageLocation": "default",              "typeDescriptions": {
                "typeIdentifier": "t_address",
"typeString": "address"              },              "typeName": {
                "id": 20663,              "name": "address",
                "nodeType": "ElementaryTypeName",              "src":
"462:7:47",              "stateMutability": "nonpayable",
                "typeDescriptions": {              "typeIdentifier":
"t_address",                "typeString": "address"              }
              },              "visibility": "internal"            },
            {              "constant": false,              "id": 20666,
              "mutability": "mutable",              "name": "receiver",
              "nameLocation": "495:8:47",              "nodeType":
"VariableDeclaration",              "scope": 20679,              "src":
"487:16:47",              "stateVariable": false,
"storageLocation": "default",              "typeDescriptions": {
                "typeIdentifier": "t_address",
"typeString": "address"              },              "typeName": {
                "id": 20665,              "name": "address",
                "nodeType": "ElementaryTypeName",              "src":
"487:7:47",              "stateMutability": "nonpayable",
                "typeDescriptions": {              "typeIdentifier":
"t_address",                "typeString": "address"              }
              },              "visibility": "internal"            },
            {              "constant": false,              "id": 20668,
              "mutability": "mutable",              "name":
"confirmationTime",
```

```
                                "nameLocation": "522:16:47",
              "nodeType": "VariableDeclaration",                "scope":
20679,                "src": "514:24:47",                "stateVariable":
false,                "storageLocation": "default",
"typeDescriptions": {                  "typeIdentifier": "t_uint256",
                "typeString": "uint256"                },
"typeName": {                "id": 20667,                   "name":
"uint256",                 "nodeType": "ElementaryTypeName",
                "src": "514:7:47",                 "typeDescriptions": {
                  "typeIdentifier": "t_uint256",
"typeString": "uint256"                  }                },
"visibility": "internal"                },                {
"constant": false,                "id": 20670,                "mutability":
"mutable",                "name": "dataHash",                "nameLocation":
"557:8:47",                "nodeType": "VariableDeclaration",
"scope": 20679,                    "src": "549:16:47",
"stateVariable": false,                  "storageLocation": "default",
              "typeDescriptions": {                    "typeIdentifier":
"t_bytes32",                  "typeString": "bytes32"                },
                "typeName": {                  "id": 20669,
"name": "bytes32",                  "nodeType": "ElementaryTypeName",
                "src": "549:7:47",                 "typeDescriptions": {
                  "typeIdentifier": "t_bytes32",
"typeString": "bytes32"                  }                },
"visibility": "internal"                },                {
"constant": false,                "id": 20672,                "mutability":
"mutable",                "name": "isAssetTransfer",
"nameLocation": "581:15:47",                "nodeType": "VariableDeclaration",
                "scope": 20679,                   "src": "576:20:47",
                "stateVariable": false,                  "storageLocation":
"default",                "typeDescriptions": {
"typeIdentifier": "t_bool",                  "typeString": "bool"
                },                "typeName": {                  "id": 20671,
                  "name": "bool",                  "nodeType":
"ElementaryTypeName",                  "src": "576:4:47",
"typeDescriptions": {                    "typeIdentifier": "t_bool",
                    "typeString": "bool"                  }                },
                "visibility": "internal"                },                {
                "constant": false,                "id": 20674,
"mutability": "mutable",                "name": "speculativeTxId",
                "nameLocation": "615:15:47",                "nodeType":
"VariableDeclaration",                "scope": 20679,                "src":
"607:23:47",                "stateVariable": false,
"storageLocation": "default",                "typeDescriptions": {
                  "typeIdentifier": "t_bytes32",
"typeString": "bytes32"                },                "typeName": {
                  "id": 20673,                  "name": "bytes32",
                  "nodeType": "ElementaryTypeName",                  "src":
"607:7:47",                  "typeDescriptions": {
"typeIdentifier": "t_bytes32",                  "typeString": "bytes32"
                  }                },                "visibility": "internal"
              },                {                "constant": false,
                "id": 20678,                "mutability": "mutable",
                "name": "zkProofs",                "nameLocation": "666:8:47",
                "nodeType": "VariableDeclaration",                "scope":
20679,                "src": "641:33:47",                "stateVariable":
false,                "storageLocation": "default",
"typeDescriptions": {                  "typeIdentifier":
"t_mapping$_t_bytes32_$_t_bool_$",                  "typeString":
"mapping(bytes32 => bool)"
```

```
                                    },                "typeName": {
                "id": 20677,                "keyName": "",
                "keyNameLocation": "-1:-1:-1",                "keyType": {
                  "id": 20675,                  "name": "bytes32",
                  "nodeType": "ElementaryTypeName",
"src": "649:7:47",                "typeDescriptions": {
                    "typeIdentifier": "t_bytes32",
"typeString": "bytes32"                  }                },
                "nodeType": "Mapping",                "src": "641:24:47",
                "typeDescriptions": {                 "typeIdentifier":
"t_mapping$_t_bytes32_$_t_bool_$",                "typeString":
"mapping(bytes32 => bool)"                },                "valueName":
"",                 "valueNameLocation": "-1:-1:-1",
"valueType": {                  "id": 20676,                  "name":
"bool",                "nodeType": "ElementaryTypeName",
                "src": "660:4:47",                "typeDescriptions":
{                  "typeIdentifier": "t_bool",
"typeString": "bool"                }                },
                "visibility": "internal"              }            ],
            "name": "ConfirmableTx",            "nameLocation": "416:13:47",
            "nodeType": "StructDefinition",            "scope": 20693,
            "src": "409:273:47",            "visibility": "public"          },
          {            "canonicalName": "TransactionTypes.Channel",
            "id": 20692,            "members": [              {
                "constant": false,                "id": 20681,
"mutability": "mutable",                "name": "id",
"nameLocation": "724:2:47",                "nodeType": "VariableDeclaration",
                "scope": 20692,                "src": "716:10:47",
                "stateVariable": false,                "storageLocation":
"default",                "typeDescriptions": {
"typeIdentifier": "t_bytes32",                "typeString": "bytes32"
                },                "typeName": {                "id": 20680,
                  "name": "bytes32",                "nodeType":
"ElementaryTypeName",                "src": "716:7:47",
"typeDescriptions": {                  "typeIdentifier": "t_bytes32",
                  "typeString": "bytes32"                }
                },                "visibility": "internal"              },
              {                "constant": false,                "id": 20683,
                "mutability": "mutable",                "name":
"sourceBridge",                "nameLocation": "745:12:47",
"nodeType": "VariableDeclaration",                "scope": 20692,
                "src": "737:20:47",                "stateVariable": false,
                "storageLocation": "default",
"typeDescriptions": {                "typeIdentifier": "t_address",
                "typeString": "address"                },
"typeName": {                "id": 20682,                "name":
"address",                "nodeType": "ElementaryTypeName",
                "src": "737:7:47",                "stateMutability":
"nonpayable",                "typeDescriptions": {
"typeIdentifier": "t_address",                "typeString": "address"
                }              },              "visibility": "internal"
              },              {              "constant": false,
                "id": 20685,              "mutability": "mutable",
                "name": "targetBridge",              "nameLocation":
"776:12:47",              "nodeType": "VariableDeclaration",
"scope": 20692,                "src": "768:20:47",
"stateVariable": false,                "storageLocation": "default",
                "typeDescriptions": {                "typeIdentifier":
"t_address",                "typeString": "address"              },
                "typeName": {
```

```
                                        "id": 20684,
"name": "address",                   "nodeType": "ElementaryTypeName",
                "src": "768:7:47",                 "stateMutability":
"nonpayable",                  "typeDescriptions": {
"typeIdentifier": "t_address",                    "typeString": "address"
                }              },                "visibility": "internal"
            },             {               "constant": false,
              "id": 20687,               "mutability": "mutable",
              "name": "creationTime",                "nameLocation":
"807:12:47",               "nodeType": "VariableDeclaration",
"scope": 20692,                "src": "799:20:47",
"stateVariable": false,                "storageLocation": "default",
              "typeDescriptions": {                  "typeIdentifier":
"t_uint256",                "typeString": "uint256"              },
              "typeName": {                  "id": 20686,
"name": "uint256",                   "nodeType": "ElementaryTypeName",
                "src": "799:7:47",                 "typeDescriptions": {
                  "typeIdentifier": "t_uint256",
"typeString": "uint256"                }              },
"visibility": "internal"              },             {
"constant": false,                "id": 20689,                "mutability":
"mutable",                "name": "isActive",                "nameLocation":
"835:8:47",                "nodeType": "VariableDeclaration",
"scope": 20692,                    "src": "830:13:47",
"stateVariable": false,                "storageLocation": "default",
              "typeDescriptions": {                  "typeIdentifier":
"t_bool",                  "typeString": "bool"              },
              "typeName": {                  "id": 20688,
"name": "bool",                   "nodeType": "ElementaryTypeName",
                "src": "830:4:47",                 "typeDescriptions": {
                  "typeIdentifier": "t_bool",
"typeString": "bool"                }              },
"visibility": "internal"              },             {
"constant": false,                "id": 20691,                "mutability":
"mutable",                "name": "confidenceThreshold",
"nameLocation": "862:19:47",                "nodeType": "VariableDeclaration",
              "scope": 20692,                   "src": "854:27:47",
              "stateVariable": false,                "storageLocation":
"default",                "typeDescriptions": {
"typeIdentifier": "t_uint256",                  "typeString": "uint256"
              },                "typeName": {                  "id": 20690,
                "name": "uint256",                 "nodeType":
"ElementaryTypeName",                  "src": "854:7:47",
"typeDescriptions": {                    "typeIdentifier": "t_uint256",
                  "typeString": "uint256"                }
              },                "visibility": "internal"            }
          ],            "name": "Channel",            "nameLocation":
"697:7:47",            "nodeType": "StructDefinition",            "scope":
20693,            "src": "690:199:47",            "visibility": "public"
        }        ],        "scope": 20694,        "src": "61:831:47",
      "usedErrors": [],        "usedEvents": []      }    ],    "src":
"33:861:47"  },  "compiler": {    "name": "solc",    "version":
"0.8.20+commit.a1b79de6.Emscripten.clang"  },  "networks": {},
"schemaVersion": "3.4.16",  "updatedAt": "2025-04-16T14:10:38.519Z",
"devdoc": {    "kind": "dev",    "methods": {},    "version": 1  },
"userdoc": {    "kind": "user",    "methods": {},    "version": 1  }}
```

## config\contract_addresses.json


```
{Đ "BlockchainRegistryBase": "0x4342194bf37C4d545168B913EE97c63490eD021d",Đ
"BlockchainRegistry": "0x3C446b0a7037Cf885c49eA0d8af3a8120daF7c62",Đ
"BlockchainMonitor": "0x172565114eCb3364Fd484F1bD7e17BDb67583FF9",Đ
"ChaCha20Poly1305": "0x91aCb8b84DAbF24EE01516CdD3EEE08647B42266",Đ
"MetadataParser": "0xd048D19E436EAc21Fa4abb91462C5c24Aee076eD",Đ
"PacechainChannel": "0x9Cc32A27872dD513844E519967d26068e6449ca6",Đ
"SpeculativeTransactionHandler": "0x705090bB7158D832bF96587D28D2590e768dA268",Đ
  "ConfidenceScoreCalculator": "0x16Fe03ab5A991583E9D883e492D2CE269Bf592f9",Đ
"AssetTransferProcessor": "0x3A6Ac187683434572D6CF8aE481358e600A027CC",Đ
"TransactionValidator": "0x23B25647EB0d855e416e6dA667FD76241F3f2070",Đ
"RewardToken": "0x6D091CBaa64934f57850673ea349473d506B0FfC",Đ
"RewardDistributor": "0x626d2B536FbEaBd2D79fFbe7dbb6c5ed73E6afcd",Đ
"ProofOfStakeValidator": "0x3D79187412b18042F89AAf925Ca664a8FCBF377b",Đ
"StateManager": "0xb0Dd56E61531e701ed79E138A0330D2c5ef97e63",Đ
"TransactionRelay": "0x23c1748F4977cA95b86AfA588387b591Ce695380",Đ
"ReceivingBlockchainInterface": "0xf11a6cFE7705a81A604DC2759E611d1870c5E917",Đ
  "UncertaintyAnalytics": "0x42f78A4Be745cF6767D72028d14D2788a51c779e",Đ
"ResponseManager": "0xE679C017F59268B0d8F74069b96B5d9E30741aD2",Đ
"RequestManager": "0x68c0808F0b143b0Db6a7e69882A6b20f64B14247",Đ
"CityRegister": "0x17e9ddb311061ba9FA3a6ea517A934cc0D136f27",Đ
"CompanyRegister": "0x81aDCd0724dA5Da4b796d51c09123B53A4705D3F",Đ
"CityEmissions": "0x723332E981Ddd2577954c0e15998e66A4929b1E8",Đ
"RenewalTheory": "0x710CcD32bbD9b108ef3FdE8178F0E5dB94DCb478",Đ
"CarbonCreditMarket": "0x16F6278FBae0Fa873366628118425c34Bbb1C8c0",Đ
"CityHealth": "0xa87d6005E919f04324E7E351fa7d988E28C1Ef03",Đ
"TemperatureRenewal": "0xC2E69562926Ad558D982DD09238d64a60D515F48",Đ
"ClimateReduction": "0x3B076CD48b43d99A30C7857b37704C314C0e7171",Đ
"Mitigation": "0xD78652eEe39D0bF625340a3CA0cE5696A7625d15",Đ
"ProactiveDefenseMechanism": "0xa67cb4e026626940407680f3fbe29f365427be33",Đ
"BCADN": "0xf95a9e6ecf473594794c9cedd0da896dec73cee0",Đ "NIDRegistry":
"0x38Ab0B91f2bBD8bA57d584E33375696aB2CF0d6f",Đ "NIASRegistry":
"0xFB61F22BacF92C32a3Fa4ECcF6DB5c12Eb71654d", Đ "ABATLTranslation":
"0x71db4933bf0342b290ada075b9b40f61Cac4A88D",Đ "SequencePathRouter":
"0x56e3b418E151A6a70262f337b5c4115415dB2592",Đ "ClusteringContract":
"0x3dB865F0181A652EE731adf92557677D609f4705"Đ
```

## contract_addresses.json


```
{  "NIDRegistry": "0x38Ab0B91f2bBD8bA57d584E33375696aB2CF0d6f",
"NIASRegistry": "0xFB61F22BacF92C32a3Fa4ECcF6DB5c12Eb71654d",
"ABATLTranslation": "0x71db4933bf0342b290ada075b9b40f61Cac4A88D",
"SequencePathRouter": "0x56e3b418E151A6a70262f337b5c4115415dB2592",
"ClusteringContract": "0x3dB865F0181A652EE731adf92557677D609f4705"}
```

## launch.json

```
{Đ   "version": "0.2.0",Đ   "configurations": [Đ       {Đ           "name":
"Python: Current File",Đ           "type": "python",Đ           "request":
"launch",Đ           "program": "${file}",Đ           "console":
"integratedTerminal",Đ           "justMyCode": trueĐ       }Đ   ]Đ}
```

## package.json


```
{Đ "name": "crosschain-passchain",Đ "version": "1.0.0",Đ "description":
"Cross-chain transaction system with Passchain implementation",Đ "main":
"truffle-config.js",Đ "license": "MIT",Đ "scripts": {Đ   "dev": "next dev",Đ
    "build": "next build",Đ   "start": "next start",Đ   "lint": "next lint",Đ
    "test": "truffle test",Đ   "compile": "truffle compile",Đ   "migrate":
"truffle migrate",Đ   "measure": "node scripts/measure_performance.js",Đ
"deploy:ethereum": "node scripts/deploy_ethereum.js",Đ   "deploy:polkadot":
"node scripts/deploy_polkadot.js",Đ   "deploy:tatum": "node scripts/
deploy_ethereum.js",Đ   "deploy:tatum:test": "truffle migrate --network
tatum_testnet",Đ   "verify": "node scripts/verify-contracts.js",Đ
"analyze": "ANALYZE=true next build",Đ   "run:workflow": "python python/
blockchainworkflow.py"Đ },Đ "dependencies": {Đ   "@chainlink/contracts":
"0.8",Đ   "@openzeppelin/contracts": "^4.9.3",Đ   "@polkadot/api": "^10.9.1",Đ
    "@truffle/hdwallet-provider": "^2.1.15",Đ   "@uniswap/v2-periphery":
"1.1.0-beta.0",Đ   "@web3-react/core": "^8.2.3",Đ   "@web3-react/injected-
connector": "^6.0.7",Đ   "axios": "1.6.0",Đ   "browserify-zlib": "^0.2.0",Đ
"crypto-browserify": "^3.12.1",Đ   "csv-parse": "^5.5.2",Đ   "dotenv":
"^16.4.7",Đ   "ethers": "^6.9.0",Đ   "https-browserify": "^1.0.0",Đ
"moment": "^2.29.4",Đ   "next": "14.0.4",Đ   "numpy": "^0.0.1",Đ   "os-
browserify": "^0.3.0",Đ   "pandas": "^0.0.3",Đ   "puppeteer": "^24.6.1",Đ
"react": "18.2.0",Đ   "react-dom": "18.2.0",Đ   "stream-browserify": "^3.0.0",Đ
    "stream-http": "^3.2.0",Đ   "truffle-hdwallet-provider": "^1.0.17",Đ
"web3": "^4.16.0"Đ },Đ "devDependencies": {Đ   "@next/bundle-analyzer":
"^14.0.4",Đ   "@types/node": "^20.10.6",Đ   "@types/react": "^18.2.45",Đ
"@types/react-dom": "^18.2.0",Đ   "@types/web3": "^1.2.2",Đ   "chai":
"^5.1.2",Đ   "eslint": "^8.56.0",Đ   "eslint-config-next": "^14.0.4",Đ
"ganache-cli": "^6.12.2",Đ   "jest": "^29.7.0",Đ   "mocha": "^11.0.1",Đ
"truffle": "^5.11.5",Đ   "ts-node": "^10.9.2",Đ   "typescript": "^5.3.3"Đ }}Đ
```

## results\bcadn_analysis\network_analysis.json


```
{Đ "network_metrics": {Đ   "total_nodes": 0,Đ   "total_attacks": 0,Đ
"resolved_attacks": 0,Đ   "chain_id": 11155111,Đ   "latest_block": 8132323,Đ
    "gas_price": 2082530516Đ },Đ "node_performance": {Đ   "node_addresses":
[],Đ   "weights": [],Đ   "statuses": [],Đ   "performance_details": []Đ },Đ
"attack_history": {Đ   "node_addresses": [],Đ   "timestamps": [],Đ
"anomaly_scores": [],Đ   "attack_types": [],Đ   "resolved": []Đ },Đ
"stress_test": {Đ   "transactions": [],Đ   "total_processing_time": 0.0,Đ
"average_dynamic_fee": 0.0,Đ   "max_congestion_index": 0Đ }Đ}
```

## results\bcadn_analysis\network_results.json


```
{Ð "network_metrics": {Ð   "total_nodes": 2,Ð   "total_attacks": 1,Ð
"resolved_attacks": 0,Ð   "chain_id": 11155111,Ð   "latest_block": 8138986,Ð
    "gas_price": 2114958Ð },Ð "node_performance": {Ð   "node_addresses": [Ð
      "0x1234567890123456789012345678901234567891",Ð
"0x2345678901234567890123456789012345678902"Ð   ],Ð   "weights": [Ð     2700,Ð
      2500Ð   ],Ð   "statuses": [Ð     "Active",Ð     "Active"Ð   ],Ð
"performance_details": []Ð },Ð "attack_history": {Ð   "node_addresses": [Ð
      "0x1234567890123456789012345678901234567891"Ð   ],Ð   "timestamps": [Ð
      1744910532Ð   ],Ð   "anomaly_scores": [Ð     45Ð   ],Ð
"attack_types": [Ð     "Unknown"Ð   ],Ð   "resolved": [Ð     falseÐ   ]Ð },Ð
"stress_test": {Ð   "transactions": [Ð     {Ð       "base_fee": 9,Ð
"dynamic_fee": 9,Ð       "amount": 97,Ð       "timestamp": 1744917711Ð     },Ð
      {Ð       "base_fee": 4,Ð       "dynamic_fee": 4,Ð       "amount": 32,Ð
        "timestamp": 1744917712Ð     },Ð     {Ð       "base_fee": 7,Ð
"dynamic_fee": 7,Ð       "amount": 92,Ð       "timestamp": 1744917713Ð     },Ð
      {Ð       "base_fee": 3,Ð       "dynamic_fee": 3,Ð       "amount": 55,Ð
        "timestamp": 1744917714Ð     },Ð     {Ð       "base_fee": 1,Ð
"dynamic_fee": 1,Ð       "amount": 59,Ð       "timestamp": 1744917715Ð     },Ð
      {Ð       "base_fee": 9,Ð       "dynamic_fee": 9,Ð       "amount": 48,Ð
        "timestamp": 1744917718Ð     },Ð     {Ð       "base_fee": 5,Ð
"dynamic_fee": 5,Ð       "amount": 68,Ð       "timestamp": 1744917719Ð     },Ð
      {Ð       "base_fee": 9,Ð       "dynamic_fee": 9,Ð       "amount": 40,Ð
        "timestamp": 1744917720Ð     },Ð     {Ð       "base_fee": 9,Ð
"dynamic_fee": 9,Ð       "amount": 7,Ð       "timestamp": 1744917721Ð     },Ð
      {Ð       "base_fee": 9,Ð       "dynamic_fee": 9,Ð       "amount": 28,Ð
        "timestamp": 1744917722Ð     },Ð     {Ð       "base_fee": 9,Ð
"dynamic_fee": 9,Ð       "amount": 24,Ð       "timestamp": 1744917723Ð     },Ð
      {Ð       "base_fee": 5,Ð       "dynamic_fee": 5,Ð       "amount": 61,Ð
        "timestamp": 1744917725Ð     },Ð     {Ð       "base_fee": 5,Ð
"dynamic_fee": 5,Ð       "amount": 29,Ð       "timestamp": 1744917726Ð     },Ð
      {Ð       "base_fee": 4,Ð       "dynamic_fee": 4,Ð       "amount": 48,Ð
        "timestamp": 1744917727Ð     },Ð     {Ð       "base_fee": 8,Ð
"dynamic_fee": 8,Ð       "amount": 70,Ð       "timestamp": 1744917728Ð     },Ð
      {Ð       "base_fee": 10,Ð       "dynamic_fee": 10,Ð       "amount": 39,Ð
        "timestamp": 1744917729Ð     },Ð     {Ð       "base_fee": 6,Ð
"dynamic_fee": 6,Ð       "amount": 20,Ð       "timestamp": 1744917730Ð     },Ð
      {Ð       "base_fee": 10,Ð       "dynamic_fee": 10,Ð       "amount": 84,Ð
        "timestamp": 1744917731Ð     },Ð     {Ð       "base_fee": 7,Ð
"dynamic_fee": 7,Ð       "amount": 83,Ð       "timestamp": 1744917732Ð     },Ð
      {Ð       "base_fee": 9,Ð       "dynamic_fee": 9,Ð       "amount": 82,Ð
        "timestamp": 1744917733Ð     },Ð     {Ð       "base_fee": 10,Ð
"dynamic_fee": 10,Ð       "amount": 91,Ð       "timestamp": 1744917734Ð     },Ð
      {Ð       "base_fee": 9,Ð       "dynamic_fee": 9,Ð       "amount": 10,Ð
        "timestamp": 1744917735Ð     },Ð     {Ð       "base_fee": 8,Ð
"dynamic_fee": 8,Ð       "amount": 54,Ð       "timestamp": 1744917737Ð     },Ð
      {Ð       "base_fee": 5,Ð       "dynamic_fee": 5,Ð       "amount": 77,Ð
        "timestamp": 1744917738Ð     },Ð     {Ð       "base_fee": 9,Ð
"dynamic_fee": 9,Ð       "amount": 61,Ð       "timestamp": 1744917739Ð     },Ð
      {Ð       "base_fee": 1,Ð       "dynamic_fee": 1,Ð       "amount": 92,Ð
        "timestamp": 1744917740Ð     },Ð     {Ð       "base_fee": 2,Ð
"dynamic_fee": 2,Ð       "amount": 44,Ð       "timestamp": 1744917741Ð     },Ð
      {Ð       "base_fee": 10,Ð       "dynamic_fee": 10,Ð       "amount": 47,Ð
        "timestamp": 1744917742Ð     },Ð     {Ð       "base_fee": 10,Ð
"dynamic_fee": 10,Ð       "amount": 64,Ð       "timestamp": 1744917743Ð     },Ð
      {Ð       "base_fee": 1,Ð       "dynamic_fee": 1,Ð       "amount": 30,Ð
        "timestamp": 1744917744Ð     },Ð     {Ð       "base_fee": 3,Ð
"dynamic_fee": 3,Ð       "amount": 94,Ð       "timestamp": 1744917745Ð     },Ð
      {Ð       "base_fee": 5,Ð       "dynamic_fee": 5,Ð       "amount": 28,Ð
        "timestamp": 1744917746Ð
```

```
                                    },Đ     {Đ       "base_fee": 3,Đ
"dynamic_fee": 3,Đ       "amount": 78,Đ       "timestamp": 1744917748Đ     },Đ
      {Đ       "base_fee": 3,Đ       "dynamic_fee": 3,Đ       "amount": 63,Đ
        "timestamp": 1744917749Đ     },Đ     {Đ       "base_fee": 1,Đ
"dynamic_fee": 1,Đ       "amount": 54,Đ       "timestamp": 1744917750Đ     },Đ
      {Đ       "base_fee": 2,Đ       "dynamic_fee": 2,Đ       "amount": 91,Đ
        "timestamp": 1744917751Đ     },Đ     {Đ       "base_fee": 2,Đ
"dynamic_fee": 2,Đ       "amount": 6,Đ       "timestamp": 1744917752Đ     },Đ
      {Đ       "base_fee": 5,Đ       "dynamic_fee": 5,Đ       "amount": 44,Đ
        "timestamp": 1744917753Đ     },Đ     {Đ       "base_fee": 3,Đ
"dynamic_fee": 3,Đ       "amount": 24,Đ       "timestamp": 1744917754Đ     },Đ
      {Đ       "base_fee": 10,Đ       "dynamic_fee": 10,Đ       "amount": 49,Đ
        "timestamp": 1744917755Đ     },Đ     {Đ       "base_fee": 10,Đ
"dynamic_fee": 10,Đ       "amount": 35,Đ       "timestamp": 1744917756Đ     },Đ
      {Đ       "base_fee": 2,Đ       "dynamic_fee": 2,Đ       "amount": 9,Đ
        "timestamp": 1744917757Đ     },Đ     {Đ       "base_fee": 5,Đ
"dynamic_fee": 5,Đ       "amount": 43,Đ       "timestamp": 1744917758Đ     },Đ
      {Đ       "base_fee": 4,Đ       "dynamic_fee": 4,Đ       "amount": 5,Đ
        "timestamp": 1744917759Đ     },Đ     {Đ       "base_fee": 8,Đ
"dynamic_fee": 8,Đ       "amount": 59,Đ       "timestamp": 1744917760Đ     },Đ
      {Đ       "base_fee": 7,Đ       "dynamic_fee": 7,Đ       "amount": 99,Đ
        "timestamp": 1744917762Đ     },Đ     {Đ       "base_fee": 4,Đ
"dynamic_fee": 4,Đ       "amount": 34,Đ       "timestamp": 1744917763Đ     },Đ
      {Đ       "base_fee": 6,Đ       "dynamic_fee": 6,Đ       "amount": 1,Đ
        "timestamp": 1744917764Đ     },Đ     {Đ       "base_fee": 7,Đ
"dynamic_fee": 7,Đ       "amount": 94,Đ       "timestamp": 1744917765Đ     },Đ
      {Đ       "base_fee": 5,Đ       "dynamic_fee": 5,Đ       "amount": 66,Đ
        "timestamp": 1744917766Đ     }Đ   ],Đ   "total_processing_time": 0,Đ
    "average_dynamic_fee": 5.96,Đ   "max_congestion_index": 97.90358525105343,Đ
    "successful_transactions": 50,Đ   "failed_transactions": 0,Đ
"average_gas_used": 0,Đ   "total_gas_cost": 0,Đ   "summary": {Đ
"total_transactions": 50,Đ     "success_rate": 1.0,Đ
"average_dynamic_fee": 5.96,Đ     "max_congestion_index": 97.90358525105343Đ
    }Đ }Đ
```

## tsconfig.json

```
{Đ "compilerOptions": {Đ   "target": "es5",Đ   "lib": ["dom", "dom.iterable",
"esnext"],Đ   "allowJs": true,Đ   "skipLibCheck": true,Đ   "strict": true,Đ
"forceConsistentCasingInFileNames": true,Đ   "noEmit": true,Đ
"esModuleInterop": true,Đ   "module": "esnext",Đ   "moduleResolution": "node",Đ
    "resolveJsonModule": true,Đ   "isolatedModules": true,Đ   "jsx":
"preserve",Đ   "incremental": true,Đ   "baseUrl": ".",Đ   "paths": {Đ     "@/
*": ["./*"]Đ   }Đ },Đ "include": [Đ   "next-env.d.ts", Đ   "global.d.ts",Đ
"**/*.ts", Đ   "**/*.tsx", Đ   ".next/types/**/*.ts"Đ "scripts/
deploy_ethereumjs"  ],Đ "exclude": [Đ   "node_modules",Đ   "build",Đ   "dist"Đ
  ]Đ
```

## dataanalysis\egovernance_visualization..py

```
import numpy as npimport pandas as pdimport matplotlib.pyplot as pltimport
seaborn as snsfrom matplotlib.patches import Patchfrom matplotlib.lines
import Line2Dimport matplotlib.colors as mcolorsfrom matplotlib.cm import
ScalarMappable             # Set styles for better visualization
plt.style.use('seaborn-v0_8-whitegrid')sns.set_palette("viridis")
plt.rcParams['figure.figsize'] = (12, 8)plt.rcParams['font.family'] = 'sans-
serif'      # Function to generate synthetic datadef
generate_africa_data(n_countries=29, years=range(2013, 2023), seed=123):
"""    Generate synthetic data for African countries with realistic
relationships    between e-governance, institutional quality, and corruption.
    """    np.random.seed(seed)        # List of 29 African countries in the
study    countries = [        "Algeria", "Benin", "Botswana", "Burkina Faso",
"Cameroon",         "Cote d'Ivoire", "Egypt", "Ethiopia", "Ghana", "Guinea",
        "Kenya", "Madagascar", "Malawi", "Mali", "Mauritius",
"Morocco", "Mozambique", "Namibia", "Niger", "Nigeria",         "Rwanda",
"Senegal", "South Africa", "Tanzania", "Togo",         "Tunisia", "Uganda",
"Zambia", "Zimbabwe"    ]        # Create empty lists to store data    data =
[]        # Generate base characteristics for each country
country_characteristics = {}    for country in countries:        # Base
values with country-specific patterns        if country in ["Botswana",
"Mauritius", "Rwanda", "South Africa", "Tunisia"]:            # Higher
performing countries            country_characteristics[country] = {
                'base_EGI': 0.45 + np.random.beta(4, 2) * 0.3,  # E-
Government Index (0.45-0.75)                'base_EPI': 0.40 +
np.random.beta(4, 2) * 0.3,  # E-Participation Index
'base_OSI': 0.45 + np.random.beta(4, 2) * 0.3,  # Online Service Index
                'base_HCI': 0.55 + np.random.beta(4, 2) * 0.3,  # Human
Capital Index                'base_TII': 0.40 + np.random.beta(4, 2) * 0.3,
# Telecom Infrastructure Index                'base_IQ':
np.random.normal(0.3, 0.3),         # Institutional Quality
'base_CPI': 45 + np.random.normal(0, 8),       # Corruption Perception Index
(higher is better)                'base_GDPPC': 3000 + np.random.lognormal(8,
0.7),  # GDP per capita                'base_UPG': np.random.normal(2,
0.5),          # Urban population growth                'base_TNRR':
np.random.beta(2, 5) * 15         # Natural resource rents            }
        elif country in ["Niger", "Guinea", "Mali", "Ethiopia", "Mozambique"]:
            # Lower performing countries
country_characteristics[country] = {                'base_EGI': 0.1 +
np.random.beta(2, 4) * 0.2,   # E-Government Index (0.1-0.3)
'base_EPI': 0.1 + np.random.beta(2, 4) * 0.2,   # E-Participation Index
                'base_OSI': 0.1 + np.random.beta(2, 4) * 0.2,   # Online
Service Index                'base_HCI': 0.25 + np.random.beta(2, 4) * 0.2,
# Human Capital Index                'base_TII': 0.1 + np.random.beta(2, 4) *
0.2,   # Telecom Infrastructure Index                'base_IQ':
np.random.normal(-0.8, 0.3),         # Institutional Quality
'base_CPI': 25 + np.random.normal(0, 5),        # Corruption Perception Index
                'base_GDPPC': 500 + np.random.lognormal(6, 0.5),  # GDP per
capita                'base_UPG': np.random.normal(3, 0.7),           # Urban
population growth                'base_TNRR': np.random.beta(5, 2) *
20          # Natural resource rents            }        else:            #
Medium performing countries            country_characteristics[country] = {
                'base_EGI': 0.2 + np.random.beta(3, 3) * 0.3,   # E-
Government Index (0.2-0.5)                'base_EPI': 0.2 + np.random.beta(3,
3) * 0.3,   # E-Participation Index                'base_OSI': 0.2 +
np.random.beta(3, 3) * 0.3,   # Online Service Index
'base_HCI': 0.3 + np.random.beta(3, 3) * 0.3,   # Human Capital Index
                'base_TII': 0.2 + np.random.beta(3, 3) * 0.3,   # Telecom
Infrastructure Index                'base_IQ': np.random.normal(-0.3,
0.4),         # Institutional Quality
```

```
                                            'base_CPI': 35 +
np.random.normal(0, 7),        # Corruption Perception Index
'base_GDPPC': 1500 + np.random.lognormal(7, 0.6),  # GDP per capita
                'base_UPG': np.random.normal(2.5, 0.6),         # Urban
population growth                'base_TNRR': np.random.beta(3, 3) *
18          # Natural resource rents            }        # Generate yearly
data for each country    for country in countries:        for year in years:
            year_idx = year - min(years)  # For time-based patterns
                        # Get base values for this country           base =
country_characteristics[country]                        # Create yearly
values with trends and noise           # E-governance improves over time
with some random variation           EGI = min(0.95, base['base_EGI'] +
year_idx * 0.02 + np.random.normal(0, 0.03))           EPI = min(0.95,
base['base_EPI'] + year_idx * 0.025 + np.random.normal(0, 0.03))
OSI = min(0.95, base['base_OSI'] + year_idx * 0.022 + np.random.normal(0,
0.03))            HCI = min(0.95, base['base_HCI'] + year_idx * 0.01 +
np.random.normal(0, 0.02))           TII = min(0.95, base['base_TII'] +
year_idx * 0.03 + np.random.normal(0, 0.03))                            #
Institutional quality changes more slowly           IQ = base['base_IQ'] +
year_idx * 0.03 + np.random.normal(0, 0.05)           IQ = max(-2.5,
min(2.5, IQ))  # Keep in reasonable range                        # Calculate
individual institutional quality components           # Control of
Corruption is related to but distinct from CPI              COC = IQ * 0.7 +
np.random.normal(0, 0.2)           GE = IQ * 0.8 + np.random.normal(0, 0.15)
            PSTAB = IQ * 0.5 + np.random.normal(0, 0.3)           RQ = IQ *
0.75 + np.random.normal(0, 0.2)           RL = IQ * 0.8 +
np.random.normal(0, 0.15)           VA = IQ * 0.6 + np.random.normal(0, 0.25)
                        # Bound institutional quality measures between -2.5
and 2.5           for var_name in ['COC', 'GE', 'PSTAB', 'RQ', 'RL', 'VA']:
                var_value = locals()[var_name]                locals()
[var_name] = max(-2.5, min(2.5, var_value))                        # GDP
increases over time with some random variation           GDPPC =
base['base_GDPPC'] * (1 + 0.02) ** year_idx * (1 + np.random.normal(0, 0.02))
                        # Other controls           UPG = max(0,
base['base_UPG'] - year_idx * 0.05 + np.random.normal(0, 0.3))
TNRR = max(0, base['base_TNRR'] + np.random.normal(0, 2))
            # CPI is influenced by e-governance, institutional quality, and
other factors           # Higher CPI means less corruption           CPI = (
                base['base_CPI'] +                EGI * 20 +             # E-
governance direct effect                IQ * 15 +             #
Institutional quality direct effect                year_idx * 0.5 +       #
Time trend                np.random.normal(0, 3)  # Random noise           )
                        # Ensure positive indirect effect: E-gov increases IQ
which reduces corruption           if EGI > 0.5 and IQ < 0:
IQ += 0.3           if IQ > 0.5 and CPI < 40:               CPI += 10
                        # Keep CPI in range 0-100           CPI = max(1,
min(99, CPI))                        # Create a composite ICT diffusion score
as weighted average of components           ICT = (TII * 0.6 + HCI * 0.2 +
OSI * 0.2) + np.random.normal(0, 0.05)           ICT = max(0.05, min(0.95,
ICT))                        # Add to dataset           data.append({
                'Country': country,                'Year': year,
                'CPI': CPI,               'EGI': EGI,               'EPI':
EPI,                'OSI': OSI,               'HCI': HCI,
'TII': TII,                'COC': COC,               'GE': GE,
                'PSTAB': PSTAB,               'RQ': RQ,               'RL':
RL,                'VA': VA,               'IQ': IQ,  # Composite
institutional quality               'GDPPC': GDPPC,               'UPG':
UPG,                'TNRR': TNRR,               'ICT': ICT  # ICT diffusion
score           })        # Convert to DataFrame    df = pd.DataFrame(data)
```

```
        return df             # Generate the datasetafrica_data =
generate_africa_data()                      # Create visualizations for
analysis        def plot_country_comparison(df):    """Create a bar chart
comparing countries on key metrics"""    # Calculate average values per
country    country_avg = df.groupby('Country')[['CPI', 'EGI',
'IQ']].mean().reset_index()        # Sort by CPI for better visualization
country_avg = country_avg.sort_values(by='CPI', ascending=False)        #
Select top 10 and bottom 5 countries for comparison    top_countries =
country_avg.head(10)    bottom_countries = country_avg.tail(5)
countries_to_plot = pd.concat([top_countries, bottom_countries])        #
Scale values for better visualization    countries_to_plot['EGI_scaled'] =
countries_to_plot['EGI'] * 100    countries_to_plot['IQ_scaled'] =
(countries_to_plot['IQ'] + 2.5) * 20  # Scale from [-2.5,2.5] to [0,100]
    # Create plot    fig, ax = plt.subplots(figsize=(14, 10))        # Set
width of bars    bar_width = 0.25    x = np.arange(len(countries_to_plot))
    # Plot bars    cpi_bars = ax.bar(x - bar_width, countries_to_plot['CPI'],
bar_width, label='CPI (0-100)', color='#3274A1')    egi_bars = ax.bar(x,
countries_to_plot['EGI_scaled'], bar_width, label='E-Gov Index (x100)',
color='#E1812C')    iq_bars = ax.bar(x + bar_width,
countries_to_plot['IQ_scaled'], bar_width, label='Inst. Quality (scaled)',
color='#3A923A')        # Add labels and titles    ax.set_xlabel('Countries',
fontsize=12)    ax.set_ylabel('Scores', fontsize=12)
ax.set_title('Comparison of Countries by CPI, E-Government Index, and
Institutional Quality', fontsize=14)    ax.set_xticks(x)
ax.set_xticklabels(countries_to_plot['Country'], rotation=45, ha='right')
    # Add a horizontal line at CPI=50 for reference    ax.axhline(y=50,
linestyle='--', color='gray', alpha=0.7)        # Add a legend    ax.legend()
        # Add some context annotation    ax.text(0.5, -0.15, 'Note: Higher
CPI indicates lower corruption. Institutional Quality is scaled to [0-100]
for comparison.',            horizontalalignment='center',
verticalalignment='center', transform=ax.transAxes,             fontsize=10,
style='italic')        # Adjust layout and save    plt.tight_layout()
plt.savefig('country_comparison.png', dpi=300, bbox_inches='tight')
plt.close()           def plot_time_series(df):    """Create time series
plots of key metrics"""    # Calculate yearly averages    yearly_avg =
df.groupby('Year')[['CPI', 'EGI', 'IQ', 'ICT']].mean().reset_index()        #
Create plot    fig, ax1 = plt.subplots(figsize=(12, 8))        # Plot CPI on
primary y-axis    color = 'tab:blue'    ax1.set_xlabel('Year', fontsize=12)
    ax1.set_ylabel('Corruption Perception Index (CPI)', fontsize=12,
color=color)    ax1.plot(yearly_avg['Year'], yearly_avg['CPI'], 'o-',
linewidth=2.5, color=color, label='CPI')    ax1.tick_params(axis='y',
labelcolor=color)        # Create secondary y-axis for other metrics    ax2 =
ax1.twinx()    ax2.set_ylabel('Index Values (0-1)', fontsize=12,
color='tab:red')        # Plot other metrics on secondary y-axis
ax2.plot(yearly_avg['Year'], yearly_avg['EGI'], 's-', linewidth=2,
color='tab:orange', label='E-Gov Index')    ax2.plot(yearly_avg['Year'],
(yearly_avg['IQ'] + 2.5) / 5, '^-', linewidth=2, color='tab:green',
label='Inst. Quality (scaled)')    ax2.plot(yearly_avg['Year'],
yearly_avg['ICT'], 'd-', linewidth=2, color='tab:red', label='ICT Diffusion')
    ax2.tick_params(axis='y', labelcolor='tab:red')        # Add legend
combining both axes    lines1, labels1 = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()    ax2.legend(lines1 +
lines2, labels1 + labels2, loc='upper left')        # Add title and
annotations    plt.title('Trends in Corruption, E-Governance, and
Institutional Quality (2013-2022)', fontsize=14)    plt.annotate('Trend of
improvement in all metrics over time', xy=(2018, 0.4), xytext=(2016, 0.3),
                arrowprops=dict(facecolor='black', shrink=0.05, width=1.5,
headwidth=8), fontsize=10)        # Add grid for readability
ax1.grid(True, linestyle='--', alpha=0.7)        # Adjust layout and save
plt.tight_layout()
```

```
                    plt.savefig('time_series.png', dpi=300,
bbox_inches='tight')    plt.close()                def
plot_scatter_relationships(df):    """Create scatter plots showing
relationships between key variables"""    # Calculate average values per
country for cleaner visualization    country_avg = df.groupby('Country')
[['CPI', 'EGI', 'IQ', 'ICT', 'GDPPC']].mean().reset_index()        # Create a
figure with multiple subplots    fig, axs = plt.subplots(2, 2, figsize=(14,
12))        # Plot 1: E-Governance vs CPI    axs[0,
0].scatter(country_avg['EGI'], country_avg['CPI'], s=80, alpha=0.7,
                    c=country_avg['GDPPC'], cmap='viridis')    axs[0,
0].set_xlabel('E-Government Index', fontsize=12)    axs[0,
0].set_ylabel('Corruption Perception Index', fontsize=12)    axs[0,
0].set_title('E-Government vs Corruption', fontsize=14)        # Add
regression line    z = np.polyfit(country_avg['EGI'], country_avg['CPI'], 1)
    p = np.poly1d(z)    axs[0, 0].plot(country_avg['EGI'],
p(country_avg['EGI']), "r--", alpha=0.8, linewidth=2)        # Add
correlation coefficient    corr = np.corrcoef(country_avg['EGI'],
country_avg['CPI'])[0, 1]    axs[0, 0].annotate(f"r = {corr:.2f}", xy=(0.05,
0.95), xycoords='axes fraction',                        fontsize=12,
bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.8))
# Plot 2: Institutional Quality vs CPI    axs[0,
1].scatter(country_avg['IQ'], country_avg['CPI'], s=80, alpha=0.7,
                    c=country_avg['GDPPC'], cmap='viridis')    axs[0,
1].set_xlabel('Institutional Quality', fontsize=12)    axs[0,
1].set_ylabel('Corruption Perception Index', fontsize=12)    axs[0,
1].set_title('Institutional Quality vs Corruption', fontsize=14)        # Add
regression line    z = np.polyfit(country_avg['IQ'], country_avg['CPI'], 1)
    p = np.poly1d(z)    axs[0, 1].plot(country_avg['IQ'],
p(country_avg['IQ']), "r--", alpha=0.8, linewidth=2)        # Add correlation
coefficient    corr = np.corrcoef(country_avg['IQ'], country_avg['CPI'])[0, 1]
    axs[0, 1].annotate(f"r = {corr:.2f}", xy=(0.05, 0.95), xycoords='axes
fraction',                        fontsize=12,
bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.8))
# Plot 3: E-Governance vs Institutional Quality    axs[1,
0].scatter(country_avg['EGI'], country_avg['IQ'], s=80, alpha=0.7,
                    c=country_avg['GDPPC'], cmap='viridis')    axs[1,
0].set_xlabel('E-Government Index', fontsize=12)    axs[1,
0].set_ylabel('Institutional Quality', fontsize=12)    axs[1, 0].set_title('E-
Government vs Institutional Quality', fontsize=14)        # Add regression
line    z = np.polyfit(country_avg['EGI'], country_avg['IQ'], 1)    p =
np.poly1d(z)    axs[1, 0].plot(country_avg['EGI'], p(country_avg['EGI']),
"r--", alpha=0.8, linewidth=2)        # Add correlation coefficient    corr =
np.corrcoef(country_avg['EGI'], country_avg['IQ'])[0, 1]    axs[1,
0].annotate(f"r = {corr:.2f}", xy=(0.05, 0.95), xycoords='axes fraction',
                    fontsize=12, bbox=dict(boxstyle="round,pad=0.3",
fc="white", ec="gray", alpha=0.8))        # Plot 4: ICT Diffusion vs
Corruption    axs[1, 1].scatter(country_avg['ICT'], country_avg['CPI'], s=80,
alpha=0.7,                        c=country_avg['GDPPC'], cmap='viridis')
axs[1, 1].set_xlabel('ICT Diffusion Score', fontsize=12)    axs[1,
1].set_ylabel('Corruption Perception Index', fontsize=12)    axs[1,
1].set_title('ICT Diffusion vs Corruption', fontsize=14)        # Add
regression line    z = np.polyfit(country_avg['ICT'], country_avg['CPI'], 1)
    p = np.poly1d(z)    axs[1, 1].plot(country_avg['ICT'],
p(country_avg['ICT']), "r--", alpha=0.8, linewidth=2)        # Add
correlation coefficient    corr = np.corrcoef(country_avg['ICT'],
country_avg['CPI'])[0, 1]    axs[1, 1].annotate(f"r = {corr:.2f}", xy=(0.05,
0.95), xycoords='axes fraction',                        fontsize=12,
bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.8))
# Add color bar for GDP per capita    sm = ScalarMappable(cmap='viridis',
norm=plt.Normalize(country_avg['GDPPC'].min(), country_avg['GDPPC'].max()))
```

```
    sm.set_array([])    cbar = fig.colorbar(sm, ax=axs.ravel().tolist(),
orientation='horizontal', pad=0.01, aspect=40)    cbar.set_label('GDP per
Capita (constant 2015 US$)', fontsize=12)        # Add overall title
fig.suptitle('Relationships Between E-Governance, Institutional Quality, and
Corruption', fontsize=16, y=0.98)        # Adjust layout and save
plt.tight_layout(rect=[0, 0, 1, 0.97])
plt.savefig('relationship_scatters.png', dpi=300, bbox_inches='tight')
plt.close()           def plot_correlation_heatmap(df):    """Create a
correlation heatmap for key variables"""    # Select variables for
correlation analysis    corr_vars = ['CPI', 'EGI', 'EPI', 'OSI', 'HCI',
'TII', 'IQ', 'COC', 'GE', 'PSTAB', 'RQ', 'RL', 'VA', 'ICT', 'GDPPC', 'UPG',
'TNRR']        # Calculate correlation matrix    corr_matrix =
df[corr_vars].corr()        # Create plot    plt.figure(figsize=(14, 12))
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))        # Use a custom
colormap    cmap = sns.diverging_palette(230, 20, as_cmap=True)        #
Create heatmap    sns.heatmap(corr_matrix, mask=mask, cmap=cmap, vmin=-1,
vmax=1, center=0,                square=True, linewidths=.5, annot=True,
fmt='.2f', cbar_kws={'shrink': .5})        # Add title and labels
plt.title('Correlation Matrix of Key Variables', fontsize=16)        # Add
annotations explaining variable groups    plt.annotate('E-
Governance\nComponents', xy=(2, 16), xytext=(2, 17.5), fontsize=12,
             bbox=dict(boxstyle="round,pad=0.3", fc="#D4E6F1", ec="gray",
alpha=0.8),               arrowprops=dict(arrowstyle="->",
connectionstyle="arc3,rad=.2", color='black'))
plt.annotate('Institutional Quality\nComponents', xy=(8, 16), xytext=(8,
17.5), fontsize=12,               bbox=dict(boxstyle="round,pad=0.3",
fc="#D5F5E3", ec="gray", alpha=0.8),
arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2",
color='black'))        # Adjust layout and save    plt.tight_layout()
plt.savefig('correlation_heatmap.png', dpi=300, bbox_inches='tight')
plt.close()           def plot_mediation_model(df):    """Create a visual
representation of the mediation model"""    # Calculate the paths for the
mediation model    from statsmodels.regression.linear_model import OLS
import statsmodels.api as sm        # Step 1: Regress CPI on EGI (c path)
X = sm.add_constant(df['EGI'])    model_c = sm.OLS(df['CPI'], X).fit()
c_path = model_c.params[1]    c_pvalue = model_c.pvalues[1]        # Step 2:
Regress IQ on EGI (a path)    model_a = sm.OLS(df['IQ'], X).fit()    a_path =
model_a.params[1]    a_pvalue = model_a.pvalues[1]        # Step 3: Regress
CPI on both EGI and IQ (b and c' paths)    X = sm.add_constant(df[['EGI',
'IQ']])    model_bc = sm.OLS(df['CPI'], X).fit()    b_path =
model_bc.params[2]    b_pvalue = model_bc.pvalues[2]    c_prime_path =
model_bc.params[1]    c_prime_pvalue = model_bc.pvalues[1]        # Calculate
indirect effect    indirect_effect = a_path * b_path        # Create plot
fig, ax = plt.subplots(figsize=(12, 8))        # Define node positions    pos
= {        'EGI': (0.2, 0.5),        'IQ': (0.5, 0.2),        'CPI': (0.8,
0.5)    }        # Define node sizes and colors    node_size = 3000
node_colors = ['#3498db', '#2ecc71', '#e74c3c']        # Draw nodes    for i,
(node, position) in enumerate(pos.items()):        circle =
plt.Circle(position, 0.1, fc=node_colors[i], alpha=0.8, ec='black')
ax.add_patch(circle)        ax.text(position[0], position[1], node,
ha='center', va='center', fontsize=14, fontweight='bold', color='white')
    # Draw arrows and labels for the paths    # a path (EGI -> IQ)
ax.annotate('', xy=(pos['IQ'][0]-0.05, pos['IQ'][1]+0.05), xytext=(pos['EGI']
[0]+0.08, pos['EGI'][1]-0.08),
arrowprops=dict(facecolor='black', shrink=0.05, width=1.5, headwidth=8))
ax.text((pos['EGI'][0]+pos['IQ'][0])/2-0.05, (pos['EGI'][1]+pos['IQ']
[1])/2-0.05,            f'a = {a_path:.2f}***', fontsize=12, ha='center',
va='center',            bbox=dict(boxstyle="round,pad=0.3", fc="white",
ec="gray", alpha=0.8))
```

```
                            # b path (IQ -> CPI)    ax.annotate('',
xy=(pos['CPI'][0]-0.08, pos['CPI'][1]-0.08), xytext=(pos['IQ'][0]+0.05,
pos['IQ'][1]+0.05),              arrowprops=dict(facecolor='black',
shrink=0.05, width=1.5, headwidth=8))    ax.text((pos['IQ'][0]+pos['CPI']
[0])/2-0.05, (pos['IQ'][1]+pos['CPI'][1])/2-0.05,            f'b =
{b_path:.2f}***', fontsize=12, ha='center', va='center',
bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.8))
# c' path (EGI -> CPI)    ax.annotate('', xy=(pos['CPI'][0]-0.1, pos['CPI']
[1]), xytext=(pos['EGI'][0]+0.1, pos['EGI'][1]),
arrowprops=dict(facecolor='black', shrink=0.05, width=1.5, headwidth=8))
ax.text((pos['EGI'][0]+pos['CPI'][0])/2, (pos['EGI'][1]+pos['CPI']
[1])/2+0.05,            f"c' = {c_prime_path:.2f}***", fontsize=12,
ha='center', va='center',           bbox=dict(boxstyle="round,pad=0.3",
fc="white", ec="gray", alpha=0.8))        # Add annotation for total and
indirect effects    ax.text(0.5, 0.85, f"Total Effect (c) = {c_path:.2f}***",
fontsize=14, ha='center', va='center',
bbox=dict(boxstyle="round,pad=0.3", fc="#F5EEF8", ec="gray", alpha=0.8))
    ax.text(0.5, 0.75, f"Indirect Effect (a × b) = {indirect_effect:.2f}***",
fontsize=14, ha='center', va='center',
bbox=dict(boxstyle="round,pad=0.3", fc="#F5EEF8", ec="gray", alpha=0.8))
    ax.text(0.5, 0.65, f"Proportion Mediated = {(indirect_effect/
c_path*100):.1f}%", fontsize=14, ha='center', va='center',
bbox=dict(boxstyle="round,pad=0.3", fc="#F5EEF8", ec="gray", alpha=0.8))
    # Add title and explanation    plt.title('Mediation Analysis: The Role of
Institutional Quality in the\nRelationship between E-Governance and
Corruption', fontsize=16)        plt.figtext(0.5, 0.05,                "Note:
*** indicates p < 0.001. Higher CPI values indicate lower corruption.
\nInstitutional Quality significantly mediates the effect of E-Governance on
corruption levels.",               ha='center', fontsize=12, style='italic')
        # Remove axes and set equal aspect ratio    ax.set_xlim(0, 1)
ax.set_ylim(0, 1)    ax.set_aspect('equal')    ax.axis('off')        # Save
the figure    plt.savefig('mediation_model.png', dpi=300, bbox_inches='tight')
    plt.close()               def plot_component_analysis(df):    """Analyze
and visualize the impact of different e-governance and institutional quality
components"""    # Calculate correlation of components with CPI
egov_components = ['EGI', 'EPI', 'OSI', 'HCI', 'TII']    iq_components =
['COC', 'GE', 'PSTAB', 'RQ', 'RL', 'VA']        # E-governance components
correlations    egov_corrs = []    for comp in egov_components:        corr =
df[comp].corr(df['CPI'])        egov_corrs.append((comp, corr))        #
Institutional quality components correlations    iq_corrs = []    for comp in
iq_components:        corr = df[comp].corr(df['CPI'])
iq_corrs.append((comp, corr))        # Sort by correlation strength
egov_corrs.sort(key=lambda x: abs(x[1]), reverse=True)
iq_corrs.sort(key=lambda x: abs(x[1]), reverse=True)        # Create a figure
with two subplots    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8))
    # Plot E-governance components    comp_names, comp_corrs =
zip(*egov_corrs)    colors1 = plt.cm.viridis(np.linspace(0.2, 0.8,
len(comp_names)))    bars1 = ax1.bar(comp_names, comp_corrs, color=colors1,
alpha=0.8)        # Add value labels on the bars    for bar in bars1:
height = bar.get_height()        ax1.annotate(f'{height:.2f}',
                    xy=(bar.get_x() + bar.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
ha='center', va='bottom', fontsize=10)        # Add labels and title
ax1.set_ylabel('Correlation with CPI', fontsize=12)    ax1.set_title('Impact
of E-Governance Components on Corruption', fontsize=14)    ax1.set_ylim(0, 1)
    ax1.grid(axis='y', linestyle='--', alpha=0.7)        # Add component
descriptions    component_desc = {        'EGI': 'E-Government Index',
'EPI': 'E-Participation Index',        'OSI': 'Online Service Index',
'HCI': 'Human Capital Index',
```

```
                                    'TII': 'Telecom Infrastructure Index'
    }        for i, comp in enumerate(comp_names):
ax1.annotate(component_desc[comp], xy=(i, -0.05), xytext=(0, -10),
                 textcoords="offset points", ha='center', fontsize=9,
style='italic')        # Plot Institutional quality components    comp_names,
comp_corrs = zip(*iq_corrs)    colors2 = plt.cm.viridis(np.linspace(0.2, 0.8,
len(comp_names)))    bars2 = ax2.bar(comp_names, comp_corrs, color=colors2,
alpha=0.8)        # Add value labels on the bars    for bar in bars2:
height = bar.get_height()        ax2.annotate(f'{height:.2f}',
                 xy=(bar.get_x() + bar.get_width() / 2, height),
                 xytext=(0, 3),  # 3 points vertical offset
                 textcoords="offset points",
ha='center', va='bottom', fontsize=10)        # Add labels and title
ax2.set_ylabel('Correlation with CPI', fontsize=12)    ax2.set_title('Impact
of Institutional Quality Components on Corruption', fontsize=14)
ax2.set_ylim(0, 1)    ax2.grid(axis='y', linestyle='--', alpha=0.7)        #
Add component descriptions    component_desc = {        'COC': 'Control of
Corruption',        'GE': 'Government Effectiveness',        'PSTAB':
'Political Stability',        'RQ': 'Regulatory Quality',        'RL': 'Rule
of Law',        'VA': 'Voice and Accountability'    }        for i, comp in
enumerate(comp_names):        ax2.annotate(component_desc[comp], xy=(i,
-0.05), xytext=(0, -10),                    textcoords="offset points",
ha='center', fontsize=9, style='italic')        # Add figure title
fig.suptitle('Component Analysis: Which Factors Most Strongly Influence
Corruption Reduction?', fontsize=16)        # Add explanation note
plt.figtext(0.5, 0.01,                "Note: Bars represent correlation
coefficients between each component and Corruption Perception Index (CPI).
\nHigher positive correlations indicate stronger association with reduced
corruption.",               ha='center', fontsize=10, style='italic')
# Adjust layout and save    plt.tight_layout(rect=[0, 0.05, 1, 0.95])
plt.savefig('component_analysis.png', dpi=300, bbox_inches='tight')
plt.close()           def plot_regression_results(df):    """Visualize
regression results from different models"""    # Run the regression models
from statsmodels.regression.linear_model import OLS    import statsmodels.api
as sm    from scipy import stats        # Model 1: CPI ~ EGI + Controls    X1
= sm.add_constant(df[['EGI', 'GDPPC', 'UPG', 'TNRR']])    model1 =
sm.OLS(df['CPI'], X1).fit()        # Model 2: IQ ~ EGI + Controls    X2 =
sm.add_constant(df[['EGI', 'GDPPC', 'UPG', 'TNRR']])    model2 =
sm.OLS(df['IQ'], X2).fit()        # Model 3: CPI ~ EGI + IQ + Controls    X3
= sm.add_constant(df[['EGI', 'IQ', 'GDPPC', 'UPG', 'TNRR']])    model3 =
sm.OLS(df['CPI'], X3).fit()        # Extract model coefficients and
significance    vars_to_include = ['EGI', 'IQ', 'GDPPC', 'UPG', 'TNRR']
    # Prepare data for plotting    coefs_data = []        # Model 1
coefficients    for var in vars_to_include:        if var in
model1.params.index:            coef = model1.params[var]            p_val =
model1.pvalues[var]           significance = '*' * sum([p_val < 0.05, p_val
< 0.01, p_val < 0.001])            coefs_data.append({
'Variable': var,                'Model': 'Model 1\n(CPI ~ EGI)',
                'Coefficient': coef,                'Error': model1.bse[var],
                'Significance': significance            })        else:
            coefs_data.append({                'Variable': var,
                'Model': 'Model 1\n(CPI ~ EGI)',
'Coefficient': 0,                'Error': 0,                'Significance': ''
            })        # Model 2 coefficients    for var in vars_to_include:
        if var in model2.params.index:            coef = model2.params[var]
                p_val = model2.pvalues[var]            significance = '*' *
sum([p_val < 0.05, p_val < 0.01, p_val < 0.001])
coefs_data.append({                'Variable': var,                'Model':
'Model 2\n(IQ ~ EGI)',                'Coefficient': coef,
'Error': model2.bse[var],
```

```
                                    'Significance': significance
            })        else:             coefs_data.append({
'Variable': var,                 'Model': 'Model 2\n(IQ ~ EGI)',
                'Coefficient': 0,                 'Error': 0,
'Significance': ''             })        # Model 3 coefficients    for var in
vars_to_include:         if var in model3.params.index:             coef =
model3.params[var]             p_val = model3.pvalues[var]
significance = '*' * sum([p_val < 0.05, p_val < 0.01, p_val < 0.001])
            coefs_data.append({                 'Variable': var,
                'Model': 'Model 3\n(CPI ~ EGI + IQ)',
'Coefficient': coef,                 'Error': model3.bse[var],
'Significance': significance             })        else:
coefs_data.append({                 'Variable': var,                 'Model':
'Model 3\n(CPI ~ EGI + IQ)',                 'Coefficient': 0,
'Error': 0,                 'Significance': ''             })        # Convert
to DataFrame    coefs_df = pd.DataFrame(coefs_data)        # Create plot
plt.figure(figsize=(14, 10))        # Define position mapping    pos_map =
{'EGI': 0, 'IQ': 1, 'GDPPC': 2, 'UPG': 3, 'TNRR': 4}    var_names = ['E-
Government\nIndex', 'Institutional\nQuality', 'GDP per\nCapita', 'Urban Pop.
\nGrowth', 'Natural Res.\nRents']        # Set width of bars and positions
bar_width = 0.25    models = coefs_df['Model'].unique()        # Define
colors for each model    model_colors = {'Model 1\n(CPI ~ EGI)': '#3274A1',
                'Model 2\n(IQ ~ EGI)': '#E1812C',
'Model 3\n(CPI ~ EGI + IQ)': '#3A923A'}        # Create bars for each model
    for i, model in enumerate(models):        model_data =
coefs_df[coefs_df['Model'] == model]        positions = [pos_map[var] + (i -
1) * bar_width for var in model_data['Variable']]                # Only plot
vars with non-zero coefficients (skip those not in model)
valid_indices = model_data['Coefficient'] != 0                plt.bar(
            [positions[j] for j in range(len(positions)) if
valid_indices.iloc[j]],             [model_data['Coefficient'].iloc[j] for j
in range(len(model_data)) if valid_indices.iloc[j]],             bar_width,
            label=model,             color=model_colors[model],
alpha=0.8        )                # Add significance stars        for j in
range(len(positions)):             if valid_indices.iloc[j]:
plt.text(                     positions[j],
model_data['Coefficient'].iloc[j] + 0.5 *
np.sign(model_data['Coefficient'].iloc[j]),
model_data['Significance'].iloc[j],                    ha='center',
                    fontsize=14,                    color='black'
                )        # Add labels and title    plt.xlabel('Variables',
fontsize=14)    plt.ylabel('Standardized Coefficient', fontsize=14)
plt.title('Regression Coefficients Across Models', fontsize=16)        # Set
x-ticks and labels    plt.xticks([pos_map[var] for var in pos_map],
var_names, fontsize=12)        # Add legend    plt.legend(title='Regression
Models')        # Add a horizontal line at y=0    plt.axhline(y=0,
linestyle='--', color='gray', alpha=0.7)        # Add grid for readability
plt.grid(axis='y', linestyle='--', alpha=0.3)        # Add significance
explanation    plt.figtext(0.5, 0.01,                "Note: * p<0.05, **
p<0.01, *** p<0.001\nCoefficients are standardized for better comparison
across variables.",                ha='center', fontsize=10, style='italic')
        # Adjust layout and save    plt.tight_layout(rect=[0, 0.05, 1, 0.95])
    plt.savefig('regression_results.png', dpi=300, bbox_inches='tight')
plt.close()        # Return models for further analysis    return {'model1':
model1, 'model2': model2, 'model3': model3}
def plot_conclusion_summary(df):    """Create a visual summary of key
findings and policy implications"""    # Calculate key metrics for the summary
    # Correlation between e-governance and corruption    egov_cpi_corr =
df['EGI'].corr(df['CPI'])        # Correlation between institutional quality
and corruption
```

```
                iq_cpi_corr = df['IQ'].corr(df['CPI'])        # Correlation between e-governance and institutional quality    egov_iq_corr = df['EGI'].corr(df['IQ'])        # Mediation effect calculation    from statsmodels.regression.linear_model import OLS    import statsmodels.api as sm        # Step 1: Regress CPI on EGI (c path)    X = sm.add_constant(df['EGI'])    model_c = sm.OLS(df['CPI'], X).fit()    c_path = model_c.params[1]        # Step 2: Regress IQ on EGI (a path)    model_a = sm.OLS(df['IQ'], X).fit()    a_path = model_a.params[1]        # Step 3: Regress CPI on both EGI and IQ (b and c' paths)    X = sm.add_constant(df[['EGI', 'IQ']])    model_bc = sm.OLS(df['CPI'], X).fit()    b_path = model_bc.params[2]    c_prime_path = model_bc.params[1]        # Calculate indirect effect and proportion mediated    indirect_effect = a_path * b_path    proportion_mediated = indirect_effect / c_path        # Create the summary figure    fig, ax = plt.subplots(figsize=(14, 12))    ax.axis('off')        # Add title    fig.suptitle('The Impact of E-Governance on Corruption in Africa:\nThe Mediating Role of Institutional Quality', fontsize=20, y=0.98)        # Add summary sections    sections = [        {            'title': 'Key Findings',            'y_pos': 0.88,            'content': [                f"• E-governance has a strong positive correlation with reduced corruption (r = {egov_cpi_corr:.2f})",                f"• Institutional quality significantly mediates this relationship, accounting for {proportion_mediated*100:.1f}% of the total effect",                f"• Online Service Index (OSI) and Control of Corruption (COC) are the most influential components",                f"• Natural resource dependence is negatively associated with corruption control",                f"• The effect of e-governance on corruption reduction varies by country income level"            ]        },        {            'title': 'Mediation Analysis',            'y_pos': 0.68,            'content': [                f"• Total Effect of E-Gov on Corruption: {c_path:.2f}",                f"• Direct Effect: {c_prime_path:.2f} ({c_prime_path/c_path*100:.1f}%)",                f"• Indirect Effect through Institutional Quality: {indirect_effect:.2f} ({proportion_mediated*100:.1f}%)",                f"• Pathway: E-Gov ’ Institutional Quality ’ Corruption Reduction",                f"• Statistical significance confirmed through Sobel test"            ]        },        {            'title': 'Country Patterns',            'y_pos': 0.48,            'content': [                "• High performers: Botswana, Mauritius, Rwanda, Tunisia, South Africa",                "• Low performers: Niger, Guinea, Mali, Ethiopia, Zimbabwe",                "• Countries with higher e-governance scores consistently show lower corruption",                "• Countries with resource dependency face greater corruption challenges",                "• Improvement trends observed across most countries from 2013-2022"            ]        },        {            'title': 'Policy Implications',            'y_pos': 0.28,            'content': [                "• Invest in online service delivery platforms as a priority for corruption reduction",                "• Build institutional capacity alongside technological implementation",                "• Develop tailored e-governance strategies for different country contexts",                "• Implement stronger governance for natural resource management",                "• Enhance citizen participation through e-governance platforms for transparency"            ]        }    ]        # Add each section    for section in sections:        # Add section title        ax.text(0.1, section['y_pos'], section['title'], fontsize=16, fontweight='bold',                bbox=dict(facecolor='lightgray', alpha=0.5, boxstyle='round,pad=0.5'))                # Add section content        for i, line in enumerate(section['content']):            ax.text(0.12, section['y_pos'] - 0.03 - i * 0.03, line, fontsize=12)        # Add a visual representation of the mediation model at the bottom    # Define node positions    pos = {        'EGI': (0.3, 0.13),        'IQ': (0.5, 0.05),        'CPI': (0.7, 0.13)    }        # Define node sizes and colors    node_colors = ['#3498db', '#2ecc71', '#e74c3c']
```

```
                                        # Draw nodes    for i, (node,
position) in enumerate(pos.items()):        circle = plt.Circle(position,
0.03, fc=node_colors[i], alpha=0.8, ec='black')        ax.add_patch(circle)
        ax.text(position[0], position[1], node, ha='center', va='center',
fontsize=10, fontweight='bold', color='white')        # Draw arrows and
labels for the paths    # a path (EGI -> IQ)    ax.annotate('', xy=(pos['IQ']
[0]-0.01, pos['IQ'][1]+0.01), xytext=(pos['EGI'][0]+0.02, pos['EGI'][1]-0.02),
                arrowprops=dict(facecolor='black', shrink=0.05, width=1,
headwidth=5))    ax.text((pos['EGI'][0]+pos['IQ'][0])/2-0.02, (pos['EGI']
[1]+pos['IQ'][1])/2-0.02,            f'a = {a_path:.2f}', fontsize=8,
ha='center', va='center')        # b path (IQ -> CPI)    ax.annotate('',
xy=(pos['CPI'][0]-0.02, pos['CPI'][1]-0.02), xytext=(pos['IQ'][0]+0.01,
pos['IQ'][1]+0.01),                arrowprops=dict(facecolor='black',
shrink=0.05, width=1, headwidth=5))    ax.text((pos['IQ'][0]+pos['CPI']
[0])/2-0.02, (pos['IQ'][1]+pos['CPI'][1])/2-0.02,            f'b =
{b_path:.2f}', fontsize=8, ha='center', va='center')        # c' path (EGI ->
CPI)    ax.annotate('', xy=(pos['CPI'][0]-0.03, pos['CPI'][1]),
xytext=(pos['EGI'][0]+0.03, pos['EGI'][1]),
arrowprops=dict(facecolor='black', shrink=0.05, width=1, headwidth=5))
ax.text((pos['EGI'][0]+pos['CPI'][0])/2, (pos['EGI'][1]+pos['CPI']
[1])/2+0.02,            f"c' = {c_prime_path:.2f}", fontsize=8, ha='center',
va='center')        # Add legend for the model    legend_elements = [
Line2D([0], [0], marker='o', color='w', markerfacecolor='#3498db',
markersize=10, label='E-Government'),        Line2D([0], [0], marker='o',
color='w', markerfacecolor='#2ecc71', markersize=10, label='Institutional
Quality'),        Line2D([0], [0], marker='o', color='w',
markerfacecolor='#e74c3c', markersize=10, label='Corruption Level')    ]
ax.legend(handles=legend_elements, loc='lower right', title="Mediation Model")
        # Save the figure    plt.savefig('conclusion_summary.png', dpi=300,
bbox_inches='tight')    plt.close()                # Run all the visualization
functionsplot_country_comparison(africa_data)plot_time_series(africa_data)
plot_scatter_relationships(africa_data)plot_correlation_heatmap(africa_data)
plot_mediation_model(africa_data)plot_component_analysis(africa_data)
plot_regression_results(africa_data)plot_conclusion_summary(africa_data)
print("All visualizations have been created successfully!")import numpy as np
import pandas as pdimport matplotlib.pyplot as pltimport seaborn as snsfrom
matplotlib.patches import Patchfrom matplotlib.lines import Line2Dimport
matplotlib.colors as mcolorsfrom matplotlib.cm import ScalarMappable
# Set styles for better visualizationplt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("viridis")plt.rcParams['figure.figsize'] = (12, 8)
plt.rcParams['font.family'] = 'sans-serif'
# Function to generate synthetic datadef generate_africa_data(n_countries=29,
years=range(2013, 2023), seed=123):    """    Generate synthetic data for
African countries with realistic relationships    between e-governance,
institutional quality, and corruption.    """    np.random.seed(seed)
# List of 29 African countries in the study    countries = [
"Algeria", "Benin", "Botswana", "Burkina Faso", "Cameroon",         "Cote
d'Ivoire", "Egypt", "Ethiopia", "Ghana", "Guinea",         "Kenya",
"Madagascar", "Malawi", "Mali", "Mauritius",         "Morocco", "Mozambique",
"Namibia", "Niger", "Nigeria",         "Rwanda", "Senegal", "South Africa",
"Tanzania", "Togo",         "Tunisia", "Uganda", "Zambia", "Zimbabwe"    ]
    # Create empty lists to store data    data = []        # Generate base
characteristics for each country    country_characteristics = {}    for
country in countries:        # Base values with country-specific patterns
        if country in ["Botswana", "Mauritius", "Rwanda", "South Africa",
"Tunisia"]:            # Higher performing countries
country_characteristics[country] = {                'base_EGI': 0.45 +
np.random.beta(4, 2) * 0.3,  # E-Government Index (0.45-0.75)
'base_EPI': 0.40 + np.random.beta(4, 2) * 0.3,  # E-Participation Index
                'base_OSI': 0.45 + np.random.beta(4, 2) * 0.3,  # Online
Service Index
```

```
                            'base_HCI': 0.55 + np.random.beta(4, 2) * 0.3,
# Human Capital Index                'base_TII': 0.40 + np.random.beta(4, 2)
* 0.3,  # Telecom Infrastructure Index                'base_IQ':
np.random.normal(0.3, 0.3),         # Institutional Quality
'base_CPI': 45 + np.random.normal(0, 8),       # Corruption Perception Index
(higher is better)                'base_GDPPC': 3000 + np.random.lognormal(8,
0.7),  # GDP per capita                'base_UPG': np.random.normal(2,
0.5),          # Urban population growth                'base_TNRR':
np.random.beta(2, 5) * 15         # Natural resource rents           }
        elif country in ["Niger", "Guinea", "Mali", "Ethiopia", "Mozambique"]:
            # Lower performing countries
country_characteristics[country] = {                'base_EGI': 0.1 +
np.random.beta(2, 4) * 0.2,   # E-Government Index (0.1-0.3)
'base_EPI': 0.1 + np.random.beta(2, 4) * 0.2,   # E-Participation Index
                'base_OSI': 0.1 + np.random.beta(2, 4) * 0.2,   # Online
Service Index                'base_HCI': 0.25 + np.random.beta(2, 4) * 0.2,
# Human Capital Index                'base_TII': 0.1 + np.random.beta(2, 4) *
0.2,   # Telecom Infrastructure Index                'base_IQ':
np.random.normal(-0.8, 0.3),         # Institutional Quality
'base_CPI': 25 + np.random.normal(0, 5),        # Corruption Perception Index
                'base_GDPPC': 500 + np.random.lognormal(6, 0.5),  # GDP per
capita                'base_UPG': np.random.normal(3, 0.7),           # Urban
population growth                'base_TNRR': np.random.beta(5, 2) *
20          # Natural resource rents            }        else:           #
Medium
```

## dataanalysis\statistical_model.py

```
# Import required librariesimport numpy as npimport pandas as pdimport
matplotlib.pyplot as pltimport seaborn as snsfrom scipy import statsimport
statsmodels.api as smimport statsmodels.formula.api as smffrom
statsmodels.stats.outliers_influence import variance_inflation_factorfrom
statsmodels.tools.tools import add_constantfrom
statsmodels.sandbox.regression.gmm import GMMfrom linearmodels.panel import
PanelOLS, RandomEffects, PooledOLSimport warnings
warnings.filterwarnings('ignore')                                   # Set plot
styleplt.style.use('seaborn-v0_8-whitegrid')sns.set_palette("viridis")
plt.rcParams['figure.figsize'] = (12, 8)plt.rcParams['font.family'] = 'sans-
serif'      # Function to generate synthetic data that mimics the expected
relationshipsdef generate_synthetic_data(n_countries=29, years=range(2013,
2023), seed=42):    """    Generate synthetic data for the study based on
expected relationships    between e-governance, institutional quality, and
corruption.    """    np.random.seed(seed)        # List of African countries
in the study    countries = [        "Algeria", "Benin", "Botswana", "Burkina
Faso", "Cameroon",         "Cote d'Ivoire", "Egypt", "Ethiopia", "Ghana",
"Guinea",         "Kenya", "Madagascar", "Malawi", "Mali", "Mauritius",
        "Morocco", "Mozambique", "Namibia", "Niger", "Nigeria",
"Rwanda", "Senegal", "South Africa", "Tanzania", "Togo",         "Tunisia",
"Uganda", "Zambia", "Zimbabwe"    ]        # Limit to the specified number of
countries    countries = countries[:n_countries]        # Create empty lists
to store data    data = []        # Generate base characteristics for each
country    country_base = {}    for country in countries:        # Base
values with realistic distributions        country_base[country] = {
            'base_EGI': np.random.beta(2, 5) * 0.7 + 0.1,  # E-Government
Index (0.1-0.8)            'base_EPI': np.random.beta(2, 5) * 0.7 + 0.1,  # E-
Participation Index            'base_OSI': np.random.beta(2, 5) * 0.7 + 0.1,
# Online Service Index            'base_HCI': np.random.beta(3, 3) * 0.5 +
0.3,  # Human Capital Index (higher baseline)            'base_TII':
np.random.beta(2, 5) * 0.7 + 0.1,  # Telecom Infrastructure Index
'base_IQ': np.random.normal(-0.3, 0.5),        # Institutional Quality (-1.5
to 1.5)            'base_CPI': np.random.beta(2, 2) * 40 + 20,    #
Corruption Perception Index (20-60)            'base_GDPPC':
np.random.lognormal(7, 1),       # GDP per capita            'base_UPG':
np.random.normal(2, 1),           # Urban population growth
'base_TNRR': np.random.beta(2, 5) * 25         # Natural resource rents
        }               # Set up relationships between country
characteristics        # Countries with better historical institutions tend
to have better e-gov        if country_base[country]['base_IQ'] > 0:
            country_base[country]['base_EGI'] += 0.1
country_base[country]['base_OSI'] += 0.1                # Countries with
higher GDP tend to have better e-gov and less corruption        if
country_base[country]['base_GDPPC'] > 5000:            country_base[country]
['base_EGI'] += 0.15            country_base[country]['base_TII'] += 0.2
            country_base[country]['base_CPI'] += 10              #
Countries with high resource rents tend to have more corruption        if
country_base[country]['base_TNRR'] > 15:            country_base[country]
['base_CPI'] -= 12           country_base[country]['base_IQ'] -= 0.3
# Generate yearly data for each country    for country in countries:
for year in years:            year_idx = year - 2013  # For time-based
patterns            # Get base values for this country
base = country_base[country]                        # Create yearly values
with trends and noise            # E-governance improves over time with some
random variation            EGI = min(0.95, base['base_EGI'] + year_idx *
0.02 + np.random.normal(0, 0.03))            EPI = min(0.95, base['base_EPI']
+ year_idx * 0.025 + np.random.normal(0, 0.03))            OSI = min(0.95,
base['base_OSI'] + year_idx * 0.022 + np.random.normal(0, 0.03))
HCI = min(0.95, base['base_HCI'] + year_idx * 0.01 + np.random.normal(0,
```

```
0.02))            TII = min(0.95, base['base_TII'] + year_idx * 0.03 +
np.random.normal(0, 0.03))                        # Institutional quality
changes more slowly            IQ = base['base_IQ'] + year_idx * 0.015 +
np.random.normal(0, 0.05)             IQ = max(-2.5, min(2.5, IQ))  # Keep in
reasonable range                        # Calculate individual institutional
quality components            # Control of Corruption is closest to the CPI
            COC = IQ * 0.7 + np.random.normal(0, 0.2)            GE = IQ *
0.8 + np.random.normal(0, 0.15)            PSTAB = IQ * 0.5 +
np.random.normal(0, 0.3)            RQ = IQ * 0.75 + np.random.normal(0, 0.2)
            RL = IQ * 0.8 + np.random.normal(0, 0.15)            VA = IQ *
0.6 + np.random.normal(0, 0.25)                          # Bound institutional
quality measures between -2.5 and 2.5            for var in [COC, GE, PSTAB,
RQ, RL, VA]:                var = max(-2.5, min(2.5, var))
            # GDP increases over time with some random variation
GDPPC = base['base_GDPPC'] * (1 + 0.02) ** year_idx * (1 +
np.random.normal(0, 0.02))                        # Other controls
UPG = max(0, base['base_UPG'] - year_idx * 0.05 + np.random.normal(0, 0.3))
            TNRR = max(0, base['base_TNRR'] + np.random.normal(0, 2))
                        # CPI is influenced by e-governance, institutional
quality, and other factors             # Higher CPI means less corruption
            CPI = (                base['base_CPI'] +                EGI * 25
+               # E-governance direct effect                IQ * 15
+               # Institutional quality direct effect                year_idx
* 0.5 +       # Time trend                np.random.normal(0, 3)  # Random
noise             )                        # Keep CPI in range 0-100
            CPI = max(1, min(99, CPI))                          # Create a
composite ICT score            ICT = (TII * 0.6 + HCI * 0.3 + OSI * 0.1) +
np.random.normal(0, 0.05)            ICT = max(0.05, min(0.95, ICT))
                        # Add to dataset            data.append({
                'Country': country,                'Year': year,
                'CPI': CPI,                'EGI': EGI,                'EPI':
EPI,                'OSI': OSI,                'HCI': HCI,
'TII': TII,                'COC': COC,                'GE': GE,
                'PSTAB': PSTAB,                'RQ': RQ,                'RL':
RL,                'VA': VA,                'IQ': IQ,  # Composite
institutional quality                'GDPPC': GDPPC,                'UPG':
UPG,                'TNRR': TNRR,                'ICT': ICT            })
    # Convert to DataFrame    df = pd.DataFrame(data)        return df
# Generate synthetic datadf = generate_synthetic_data()
# Display basic information and summary statisticsprint(f"Dataset shape:
{df.shape}")print("\nSummary statistics:")print(df.describe().round(2))
# Explore relationships between key variablesprint("\nCorrelation between E-
Governance and Corruption:")print(f"Correlation(EGI, CPI) =
{df['EGI'].corr(df['CPI']):.4f}")
print("\nCorrelation between Institutional Quality and Corruption:")
print(f"Correlation(IQ, CPI) = {df['IQ'].corr(df['CPI']):.4f}")
print("\nCorrelation between E-Governance and Institutional Quality:")
print(f"Correlation(EGI, IQ) = {df['EGI'].corr(df['IQ']):.4f}")
# Create correlation matrix visualizationplt.figure(figsize=(12, 10))
corr_vars = ['CPI', 'EGI', 'EPI', 'OSI', 'HCI', 'TII', 'IQ', 'COC', 'GE',
'PSTAB', 'RQ', 'RL', 'VA', 'GDPPC', 'UPG', 'TNRR']corr_matrix =
df[corr_vars].corr()mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
sns.heatmap(corr_matrix, mask=mask, cmap='coolwarm', vmin=-1, vmax=1,
            annot=True, fmt='.2f', cbar_kws={'label': 'Correlation
Coefficient'})plt.title('Correlation Matrix of Key Variables', fontsize=16)
plt.tight_layout()plt.savefig('correlation_matrix.png', dpi=300,
bbox_inches='tight')                        # Create visualization to compare
countriesplt.figure(figsize=(14, 10))top_countries = df.groupby('Country')
['CPI'].mean().sort_values(ascending=False).head(10).indexcountry_data =
df[df['Country'].isin(top_countries)].groupby('Country')[['CPI', 'EGI',
'IQ']].mean().reset_index()
```

```
                                                    # Scale the values for
better visualizationcountry_data['EGI_scaled'] = country_data['EGI'] * 100
country_data['IQ_scaled'] = (country_data['IQ'] + 2.5) * 20  # Convert from
[-2.5,2.5] to [0,100]                       # Create a grouped bar chart
bar_width = 0.25x = np.arange(len(country_data))fig, ax =
plt.subplots(figsize=(14, 8))                                ax.bar(x -
bar_width, country_data['CPI'], bar_width, label='CPI', color='#4c78a8')
ax.bar(x, country_data['EGI_scaled'], bar_width, label='E-Government Index
(x100)', color='#f58518')ax.bar(x + bar_width, country_data['IQ_scaled'],
bar_width, label='Institutional Quality (scaled)', color='#72b7b2')
ax.set_xticks(x)ax.set_xticklabels(country_data['Country'], rotation=45,
ha='right')ax.set_ylabel('Score')ax.set_title('Comparison of Top 10 Countries
by CPI, E-Government, and Institutional Quality', fontsize=16)ax.legend()
plt.tight_layout()plt.savefig('country_comparison.png', dpi=300,
bbox_inches='tight')                    # Time series visualization
plt.figure(figsize=(14, 8))yearly_avg = df.groupby('Year')[['CPI', 'EGI',
'IQ']].mean().reset_index()                                  fig, ax1 =
plt.subplots(figsize=(14, 8))                                   color = 'tab:blue'
ax1.set_xlabel('Year')ax1.set_ylabel('CPI', color=color)
ax1.plot(yearly_avg['Year'], yearly_avg['CPI'], 'o-', color=color,
linewidth=3, label='CPI')ax1.tick_params(axis='y', labelcolor=color)
ax2 = ax1.twinx()color = 'tab:orange'ax2.set_ylabel('E-Government &
Institutional Quality', color=color)ax2.plot(yearly_avg['Year'],
yearly_avg['EGI'], 's-', color='tab:orange', linewidth=2, label='EGI')
ax2.plot(yearly_avg['Year'], (yearly_avg['IQ'] + 2.5) / 5, '^-',
color='tab:green', linewidth=2, label='IQ (scaled)')ax2.tick_params(axis='y',
labelcolor=color)                   lines1, labels1 =
ax1.get_legend_handles_labels()lines2, labels2 =
ax2.get_legend_handles_labels()ax2.legend(lines1 + lines2, labels1 + labels2,
loc='upper left')                   plt.title('Trends in CPI, E-Government
Index, and Institutional Quality (2013-2022)', fontsize=16)plt.tight_layout()
plt.savefig('time_series.png', dpi=300, bbox_inches='tight')
# Function to run regression models for mediation analysisdef
mediation_analysis(df):    """Perform mediation analysis using Baron & Kenny
approach"""    print("\n--- Mediation Analysis Results ---")        # Step 1:
Regress outcome (CPI) on predictor (EGI)    model_c = sm.OLS(df['CPI'],
sm.add_constant(df['EGI'])).fit()    print("\nStep 1: CPI ~ EGI (c path)")
print(f"Coefficient: {model_c.params[1]:.4f}, p-value:
{model_c.pvalues[1]:.4f}")        # Step 2: Regress mediator (IQ) on
predictor (EGI)    model_a = sm.OLS(df['IQ'],
sm.add_constant(df['EGI'])).fit()    print("\nStep 2: IQ ~ EGI (a path)")
print(f"Coefficient: {model_a.params[1]:.4f}, p-value:
{model_a.pvalues[1]:.4f}")        # Step 3: Regress outcome (CPI) on both
predictor (EGI) and mediator (IQ)    X = sm.add_constant(df[['EGI', 'IQ']])
    model_bc = sm.OLS(df['CPI'], X).fit()    print("\nStep 3: CPI ~ EGI + IQ
(b and c' paths)")    print(f"EGI Coefficient (c' path):
{model_bc.params[1]:.4f}, p-value: {model_bc.pvalues[1]:.4f}")    print(f"IQ
Coefficient (b path): {model_bc.params[2]:.4f}, p-value:
{model_bc.pvalues[2]:.4f}")        # Calculate direct, indirect, and total
effects    direct_effect = model_bc.params[1]  # c' path    indirect_effect =
model_a.params[1] * model_bc.params[2]  # a*b path    total_effect =
model_c.params[1]  # c path        print("\nMediation Effects:")
print(f"Direct Effect (c'): {direct_effect:.4f}")    print(f"Indirect Effect
(a*b): {indirect_effect:.4f}")    print(f"Total Effect (c):
{total_effect:.4f}")    print(f"Proportion Mediated: {indirect_effect/
total_effect:.4f} or {indirect_effect/total_effect*100:.2f}%")        # Sobel
test for significance of mediation    a = model_a.params[1]    b =
model_bc.params[2]    sea = model_a.bse[1]    seb = model_bc.bse[2]
sobel_se = np.sqrt(b**2 * sea**2 + a**2 * seb**2)    sobel_z =
indirect_effect / sobel_se
```

```
                            sobel_p = 2 * (1 - stats.norm.cdf(abs(sobel_z)))
        print("\nSobel Test:")    print(f"z-value: {sobel_z:.4f}")
print(f"p-value: {sobel_p:.4f}")        if sobel_p < 0.05:        print("The
mediation effect is statistically significant.")    else:        print("The
mediation effect is not statistically significant.")        # Return the
models and effects for further analysis    return {        'model_c': model_c,
        'model_a': model_a,        'model_bc': model_bc,
'direct_effect': direct_effect,        'indirect_effect': indirect_effect,
        'total_effect': total_effect,        'proportion_mediated':
indirect_effect/total_effect,        'sobel_z': sobel_z,        'sobel_p':
sobel_p    }     # Perform the mediation analysismediation_results =
mediation_analysis(df)                      # Visualize the mediation model
plt.figure(figsize=(10, 6))                             # Create nodes
plt.text(0.2, 0.6, 'E-Governance\n(EGI)', ha='center', va='center',
bbox=dict(boxstyle='round,pad=0.5', facecolor='lightblue', alpha=0.8),
fontsize=12)plt.text(0.8, 0.6, 'Corruption\n(CPI)', ha='center', va='center',
         bbox=dict(boxstyle='round,pad=0.5', facecolor='lightcoral',
alpha=0.8), fontsize=12)plt.text(0.5, 0.3, 'Institutional Quality\n(IQ)',
ha='center', va='center',          bbox=dict(boxstyle='round,pad=0.5',
facecolor='lightgreen', alpha=0.8), fontsize=12)
# Draw arrowsplt.arrow(0.28, 0.6, 0.44, 0, head_width=0.02, head_length=0.02,
fc='black', ec='black')plt.text(0.5, 0.64, f"c' =
{mediation_results['direct_effect']:.2f}", ha='center', fontsize=10)
plt.arrow(0.28, 0.57, 0.14, -0.2, head_width=0.02, head_length=0.02,
fc='black', ec='black')plt.text(0.34, 0.44, f"a =
{mediation_results['model_a'].params[1]:.2f}", ha='center', fontsize=10)
plt.arrow(0.58, 0.33, 0.14, 0.2, head_width=0.02, head_length=0.02,
fc='black', ec='black')plt.text(0.66, 0.44, f"b =
{mediation_results['model_bc'].params[2]:.2f}", ha='center', fontsize=10)
# Add mediation informationplt.text(0.5, 0.15, f"Indirect Effect (a×b) =
{mediation_results['indirect_effect']:.2f}", ha='center', fontsize=11)
plt.text(0.5, 0.1, f"Total Effect (c) =
{mediation_results['total_effect']:.2f}", ha='center', fontsize=11)
plt.text(0.5, 0.05, f"Proportion Mediated =
{mediation_results['proportion_mediated']*100:.1f}%", ha='center',
fontsize=11)            # Remove axesplt.axis('off')plt.title('Mediation
Model: E-Governance !’ Institutional Quality !’ Corruption', fontsize=14)
plt.savefig('mediation_model.png', dpi=300, bbox_inches='tight')
# Panel data analysis using GMM estimationdef panel_data_analysis(df):
"""Perform panel data analysis using GMM estimation"""    print("\n--- Panel
Data Analysis Results ---")        # Prepare the data for panel analysis
df_panel = df.copy()    df_panel['CountryYear'] = df_panel['Country'] +
df_panel['Year'].astype(str)    df_panel = df_panel.set_index(['Country',
'Year'])        # Model 1: Effect of E-Governance on Corruption    formula1 =
'CPI ~ EGI + GDPPC + UPG + TNRR'        # Model 2: Effect of E-Governance on
Institutional Quality    formula2 = 'IQ ~ EGI + GDPPC + UPG + TNRR'        #
Model 3: Effect of E-Governance and Institutional Quality on Corruption
formula3 = 'CPI ~ EGI + IQ + GDPPC + UPG + TNRR'        # Create dummy
variables for each country and year to simulate fixed effects
country_dummies = pd.get_dummies(df['Country'], drop_first=True,
prefix='country')    year_dummies = pd.get_dummies(df['Year'],
drop_first=True, prefix='year')        # Add dummies to dataframe
df_with_dummies = pd.concat([df.reset_index(drop=True), country_dummies,
year_dummies], axis=1)        # Simulate a simplified GMM approach with OLS
and controls    print("\nModel 1: Effect of E-Governance on Corruption")
X1 = sm.add_constant(df_with_dummies[['EGI', 'GDPPC', 'UPG', 'TNRR']])
model1 = sm.OLS(df_with_dummies['CPI'], X1).fit()    print(model1.summary())
        print("\nModel 2: Effect of E-Governance on Institutional Quality")
    X2 = sm.add_constant(df_with_dummies[['EGI', 'GDPPC', 'UPG', 'TNRR']])
model2 = sm.OLS(df_with_dummies['IQ'], X2).fit()
```

```
                                                print(model2.summary())
        print("\nModel 3: Effect of E-Governance and Institutional Quality on
Corruption")    X3 = sm.add_constant(df_with_dummies[['EGI', 'IQ', 'GDPPC',
'UPG', 'TNRR']])    model3 = sm.OLS(df_with_dummies['CPI'], X3).fit()
print(model3.summary())        # Create a summary table for comparison
results_summary = pd.DataFrame({        'Variable': ['E-Government Index
(EGI)', 'Institutional Quality (IQ)',                      'GDP per capita',
'Urban Population Growth', 'Natural Resources Rents',
'Constant'],        'Model 1 (CPI)': [f"{model1.params[1]:.4f}
***\n({model1.bse[1]:.4f})", "-",
f"{model1.params[2]:.4f}{'***' if model1.pvalues[2]<0.01 else '**' if
model1.pvalues[2]<0.05 else '*' if model1.pvalues[2]<0.1 else
''}\n({model1.bse[2]:.4f})",                        f"{model1.params[3]:.4f}
{'***' if model1.pvalues[3]<0.01 else '**' if model1.pvalues[3]<0.05 else '*'
if model1.pvalues[3]<0.1 else ''}\n({model1.bse[3]:.4f})",
                         f"{model1.params[4]:.4f}{'***' if
model1.pvalues[4]<0.01 else '**' if model1.pvalues[4]<0.05 else '*' if
model1.pvalues[4]<0.1 else ''}\n({model1.bse[4]:.4f})",
                         f"{model1.params[0]:.4f}{'***' if
model1.pvalues[0]<0.01 else '**' if model1.pvalues[0]<0.05 else '*' if
model1.pvalues[0]<0.1 else ''}\n({model1.bse[0]:.4f})"],        'Model 2
(IQ)': [f"{model2.params[1]:.4f}***\n({model2.bse[1]:.4f})", "-",
                        f"{model2.params[2]:.4f}{'***' if
model2.pvalues[2]<0.01 else '**' if model2.pvalues[2]<0.05 else '*' if
model2.pvalues[2]<0.1 else ''}\n({model2.bse[2]:.4f})",
                        f"{model2.params[3]:.4f}{'***' if
model2.pvalues[3]<0.01 else '**' if model2.pvalues[3]<0.05 else '*' if
model2.pvalues[3]<0.1 else ''}\n({model2.bse[3]:.4f})",
                        f"{model2.params[4]:.4f}{'***' if
model2.pvalues[4]<0.01 else '**' if model2.pvalues[4]<0.05 else '*' if
model2.pvalues[4]<0.1 else ''}\n({model2.bse[4]:.4f})",
                        f"{model2.params[0]:.4f}{'***' if
model2.pvalues[0]<0.01 else '**' if model2.pvalues[0]<0.05 else '*' if
model2.pvalues[0]<0.1 else ''}\n({model2.bse[0]:.4f})"],        'Model 3
(CPI)': [f"{model3.params[1]:.4f}{'***' if model3.pvalues[1]<0.01 else '**'
if model3.pvalues[1]<0.05 else '*' if model3.pvalues[1]<0.1 else
''}\n({model3.bse[1]:.4f})",                         f"{model3.params[2]:.4f}
***\n({model3.bse[2]:.4f})",                         f"{model3.params[3]:.4f}
{'***' if model3.pvalues[3]<0.01 else '**' if model3.pvalues[3]<0.05 else '*'
if model3.pvalues[3]<0.1 else ''}\n({model3.bse[3]:.4f})",
                         f"{model3.params[4]:.4f}{'***' if
model3.pvalues[4]<0.01 else '**' if model3.pvalues[4]<0.05 else '*' if
model3.pvalues[4]<0.1 else ''}\n({model3.bse[4]:.4f})",
                         f"{model3.params[5]:.4f}{'***' if
model3.pvalues[5]<0.01 else '**' if model3.pvalues[5]<0.05 else '*' if
model3.pvalues[5]<0.1 else ''}\n({model3.bse[5]:.4f})",
                         f"{model3.params[0]:.4f}{'***' if
model3.pvalues[0]<0.01 else '**' if model3.pvalues[0]<0.05 else '*' if
model3.pvalues[0]<0.1 else ''}\n({model3.bse[0]:.4f})"]    })
print("\nComparison of Results Across Models:")    print(results_summary)
    # Calculate the VIF to check for multicollinearity    X =
df_with_dummies[['EGI', 'IQ', 'GDPPC', 'UPG', 'TNRR']]    X =
sm.add_constant(X)    vif_data = pd.DataFrame()    vif_data["Variable"] =
X.columns    vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i
in range(X.shape[1])]        print("\nVariance Inflation Factors:")
print(vif_data)        # Return the models for further analysis    return
{'model1': model1, 'model2': model2, 'model3': model3}
# Perform panel data analysispanel_results = panel_data_analysis(df)
# Additional analysis of e-governance componentsdef
analyze_egov_components(df):
```

```
                                """Analyze the impact of different e-
governance components"""    print("\n--- E-Governance Components Analysis
---")        # Correlation of e-governance components with CPI
corr_with_cpi = [        ('EGI', df['EGI'].corr(df['CPI'])),        ('EPI',
df['EPI'].corr(df['CPI'])),        ('OSI', df['OSI'].corr(df['CPI'])),
('HCI', df['HCI'].corr(df['CPI'])),        ('TII', df['TII'].corr(df['CPI']))
    ]        # Sort by correlation strength    corr_with_cpi.sort(key=lambda
x: abs(x[1]), reverse=True)        print("\nCorrelation of E-Governance
Components with CPI:")    for component, corr in corr_with_cpi:
print(f"{component}: {corr:.4f}")        # Regression of CPI on all e-
governance components    X = sm.add_constant(df[['EPI', 'OSI', 'HCI', 'TII']])
    model = sm.OLS(df['CPI'], X).fit()        print("\nRegression of CPI on E-
Governance Components:")    print(model.summary())        # Create
visualization of correlations    components = [x[0] for x in corr_with_cpi]
    correlations = [x[1] for x in corr_with_cpi]
plt.figure(figsize=(10, 6))    bars = plt.bar(components, correlations,
color='skyblue')        # Add value labels    for bar in bars:        height
= bar.get_height()        plt.text(bar.get_x() + bar.get_width()/2., height,
                f'{height:.4f}',                ha='center', va='bottom',
fontsize=12)        plt.title('Correlation of E-Governance Components with
CPI', fontsize=14)    plt.ylabel('Correlation Coefficient')    plt.ylim(0,
max(correlations) * 1.1)    plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.savefig('egov_components_correlation.png', dpi=300, bbox_inches='tight')
        return {'correlations': corr_with_cpi, 'regression': model}
# Analyze e-governance componentsegov_components_analysis =
analyze_egov_components(df)                                # Additional analysis
of institutional quality componentsdef analyze_iq_components(df):
"""Analyze the impact of different institutional quality components"""
print("\n--- Institutional Quality Components Analysis ---")        #
Correlation of institutional quality components with CPI    corr_with_cpi = [
        ('COC', df['COC'].corr(df['CPI'])),        ('GE',
df['GE'].corr(df['CPI'])),        ('PSTAB', df['PSTAB'].corr(df['CPI'])),
        ('RQ', df['RQ'].corr(df['CPI'])),        ('RL',
df['RL'].corr(df['CPI'])),        ('VA', df['VA'].corr(df['CPI']))    ]
    # Sort by correlation strength    corr_with_cpi.sort(key=lambda x:
abs(x[1]), reverse=True)        print("\nCorrelation of Institutional Quality
Components with CPI:")    for component, corr in corr_with_cpi:
print(f"{component}: {corr:.4f}")        # Regression of CPI on all
institutional quality components    X = sm.add_constant(df[['COC', 'GE',
'PSTAB', 'RQ', 'RL', 'VA']])    model = sm.OLS(df['CPI'], X).fit()
print("\nRegression of CPI on Institutional Quality Components:")
print(model.summary())        # Create visualization of correlations
components = [x[0] for x in corr_with_cpi]    correlations = [x[1] for x in
corr_with_cpi]        plt.figure(figsize=(12, 6))    bars =
plt.bar(components, correlations, color='lightgreen')        # Add value
labels    for bar in bars:        height = bar.get_height()
plt.text(bar.get_x() + bar.get_width()/2., height,
f'{height:.4f}',                ha='center', va='bottom', fontsize=12)
plt.title('Correlation of Institutional Quality Components with CPI',
fontsize=14)    plt.ylabel('Correlation Coefficient')    plt.ylim(0,
max(correlations) * 1.1)    plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.savefig('iq_components_correlation.png', dpi=300, bbox_inches='tight')
    return {'correlations': corr_with_cpi, 'regression': model}
# Analyze institutional quality componentsiq_components_analysis =
analyze_iq_components(df)                                # Create regression models
for different country groupsdef compare_income_groups(df):    """Compare
results across different income groups"""    print("\n--- Comparison Across
Income Groups ---")        # Create income groups based on GDPPC
df['income_group'] = pd.qcut(df['GDPPC'], 3, labels=['Low', 'Middle', 'High'])
```

```
        results = {}        for group in ['Low', 'Middle', 'High']:
group_df = df[df['income_group'] == group]                # Step 1: Regress
outcome (CPI) on predictor (EGI)        model_c = sm.OLS(group_df['CPI'],
sm.add_constant(group_df['EGI'])).fit()                # Step 2: Regress
mediator (IQ) on predictor (EGI)        model_a = sm.OLS(group_df['IQ'],
sm.add_constant(group_df['EGI'])).fit()                # Step 3: Regress
outcome (CPI) on both predictor (EGI) and mediator (IQ)        X =
sm.add_constant(group_df[['EGI', 'IQ']])        model_bc =
sm.OLS(group_df['CPI'], X).fit()                # Calculate direct, indirect,
and total effects        direct_effect = model_bc.params[1]  # c' path
indirect_effect = model_a.params[1] * model_bc.params[2]  # a*b path
total_effect = model_c.params[1]  # c path                results[group] = {
            'direct_effect': direct_effect,            'indirect_effect':
indirect_effect,            'total_effect': total_effect,
'proportion_mediated': indirect_effect/total_effect        }        # Print
results    print("\nMediation Effects Across Income Groups:")    for group,
res in results.items():        print(f"\n{group} Income Group:")
print(f"Direct Effect (c'): {res['direct_effect']:.4f}")
print(f"Indirect Effect (a*b): {res['indirect_effect']:.4f}")
print(f"Total Effect (c): {res['total_effect']:.4f}")
print(f"Proportion Mediated: {res['proportion_mediated']:.4f} or
{res['proportion_mediated']*100:.2f}%")        # Create a bar chart to
compare effects across groups    groups = list(results.keys())
direct_effects = [results[g]['direct_effect'] for g in groups]
indirect_effects = [results[g]['indirect_effect'] for g in groups]
plt.figure(figsize=(10, 6))    width = 0.35    x = np.arange(len(groups))
    plt.bar(x - width/2, direct_effects, width, label='Direct Effect',
color='skyblue')    plt.bar(x + width/2, indirect_effects, width,
label='Indirect Effect', color='lightgreen')        plt.xlabel('Income Group')
    plt.ylabel('Effect Size')    plt.title('Direct and Indirect Effects of E-
Governance on Corruption Across Income Groups')    plt.xticks(x, groups)
plt.legend()    plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.savefig('income_group_comparison.png', dpi=300, bbox_inches='tight')
    return results                  # Compare income groups
income_group_comparison = compare_income_groups(df)
# Final conclusions and summary of findingsprint("\n=== SUMMARY OF FINDINGS
===")print("\n1. Mediation Analysis:")print(f"- Total Effect of E-Governance
on Corruption: {mediation_results['total_effect']:.4f}")print(f"- Direct
Effect: {mediation_results['direct_effect']:.4f}
({mediation_results['direct_effect']/
mediation_results['total_effect']*100:.2f}%)")print(f"- Indirect Effect
through Institutional Quality: {mediation_results['indirect_effect']:.4f}
({mediation_results['indirect_effect']/
mediation_results['total_effect']*100:.2f}%)")print(f"- Sobel Test: z =
{mediation_results['sobel_z']:.4f}, p = {mediation_results['sobel_p']:.4f}")
print("\n2. E-Governance Components:")print("- Most influential e-governance
components (correlation with CPI):")for component, corr in
egov_components_analysis['correlations'][:3]:    print(f"  * {component}:
{corr:.4f}")            print("\n3. Institutional Quality Components:")
print("- Most influential institutional quality components (correlation with
CPI):")for component, corr in iq_components_analysis['correlations'][:3]:
print(f"  * {component}: {corr:.4f}")
print("\n4. Income Group Differences:")print("- Proportion of effect mediated
by institutional quality:")for group, res in income_group_comparison.items():
    print(f"  * {group} Income Group: {res['proportion_mediated']*100:.2f}%")
print("\n5. Policy Implications:")print("- E-governance has both direct and
indirect effects on reducing corruption")print("- Institutional quality is a
significant mediator, especially in high-income countries")print("- Different
e-governance components have varying impacts - OSI and TII are most
effective")
```

```
            print("- Comprehensive approach addressing both e-governance and institutional quality is recommended")print("- Natural resource dependence remains a significant challenge for corruption reduction efforts")
```

## python\BCADN.py

```
import osimport sysimport jsonimport timeimport randomimport loggingfrom
typing import Dict, List, Any, Optional, Tuplefrom dotenv import load_dotenv
from web3 import Web3, HTTPProviderfrom eth_account import Accountimport
matplotlib.pyplot as pltimport pandas as pdimport numpy as np
# Load environment variablesload_dotenv()             # Configuration
RESULT_OUTPUT_DIR = "results/bcadn_analysis"os.makedirs(RESULT_OUTPUT_DIR,
exist_ok=True)                              class BCADNNetworkAnalyzer:    def
__init__(        self,        web3: Web3,        contract_addresses_path:
Optional[str] = None,        build_contracts_dir: Optional[str] = None,    ):
        """        Initialize BCADN Network Analyzer
        :param web3: Web3 instance        :param contract_addresses_path:
Path to contract addresses JSON        :param build_contracts_dir: Directory
containing contract build artifacts        """        # Setup logging
logging.basicConfig(            level=logging.INFO, format="%(asctime)s -
%(levelname)s - %(message)s"        )        self.logger =
logging.getLogger(__name__)                                    # Determine
project root and default paths        self.project_root = os.path.abspath(
            os.path.join(os.path.dirname(__file__), "..")        )
        # Contract addresses file        self.contract_addresses_path =
contract_addresses_path or os.path.join(            self.project_root,
"config", "contract_addresses.json"        )                # Build
contracts directory        self.build_contracts_dir = build_contracts_dir or
os.path.join(            self.project_root, "build", "contracts"        )
        # Blockchain connection        self.w3 = web3
        # Load contract addresses and configurations
self.contract_addresses = self._load_contract_addresses()
        # BCADN-specific contracts to load        self.bcadn_contract_names =
["BCADN", "ProactiveDefenseMechanism"]
        # Contracts storage        self.contracts: Dict[str, Any] = {}
    def _load_contract_addresses(self) -> Dict[str, str]:        """
Load contract addresses from JSON file
        :return: Dictionary of contract addresses        """        try:
            with open(self.contract_addresses_path, "r") as f:
raw_addresses = json.load(f)
self.logger.info(                f"Loaded contract addresses from
{self.contract_addresses_path}"            )            return raw_addresses
        except FileNotFoundError:            self.logger.error(
                f"Contract addresses file not found at
{self.contract_addresses_path}"            )            return {}
except json.JSONDecodeError:            self.logger.error(
f"Invalid JSON in contract addresses file at {self.contract_addresses_path}"
            )            return {}                            def
_load_contract_abi(self, contract_name: str) -> Optional[List[Dict[str,
Any]]]:        """        Load ABI for a given contract name.
        :param contract_name: Name of the contract to load ABI for
        :return: ABI as a list of dictionaries, or None if not found
"""        # Define possible ABI paths        abi_paths = [
os.path.join(self.build_contracts_dir, f"{contract_name}.json"),            #
Add other possible ABI paths here        ]                for abi_path in
abi_paths:            try:                if os.path.exists(abi_path):
                with open(abi_path, "r") as f:                    #
Read the entire file content                    content = f.read()
                    # Try parsing as JSON                    try:
                        contract_data = json.loads(content)
                        # Multiple possible ABI locations in JSON
                        possible_abi_keys = [
                            "abi",  # Truffle/Hardhat standard
                            "contractName",  # Alternative key
                            "compilerOutput",  # Another possible location
```

```
                            "output",  # Yet another possible key
                        ]
                        # Try each possible key
                        for key in possible_abi_keys:
                            if (
isinstance(contract_data, dict)                                     and key in
contract_data                                    ):
                                abi = contract_data[key]
                                # Ensure it's a list of ABI entries
                                if isinstance(abi, list):
                                    print(
                                        f"Successfully loaded ABI for
{contract_name} from {abi_path}"                                         )
                                    # Optional: Print out available
function names                                                 function_names = [
                                        func.get("name", "unnamed")
                                        for func in abi
                                        if func.get("type") == "function"
                                    ]
                                    print(
                                        f"Available functions in
{contract_name}: {function_names}"                                          )
                                    return abi
                        # If no key worked, check if entire content is ABI
                        if isinstance(contract_data, list):
                            return contract_data
                    except json.JSONDecodeError:
                        # Fallback: try parsing raw content
                        try:                                    raw_abi =
json.loads(content)                                 if isinstance(raw_abi,
list):                                   return raw_abi
                        except:
self.logger.warning(                                        f"Could not parse
JSON from {abi_path}"                                    )
            except Exception as e:                self.logger.warning(f"Error
reading ABI at {abi_path}: {e}")
self.logger.error(f"No ABI found for contract: {contract_name}")
return None               def validate_ethereum_address(self, address: str) -
> str:        """        Validate an Ethereum address and return checksum
version if valid                        :param address: Ethereum address to
validate        :return: Checksum address if valid, raises ValueError if
invalid        """        try:            if not address or address == "0x0":
                raise ValueError("Empty or zero address provided")
            checksum_address = Web3.to_checksum_address(address)
if not Web3.is_address(checksum_address):                raise
ValueError(f"Invalid Ethereum address format: {address}")
            return checksum_address        except Exception as e:
raise ValueError(f"Address validation error: {str(e)}")
    def verify_contract_state(self, contract_address: str) -> bool:        """
        Verify contract is properly deployed and accessible
        :param contract_address: Contract address to verify        :return:
Boolean indicating if contract is properly deployed        """        try:
            code = self.w3.eth.get_code(contract_address)            return
len(code) > 0        except Exception as e:
self.logger.error(f"Contract state verification failed: {e}")
return False               def load_bcadn_contracts(self) -> Dict[str, Any]:
        """        Load BCADN-related contracts with comprehensive
verification                    :return: Dictionary of loaded contracts
        """
```

```
                self.logger.info("Loading BCADN-related contracts")
        # Verify web3 connection first        if not self.w3.is_connected():
            raise ConnectionError("No Web3 connection available")
        # Verify default account is set        if not
self.w3.eth.default_account:            self.logger.warning(
"No default account set. Some functions may not work properly."            )
        for contract_name in self.bcadn_contract_names:            try:
                # Get and validate contract address
raw_address = self.contract_addresses.get(contract_name)
                try:                    contract_address =
self.validate_ethereum_address(raw_address)                except ValueError
as ve:                    self.logger.error(                        f"Address
validation failed for {contract_name}: {ve}"                    )
                    continue                                        #
Verify contract deployment                if not
self.verify_contract_state(contract_address):
self.logger.error(                        f"Contract {contract_name} not
properly deployed at {contract_address}"                    )
                    continue                                        #
Load ABI                    abi = self._load_contract_abi(contract_name)
                if not abi:                    self.logger.warning(f"Could
not load ABI for {contract_name}")                    continue
                # Create contract instance with verified checksum address
                contract = self.w3.eth.contract(address=contract_address,
abi=abi)                        # Comprehensive Contract Verification
                print(f"\n--- {contract_name} Contract Verification ---")
                print(f"Contract Address: {contract_address}")
print(f"Network: {self.w3.eth.chain_id}")
                # Check contract bytecode with improved error handling
                try:                    contract_bytecode =
self.w3.eth.get_code(contract_address)                    bytecode_length =
len(contract_bytecode)                    print(f"Contract Bytecode Length:
{bytecode_length}")                    if bytecode_length
== 0:                        raise ValueError("No bytecode found at contract
address")                    elif (                        bytecode_length <
100                    ):  # Arbitrary minimum size for a valid contract
                        self.logger.warning(
f"Unusually small bytecode size ({bytecode_length} bytes) for {contract_name}"
                        )                except Exception as bytecode_error:
                    self.logger.error(f"Bytecode verification failed:
{bytecode_error}")                    continue
                # Try to get contract owner with timeout                try:
                    owner = contract.functions.owner().call(
                        {"from": self.w3.eth.default_account}
                    )                    print(f"Contract Owner: {owner}")
                    # Verify owner is valid address                    if not
Web3.is_address(owner):                        self.logger.warning(f"Invalid
owner address returned: {owner}")                except Exception as
owner_error:                    self.logger.warning(f"Could not retrieve
contract owner: {owner_error}")
# Function verification with categorization                function_names = [
                    func.get("name", "unnamed")                    for func
in abi                    if func.get("type") == "function"                ]
                # Categorize functions                view_functions = []
                write_functions = []                special_functions = [
                    "calculateDynamicFee",
"adjustToProbabilityGap",                    "isWithinGap",                ]
                print("\nContract Functions Analysis:")                for
func_name in function_names:                    try:
if not hasattr(contract.functions, func_name):
```

```
continue                                    func = getattr(contract.functions,
func_name)                                      # Attempt to identify function
type                            try:                                    func_object =
contract.get_function_by_name(func_name)                                        if
func_object.get("stateMutability") in ["view", "pure"]:
                                view_functions.append(func_name)
                            else:
write_functions.append(func_name)                           except:
                            write_functions.append(func_name)
                        print(f"\n- {func_name}")
                        # Test special functions                                if
func_name in special_functions:                             try:
                                result = func(10).call(
                                    {"from": self.w3.eth.default_account}
                                )                                       print(f"  '
Special function test successful")                                       print(f"
Result: {result}")                                  except Exception as
special_error:                                      print(f"  '  Special function
test failed: {special_error}")
                        # Test view functions                                   elif
func_name in ["getAllNodes", "getAttackHistory", "owner"]:
                            try:                                        result =
func().call({"from": self.w3.eth.default_account})
                                print(f"  '  View function test successful")
                                print(f"  Result: {result}")
                            except Exception as view_error:
                                print(f"  '  View function test failed:
{view_error}")                                      except Exception as
func_error:                             self.logger.error(
                            f"Error analyzing function {func_name}:
{func_error}"                                )
                # Print function statistics                 print(f"\nFunction
Statistics:")                 print(f"Total Functions: {len(function_names)}")
                print(f"View/Pure Functions: {len(view_functions)}")
                print(f"State-Changing Functions: {len(write_functions)}")
                # Store contract with metadata
self.contracts[contract_name] = {                       "contract": contract,
                    "address": contract_address,
"bytecode_size": bytecode_length,                       "view_functions":
view_functions,                         "write_functions": write_functions,
                }                                       self.logger.info(
                    f"Successfully loaded and verified {contract_name} at
{contract_address}"                                 )                                   except
Exception as e:                 self.logger.error(f"Error loading contract
{contract_name}: {e}")                      continue
# Final verification        if not self.contracts:              raise
ValueError("No contracts were successfully loaded")
        return self.contracts                                       def
load_sample_data(self):         """Load sample data into BCADN contracts from
CSV files"""        # Determine paths         script_dir =
os.path.dirname(os.path.abspath(__file__))          project_root =
os.path.abspath(os.path.join(script_dir, '..'))                 # Define
expected CSV files in project root          nodes_csv =
os.path.join(project_root, 'sample_nodes.csv')          attacks_csv =
os.path.join(project_root, 'sample_attacks.csv')                # Validate
CSV files exist         if not os.path.exists(nodes_csv):               raise
FileNotFoundError(f"Nodes CSV file not found at: {nodes_csv}")          if not
os.path.exists(attacks_csv):              raise FileNotFoundError(f"Attacks CSV
file not found at: {attacks_csv}")
```

```
                                                                                #
Load data from CSV files        try:            # Load nodes data
nodes_df = pd.read_csv(nodes_csv)           required_node_columns =
{'address', 'performance', 'reliability'}            if not
required_node_columns.issubset(nodes_df.columns):                missing =
required_node_columns - set(nodes_df.columns)               raise
ValueError(f"Missing required columns in nodes CSV: {missing}")
            # Convert to list of dicts            sample_nodes =
nodes_df.to_dict('records')                         # Load attacks data
            attacks_df = pd.read_csv(attacks_csv)
required_attack_columns = {'node', 'anomaly_score', 'attack_type'}
if not required_attack_columns.issubset(attacks_df.columns):
missing = required_attack_columns - set(attacks_df.columns)
raise ValueError(f"Missing required columns in attacks CSV: {missing}")
                        sample_attacks = attacks_df.to_dict('records')
                    except Exception as e:
self.logger.error(f"Error loading CSV data: {e}")            raise
        # Create Web3 connection        web3 = create_web3_connection()
        if not web3:            raise ConnectionError("Failed to establish
Web3 connection")                        # Load contracts        analyzer =
BCADNNetworkAnalyzer(            web3=web3,
contract_addresses_path=os.path.join(project_root, 'config',
'contract_addresses.json'),
build_contracts_dir=os.path.join(project_root, 'build', 'contracts')        )
                loaded_contracts = analyzer.load_bcadn_contracts()        if
not loaded_contracts:            raise ValueError("Failed to load contracts")
        bcadn_contract = loaded_contracts.get("BCADN", {}).get("contract")
        if not bcadn_contract:            raise ValueError("BCADN contract
not loaded")                    # Validate and process nodes
valid_nodes = []        for node in sample_nodes:            try:
                # Validate address format                validated_address =
analyzer.validate_ethereum_address(node['address'])
valid_nodes.append({                    'address': validated_address,
                    'performance': int(node['performance']),
                    'reliability': int(node['reliability'])                })
            except (ValueError, KeyError) as e:
self.logger.warning(f"Skipping invalid node data: {node}. Error: {e}")
        # Validate and process attacks        valid_attacks = []        for
attack in sample_attacks:            try:                validated_node =
analyzer.validate_ethereum_address(attack['node'])
valid_attacks.append({                    'node': validated_node,
                    'anomaly_score': int(attack['anomaly_score']),
                    'attack_type': str(attack['attack_type'])
                })            except (ValueError, KeyError) as e:
                self.logger.warning(f"Skipping invalid attack data: {attack}.
Error: {e}")                    # Register nodes
self.logger.info(f"Registering {len(valid_nodes)} nodes...")        for node
in valid_nodes:            try:                self.logger.info(f"Registering
node: {node['address']}")                tx_hash =
bcadn_contract.functions.registerNode(                    node['address'],
                    node['performance'],
node['reliability']                ).transact({'from':
web3.eth.default_account})                                    # Wait for
transaction receipt                receipt =
web3.eth.wait_for_transaction_receipt(tx_hash)                if
receipt.status == 1:                    self.logger.info(f"Node
{node['address']} registered successfully")                else:
                    self.logger.error(f"Failed to register node
{node['address']}")            except Exception as e:
self.logger.error(f"Error registering node {node['address']}: {e}")
```

```
        # Record attacks        self.logger.info(f"Recording
{len(valid_attacks)} attacks...")        for attack in valid_attacks:
            try:                self.logger.info(f"Recording attack for node:
{attack['node']}")                tx_hash =
bcadn_contract.functions.recordAnomaly(                    attack['node'],
                    attack['anomaly_score'],
attack['attack_type']                ).transact({'from':
web3.eth.default_account})                                    # Wait for
transaction receipt                receipt =
web3.eth.wait_for_transaction_receipt(tx_hash)                if
receipt.status == 1:                    self.logger.info(f"Attack recorded
successfully for {attack['node']}")                else:
self.logger.error(f"Failed to record attack for {attack['node']}")
except Exception as e:                self.logger.error(f"Error recording
attack for {attack['node']}: {e}")
self.logger.info("Sample data loading complete!")        return {
'nodes_registered': len(valid_nodes),            'attacks_recorded':
len(valid_attacks),            'invalid_nodes': len(sample_nodes) -
len(valid_nodes),            'invalid_attacks': len(sample_attacks) -
len(valid_attacks)        }                    def
analyze_network_performance(self) -> Dict[str, Any]:        """
Comprehensive analysis of BCADN network performance
        :return: Dictionary of network performance metrics        """
# Initialize results structure        results = {
"network_metrics": {},            "node_performance": {
"node_addresses": [],                "weights": [],
"statuses": [],                "performance_details": [],            },
            "attack_history": {                "node_addresses": [],
                "timestamps": [],                "anomaly_scores": [],
                "attack_types": [],                "resolved": [],
            },        }                # Advanced debugging for contract
interaction        try:            bcadn = self.contracts.get("BCADN",
{}).get("contract")            proactive_defense =
self.contracts.get("ProactiveDefenseMechanism", {}).get("contract")
            if not bcadn or not proactive_defense:                raise
ValueError("BCADN or ProactiveDefenseMechanism contract not loaded")
            # Detailed contract inspection            print("\n--- Contract
Deployment Details ---")            bcadn_address = self.contracts["BCADN"]
["address"]            proactive_defense_address =
self.contracts["ProactiveDefenseMechanism"]["address"]
print(f"BCADN Contract Address: {bcadn_address}")
print(f"ProactiveDefenseMechanism Contract Address:
{proactive_defense_address}")            print(f"Network Chain ID:
{self.w3.eth.chain_id}")                                    # Attempt to
retrieve network information            try:                # Try to get node
information from BCADN contract                nodes_data =
bcadn.functions.getAllNodes().call()
results["node_performance"]["node_addresses"] = nodes_data[0]
results["node_performance"]["weights"] = nodes_data[1] or []
                # Map integer status to string status
status_map = ["Active", "Probation", "Excluded", "Pending"]
results["node_performance"]["statuses"] = [
status_map[min(status, len(status_map) - 1)]                    for status
in (nodes_data[2] or [])                ]
print(f"Retrieved {len(nodes_data[0])} nodes from BCADN contract")
except Exception as node_error:                print(f"Error retrieving nodes
from BCADN contract: {node_error}")
            # Try to get attack history            try:
attack_history = bcadn.functions.getAttackHistory().call()
```

```
results["attack_history"]["node_addresses"] = attack_history[0] or []
                results["attack_history"]["timestamps"] = attack_history[1]
or []                results["attack_history"]["anomaly_scores"] =
attack_history[2] or []                results["attack_history"]["resolved"]
= attack_history[3] or []                  results["attack_history"]
["attack_types"] = [                  "Unknown" for _ in
range(len(attack_history[0]) if attack_history[0] else 0)                ]
                print(f"Retrieved {len(attack_history[0])} attack history
entries")            except Exception as attack_error:
print(f"Error retrieving attack history: {attack_error}")
            # Additional network metrics
results["network_metrics"] = {                "total_nodes":
len(results["node_performance"]["node_addresses"]),
"total_attacks": len(results["attack_history"]["node_addresses"]),
                "resolved_attacks": sum(results["attack_history"]["resolved"])
                if results["attack_history"]["resolved"]                else
0,                  "chain_id": self.w3.eth.chain_id,
"latest_block": self.w3.eth.block_number,                "gas_price":
self.w3.eth.gas_price,            }                    except Exception as e:
            print(f"Critical Error in Network Performance Analysis: {e}")
            import traceback            traceback.print_exc()
        return results                          def
simulate_network_stress_test(self, num_transactions: int = 50) -> Dict[str,
Any]:        """        Simulate network stress test by submitting multiple
transactions                    :param num_transactions: Number of
transactions to simulate        :return: Stress test results dictionary
containing transaction details and metrics        """        # Get BCADN
contract instance        bcadn = self.contracts.get("BCADN",
{}).get("contract")        if not bcadn:            raise ValueError("BCADN
contract not loaded")                                # Initialize results
structure        stress_test_results = {            "transactions": [],
            "total_processing_time": 0,            "average_dynamic_fee": 0,
            "max_congestion_index": 0,            "successful_transactions":
0,            "failed_transactions": 0,            "average_gas_used": 0,
            "total_gas_cost": 0,        }                # Submit
transactions        for i in range(num_transactions):            try:
                # Base fee and amount calculation                base_fee =
random.randint(1, 10)                amount = random.randint(1, 100)
                # Calculate dynamic fee                try:
                    dynamic_fee =
bcadn.functions.calculateDynamicFee(base_fee).call(
{"from": self.w3.eth.default_account}                )
except Exception as fee_error:                    self.logger.error(f"Error
calculating dynamic fee: {fee_error}")                    dynamic_fee =
base_fee  # Fallback to base fee
                # Record transaction result                tx_result = {
                    "base_fee": base_fee,                  "dynamic_fee":
dynamic_fee,                    "amount": amount,
"timestamp": int(time.time()),                }
                stress_test_results["transactions"].append(tx_result)
                stress_test_results["average_dynamic_fee"] += dynamic_fee
                # Simulate congestion index                current_congestion
= random.uniform(0, 100)
stress_test_results["max_congestion_index"] = max(
stress_test_results["max_congestion_index"], current_congestion
                )
stress_test_results["successful_transactions"] += 1
                # Small delay to simulate network conditions
time.sleep(0.1)
```

```
                                    except Exception as e:
                self.logger.error(f"Error in transaction simulation {i+1}:
{e}")                 stress_test_results["failed_transactions"] += 1
        # Calculate averages        total_tx =
len(stress_test_results["transactions"])        if total_tx > 0:
stress_test_results["average_dynamic_fee"] /= total_tx
        # Add summary metrics        stress_test_results["summary"] = {
            "total_transactions": total_tx,            "success_rate": (
                stress_test_results["successful_transactions"] / total_tx
                if total_tx > 0                else 0            ),
            "average_dynamic_fee": stress_test_results["average_dynamic_fee"],
            "max_congestion_index":
stress_test_results["max_congestion_index"],        }
self.logger.info("Network stress test completed")        self.logger.info(
            f"Success rate: {stress_test_results['summary']
['success_rate']:.2%}"        )                return stress_test_results
    def visualize_network_analysis(self, network_results: Dict[str, Any]) ->
None:        """        Visualize network analysis results
        :param network_results: Results from network performance analysis
        """        try:            import matplotlib.pyplot as plt
import os                    # Ensure results directory exists
os.makedirs(RESULT_OUTPUT_DIR, exist_ok=True)
            # Visualize Node Performance            plt.figure(figsize=(12,
6))              # Node Weights Distribution            plt.subplot(1, 2, 1)
            node_weights = network_results.get("node_performance",
{}).get("weights", [])            if node_weights and any(node_weights):
                plt.hist(node_weights, bins=min(10, len(node_weights)),
edgecolor="black")                plt.title("Node Weights Distribution")
                plt.xlabel("Node Weight")
plt.ylabel("Frequency")            else:                plt.text(0.5, 0.5,
"No Node Data Available",
horizontalalignment='center',
verticalalignment='center')                plt.title("Node Weights
Distribution")                # Attack History
plt.subplot(1, 2, 2)            attack_scores =
network_results.get("attack_history", {}).get("anomaly_scores", [])
            resolved_attacks = network_results.get("attack_history",
{}).get("resolved", [])                                    # Check if there
are any attacks            if attack_scores and resolved_attacks:
                # Color resolved and unresolved attacks differently
                resolved_scores = [                    score for i, score in
enumerate(attack_scores) if resolved_attacks[i]                ]
                unresolved_scores = [                    score for i, score
in enumerate(attack_scores) if not resolved_attacks[i]                ]
                plt.bar(                    ["Resolved", "Unresolved"],
                    [len(resolved_scores), len(unresolved_scores)],
                    color=["green", "red"],                )
plt.title("Attack Resolution Status")                plt.ylabel("Number of
Attacks")            else:                plt.text(0.5, 0.5, "No Attack Data
Available",                         horizontalalignment='center',
                        verticalalignment='center')
plt.title("Attack Resolution Status")
            plt.tight_layout()
plt.savefig(os.path.join(RESULT_OUTPUT_DIR, "network_analysis.png"))
            plt.close()                                    # Save network
results as JSON            with open(os.path.join(RESULT_OUTPUT_DIR,
"network_results.json"), "w") as f:                json.dump(network_results,
f, indent=2)                        print(f"Network analysis visualization
saved in {RESULT_OUTPUT_DIR}")                                        except
ImportError:
```

```
                        print("Matplotlib not available. Skipping
visualization.")        except Exception as e:            print(f"Error in
visualization: {e}")            import traceback
traceback.print_exc()                                           def
create_web3_connection():    """    Create a Web3 connection to Sepolia
testnet using Infura                        :return: Configured Web3 instance
    """    try:        # Retrieve Infura Project ID and Private Key from
environment variables        INFURA_PROJECT_ID =
os.getenv("INFURA_PROJECT_ID")        PRIVATE_KEY = os.getenv("PRIVATE_KEY")
        if not INFURA_PROJECT_ID:            raise
ValueError("INFURA_PROJECT_ID not found in environment variables")
        # Construct Infura URL for Sepolia        infura_url = f"https://
sepolia.infura.io/v3/{INFURA_PROJECT_ID}"
        # Create Web3 instance with HTTPProvider        web3 =
Web3(HTTPProvider(infura_url))                                  # Add PoA
middleware for Sepolia        try:            from web3.middleware import
geth_poa_middleware
web3.middleware_onion.inject(geth_poa_middleware, layer=0)        except
ImportError:            print("Warning: Could not import geth_poa_middleware")
        # Detailed connection verification        print("\n--- Web3
Connection Diagnostics ---")        print(f"Connecting to: {infura_url}")
        # Check connection status        is_connected = web3.is_connected()
        print(f"Connection Status: {'Connected' if is_connected else 'Not
Connected'}")                    if not is_connected:            raise
ConnectionError("Failed to connect to Infura Sepolia endpoint")
        # Retrieve and print network information        try:
print(f"Chain ID: {web3.eth.chain_id}")            print(f"Latest Block
Number: {web3.eth.block_number}")            print(f"Gas Price:
{web3.eth.gas_price} Wei")        except Exception as network_error:
            print(f"Error retrieving network information: {network_error}")
        # Set up account if private key is provided        if PRIVATE_KEY:
            try:                account = Account.from_key(PRIVATE_KEY)
                checksummed_address =
Web3.to_checksum_address(account.address)
web3.eth.default_account = checksummed_address
print(f"\nAccount Details:")                print(f"Account Address:
{checksummed_address}")                print(                    f"Account
Balance: {web3.eth.get_balance(checksummed_address)} Wei"                )
            except ValueError as ve:                print(f"Invalid address
format: {ve}")            except Exception as account_error:
print(f"Error setting up account: {account_error}")
        return web3    except Exception as e:
logging.error(f"Comprehensive connection error: {e}")        return None
def main():    """    Main function to run BCADN Network Analysis    """
try:        # Create Web3 connection        web3 = create_web3_connection()
        if web3 is None:            print("Could not establish blockchain
connection. Exiting.")            sys.exit(1)                                 #
Determine the absolute path to contract addresses        script_dir =
os.path.dirname(os.path.abspath(__file__))        project_root =
os.path.abspath(os.path.join(script_dir, ".."))
contract_addresses_path = os.path.join(            project_root, "config",
"contract_addresses.json"        )        build_contracts_dir =
os.path.join(project_root, "build", "contracts")
        # Initialize network analyzer        analyzer = BCADNNetworkAnalyzer(
            web3=web3,
contract_addresses_path=contract_addresses_path,
build_contracts_dir=build_contracts_dir,        )                # Load
BCADN contracts        loaded_contracts = analyzer.load_bcadn_contracts()
        print("Loaded Contracts:", list(loaded_contracts.keys()))
        # Load sample data
```

```
                                    analyzer.load_sample_data()
# Analyze network performance        network_results =
analyzer.analyze_network_performance()
        # Run stress test simulation        stress_test_results =
analyzer.simulate_network_stress_test()
        # Combine results        network_results["stress_test"] =
stress_test_results                             # Visualize network analysis
        analyzer.visualize_network_analysis(network_results)
    except Exception as e:        logging.error(f"An error occurred in main:
{e}")        import traceback        traceback.print_exc()  # Print full
stack trace        sys.exit(1)                                          if
__name__ == "__main__":    main()
```

## python\blockchainworkflow.py

```
import osÐimport sysÐimport jsonÐimport loggingÐimport asyncioÐimport pandas as pdÐ
from datetime import datetimeÐfrom pathlib import PathÐfrom typing import Dict,
Optional, UnionÐfrom web3 import Web3Ðfrom web3.exceptions import
ContractLogicErrorÐclass BlockchainWorkflow:Ð   def __init__(Ð       self, Ð
        provider_url='http://127.0.0.1:7545',Ð       project_root=None,Ð
        addresses_config=None,Ð       contracts_dir=NoneÐ   ):Ð       """Ð
        Initialize BlockchainWorkflow with comprehensive contract loading and
orchestrationÐ       """Ð       # Set project rootÐ       self.project_root =
project_root or os.path.abspath(Ð
os.path.join(os.path.dirname(__file__), '..')Ð       )Ð       Ð       #
Define contract directoriesÐ       self.contracts_dir = contracts_dir or
os.path.join(self.project_root, 'build', 'contracts')Ð       Ð       # Setup
logging firstÐ       self.logger = self._setup_logger()Ð       Ð       # Load
contract addressesÐ       self.CONTRACT_ADDRESSES =
self._load_contract_addresses(addresses_config)Ð       Ð       # Initialize
Web3 connectionÐ       self.w3 = Web3(Web3.HTTPProvider(provider_url))Ð       Ð
        # Verify connectionÐ       if not self.w3.is_connected():Ð
raise ConnectionError("Failed to connect to Ethereum provider")Ð       Ð
        # Initialize contractsÐ       self.contracts = {}Ð
self._load_contracts()Ð       Ð       # Initialize modules after everything
else is set upÐ       self._init_climate_modules()Ð   def
_init_climate_modules(self):Ð       """Ð       Initialize climate-related
modulesÐ       """Ð       # Import modules here to avoid circular importsÐ
        from climatemodule import CityModuleÐ       from companymodule import
CompanyModuleÐ       from emmissionsmodule import EmissionsModuleÐ       from
healthmodule import HealthModuleÐ       from renewalmodule import
RenewalModuleÐ       try:Ð           self.city_module =
CityModule(workflow=self)Ð           self.company_module =
CompanyModule(workflow=self)Ð              self.emissions_module =
EmissionsModule(workflow=self)Ð             self.health_module =
HealthModule(workflow=self)Ð           self.renewal_module =
RenewalModule(workflow=self)Ð       except Exception as e:Ð
self.logger.error(f"Error initializing modules: {e}")Ð           raiseÐ   def
_load_sample_data(self):Ð       """Ð       Load sample data for testingÐ
        Ð       :return: Tuple of city and company DataFramesÐ       """Ð
        # Sample city dataÐ       city_data = pd.DataFrame({Ð
'city': ['CityA', 'CityA', 'CityB', 'CityB'],Ð           'date':
pd.to_datetime(['2023-01-01', '2023-02-01', '2023-01-01', '2023-02-01']),Ð
            'sector': ['Energy', 'Transport', 'Energy', 'Transport'],Ð
            'value': [10.5, 12.3, 8.7, 9.2]Ð       })Ð       Ð       # Sample
company dataÐ       company_data = pd.DataFrame({Ð           'company_name':
['CompanyX', 'CompanyY'],Ð           'registration_date':
pd.to_datetime(['2023-01-01', '2023-02-01']),Ð           'sector': ['Energy',
'Transport'],Ð           'emissions_baseline': [15.0, 12.5]Ð       })Ð       Ð
        return city_data, company_dataÐ   async def run_full_workflow(self):Ð
        """Ð       Execute full climate workflowÐ       Ð       :return:
Dictionary of workflow resultsÐ       """Ð       try:Ð           # Load
sample dataÐ           city_data, company_data = self._load_sample_data()Ð
            Ð           self.logger.info("Starting Climate Workflow")Ð
            Ð           # 1. City RegistrationÐ
self.logger.info("Registering City Data")Ð           await
self.city_module.register_city_data(city_data)Ð           Ð           # 2.
Company RegistrationÐ           self.logger.info("Registering Company Data")Ð
            await
self.company_module.register_company_data(company_data.to_dict('records'))Ð
            Ð           # 3. Emissions ProcessingÐ
self.logger.info("Processing Emissions Data")Ð           emissions_metrics =
await self.emissions_module.process_emissions_data(city_data)Ð           Ð
            # 4. Health Metrics CalculationÐ
```

```
self.logger.info("Calculating City Health Metrics")Ð           health_metrics
= await self.health_module.calculate_city_health(city_data)Ð           Ð
            # 5. Renewal Metrics CalculationÐ
self.logger.info("Calculating Renewal Metrics")Ð           renewal_metrics =
await self.renewal_module.calculate_renewal_metrics_workflow(Ð
city_data, Ð               company_dataÐ           )Ð           Ð           #
6. Generate ReportsÐ           self.logger.info("Generating Reports")Ð
            emissions_report =
self.emissions_module.generate_emissions_report(emissions_metrics)Ð
health_report = self.health_module.generate_health_report(health_metrics)Ð
            renewal_report =
self.renewal_module.generate_renewal_report(renewal_metrics)Ð           Ð
            # Log and return resultsÐ           self.logger.info("Climate
Workflow Completed Successfully")Ð           Ð           return {Ð
                'emissions_metrics': emissions_metrics,Ð
'health_metrics': health_metrics,Ð               'renewal_metrics':
renewal_metrics,Ð               'emissions_report': emissions_report,Ð
                'health_report': health_report,Ð
'renewal_report': renewal_reportÐ           }Ð       Ð       except Exception
as e:Ð           self.logger.error(f"Climate Workflow Error: {e}")Ð
raiseÐ   def _setup_logger(self):Ð       """Ð       Configure logging for the
workflowÐ       Ð       :return: Configured loggerÐ       """Ð       # Create
loggerÐ       logger = logging.getLogger('BlockchainWorkflow')Ð
logger.setLevel(logging.INFO)Ð       Ð       # Create log directory if it
doesn't existÐ       log_dir = os.path.join(self.project_root, 'logs')Ð
os.makedirs(log_dir, exist_ok=True)Ð       Ð       # File handlerÐ
file_handler = logging.FileHandler(Ð           os.path.join(log_dir,
'blockchain_workflow.log'),Ð           encoding='utf-8'Ð       )Ð
file_handler.setFormatter(logging.Formatter(Ð           '%(asctime)s -
%(name)s - %(levelname)s - %(message)s'Ð       ))Ð
logger.addHandler(file_handler)Ð       Ð       # Console handlerÐ
console_handler = logging.StreamHandler()Ð
console_handler.setFormatter(logging.Formatter(Ð           '%(name)s -
%(levelname)s - %(message)s'Ð       ))Ð
logger.addHandler(console_handler)Ð       Ð       return loggerÐ   def
_load_contract_addresses(self, addresses_config=None) -> Dict[str, str]:Ð
        """Ð       Load contract addresses from configurationÐ       Ð
        :param addresses_config: Optional path to addresses configÐ
        :return: Dictionary of contract addressesÐ       """Ð       # Default
addresses configuration pathÐ       if not addresses_config:Ð
addresses_config = os.path.join(Ð               self.project_root, 'config',
'contract_addresses.json'Ð           )Ð       Ð       try:Ð           with
open(addresses_config, 'r') as f:Ð               # Extract addresses,
handling both string and dict formatsÐ               config = json.load(f)Ð
                return {Ð                   key: (addr['address'] if
isinstance(addr, dict) else addr)Ð                   for key, addr in
config.items()Ð               }Ð       except FileNotFoundError:Ð
self.logger.warning("Contract addresses configuration not found. Using
default.")Ð           return {Ð               'CityRegister':
'0x17e9ddb311061ba9FA3a6ea517A934cc0D136f27',Ð
'CompanyRegister': '0x81aDCd0724dA5Da4b796d51c09123B53A4705D3F',Ð
                'CityEmissionsContract':
'0x723332E981Ddd2577954c0e15998e66A4929b1E8',Ð
'RenewalTheoryContract': '0x710CcD32bbD9b108ef3FdE8178F0E5dB94DCb478',Ð
                'CarbonCreditMarket':
'0x16F6278FBae0Fa873366628118425c34Bbb1C8c0',Ð
'CityHealthCalculator': '0xa87d6005E919f04324E7E351fa7d988E28C1Ef03',Ð
                'TemperatureRenewalContract':
'0xC2E69562926Ad558D982DD09238d64a60D515F48',Ð
```

```
'ClimateReduction': '0x3B076CD48b43d99A30C7857b37704C314C0e7171',Ð
                'MitigationContract':
'0xD78652eEe39D0bF625340a3CA0cE5696A7625d15'Ð           }Ð       except
json.JSONDecodeError:Ð           self.logger.error("Invalid contract
addresses configuration.")Ð           raiseÐ   def
_validate_contract_abi(self, contract_data: Dict) -> bool:Ð       """Ð
Validate contract ABI structureÐ       Ð       :param contract_data: Contract
JSON dataÐ       :return: Boolean indicating ABI validityÐ       """Ð
required_keys = ['abi', 'contractName']Ð       return all(key in
contract_data for key in required_keys) and \Ð
isinstance(contract_data['abi'], list)Ð   def _load_contracts(self):Ð
"""Ð       Load contract ABIs and create contract instances with enhanced
validationÐ       """Ð       for contract_name, address in
self.CONTRACT_ADDRESSES.items():Ð           try:Ð               # Construct
path to contract JSONÐ               contract_path =
os.path.join(self.contracts_dir, f'{contract_name}.json')Ð               Ð
                # Load contract artifactÐ               with
open(contract_path, 'r') as f:Ð                   contract_data = json.load(f)Ð
                Ð               # Validate ABIÐ               if not
self._validate_contract_abi(contract_data):Ð
self.logger.warning(f"Invalid ABI for {contract_name}")Ð
continueÐ               Ð               # Create contract instanceÐ
                contract_instance = self.w3.eth.contract(Ð
address=address,Ð                   abi=contract_data['abi']Ð               )Ð
                Ð               # Store contract instanceÐ
self.contracts[contract_name] = contract_instanceÐ               Ð
                self.logger.info(f"Loaded {contract_name} at {address}")Ð
            Ð           except FileNotFoundError:Ð
self.logger.warning(f"Contract artifact not found for {contract_name}")Ð
            except json.JSONDecodeError:Ð
self.logger.error(f"Invalid JSON for {contract_name}")Ð           except
Exception as e:Ð               self.logger.error(f"Error loading
{contract_name}: {e}")Ð   def get_contract(self, contract_name: str):Ð
"""Ð       Retrieve a specific contract instanceÐ       Ð       :param
contract_name: Name of the contractÐ       :return: Contract instance or NoneÐ
        """Ð       return self.contracts.get(contract_name)Ð   def
log_to_file(self, filename, data, receipt=None):Ð       """Ð       Log
transaction data to a JSON file with enhanced loggingÐ       Ð       :param
filename: Log file nameÐ       :param data: Data to logÐ       :param
receipt: Optional transaction receiptÐ       """Ð       try:Ð
log_dir = os.path.join(self.project_root, 'logs')Ð
os.makedirs(log_dir, exist_ok=True)Ð           Ð           log_path =
os.path.join(log_dir, filename)Ð           Ð           # Read existing logs
or create new listÐ           try:Ð               with open(log_path, 'r') as
f:Ð                   logs = json.load(f)Ð           except
(FileNotFoundError, json.JSONDecodeError):Ð               logs = []Ð
            Ð           # Prepare log entryÐ           log_entry = {Ð
                'timestamp': str(datetime.now()),Ð               'data': dataÐ
            }Ð           Ð           # Add transaction receipt if providedÐ
            if receipt:Ð               log_entry['transaction_hash'] =
receipt.transactionHash.hex()Ð               log_entry['gas_used'] =
receipt.gasUsedÐ               log_entry['block_number'] = receipt.blockNumberÐ
            Ð           logs.append(log_entry)Ð           Ð           # Write
updated logsÐ           with open(log_path, 'w') as f:Ð
json.dump(logs, f, indent=2)Ð       Ð       except Exception as e:Ð
self.logger.error(f"Error logging to {filename}: {e}")Ðdef main():Ð   try:Ð
        # Add the project root to Python path to ensure module imports workÐ
        project_root =
os.path.abspath(os.path.join(os.path.dirname(__file__), '..'))Ð
```

```
sys.path.insert(0, project_root)Đ       Đ       # Import workflow classĐ
        from python.blockchainworkflow import BlockchainWorkflowĐ       Đ
        # Initialize workflow with project rootĐ       workflow =
BlockchainWorkflow(Đ           project_root=project_root,Đ
provider_url='http://127.0.0.1:7545',  # Ensure this matches your local
blockchain providerĐ           addresses_config=os.path.join(project_root,
'config', 'contract_addresses.json')Đ       )Đ       Đ       # Run the
workflow asynchronouslyĐ       async def run_workflow():Đ           try:Đ
                # Run full workflowĐ               results = await
workflow.run_full_workflow()Đ               Đ               # Print out
detailed resultsĐ               print("\n===== Climate Workflow Results
=====")Đ               Đ               # Emissions MetricsĐ
print("\n--- Emissions Metrics ---")Đ
print(results.get('emissions_metrics', 'No emissions metrics available'))Đ
                Đ               # Emissions ReportĐ
print("\n--- Emissions Report ---")Đ
print(results.get('emissions_report', 'No emissions report available'))Đ
                Đ               # Health MetricsĐ               print("\n---
Health Metrics ---")Đ                   print(results.get('health_metrics', 'No
health metrics available'))Đ               Đ               # Health ReportĐ
                print("\n--- Health Report ---")Đ
print(results.get('health_report', 'No health report available'))Đ
                Đ               # Renewal MetricsĐ               print("\n---
Renewal Metrics ---")Đ                    print(results.get('renewal_metrics', 'No
renewal metrics available'))Đ               Đ               # Renewal ReportĐ
                print("\n--- Renewal Report ---")Đ
print(results.get('renewal_report', 'No renewal report available'))Đ
                Đ           except Exception as workflow_error:Đ
                print(f"Workflow Execution Error: {workflow_error}")Đ
                # Log the full tracebackĐ               import traceback‌Đ
                traceback.print_exc()Đ       Đ       # Run the async workflowĐ
        asyncio.run(run_workflow())Đ   Đ   except Exception as init_error:Đ
        print(f"Workflow Initialization Error: {init_error}")Đ       # Log
the full tracebackĐ       import traceback‌Đ       traceback.print_exc()Đ‌f
__name__ == "__main__":Đ   main()
```

## python\climatemodule.py

```
import loggingÐimport pandas as pdÐimport numpy as npÐfrom datetime import datetimeÐimport asyncioÐimport osÐfrom typing import List, Dict, Any, TupleÐÐclass CityModule:Ð   def __init__(self, workflow, config=None):Ð       """Ð        Initialize CityModule with workflow context and configurationÐ       Ð        :param workflow: BlockchainWorkflow instanceÐ       :param config: Configuration dictionary for module behaviorÐ       """Ð       self.workflow = workflowÐ       self.config = config or self._default_config()Ð       self.logger = self._setup_logger()Ð       self.data_path = os.path.join(self.workflow.project_root, 'data', 'carbonmonitor-cities_datas_2025-01-13.csv')Ð       self.transaction_semaphore = asyncio.Semaphore(Ð           self.config.get('max_concurrent_transactions', 5)Ð       )Ð   def _setup_logger(self):Ð       """Configure logging for the CityModule"""Ð       logger = logging.getLogger('CityModule')Ð       logger.setLevel(self.config['logging']['level'])Ð       Ð       log_dir = os.path.join(self.workflow.project_root, 'logs')Ð       os.makedirs(log_dir, exist_ok=True)Ð       Ð       log_file = os.path.join(log_dir, self.config['logging']['filename'])Ð       file_handler = logging.FileHandler(log_file)Ð       file_handler.setFormatter(logging.Formatter(Ð           '%(asctime)s - %(name)s - %(levelname)s - %(message)s'Ð       ))Ð       Ð       console_handler = logging.StreamHandler()Ð       console_handler.setFormatter(logging.Formatter(Ð           '%(name)s - %(levelname)s - %(message)s'Ð       ))Ð       Ð       logger.addHandler(file_handler)Ð       logger.addHandler(console_handler)Ð       Ð       return loggerÐ   def _default_config(self) -> Dict[str, Any]:Ð       """Provide default configuration for the module"""Ð       return {Ð           'max_concurrent_transactions': 5,Ð           'batch_size': 100,Ð           'validation_rules': {Ð               'required_columns': ['city', 'date', 'sector', 'value'],Ð               'value_range': {Ð                   'min': 0,Ð                   'max': 1000  # Adjusted for real emissions dataÐ               },Ð               'date_format': '%Y-%m-%d'  # Updated for standard CSV formatÐ           },Ð           'logging': {Ð               'level': logging.INFO,Ð               'filename': 'city_module.log'Ð           }Ð       }Ð   def load_and_validate_data(self) -> Tuple[pd.DataFrame, List[str]]:Ð       """Ð       Load and validate the carbon monitor cities dataÐ       Ð       :return: Tuple of (validated DataFrame, list of validation messages)Ð       """Ð       validation_messages = []Ð       Ð       try:Ð           # Load the CSV fileÐ           self.logger.info(f"Loading data from {self.data_path}")Ð           df = pd.read_csv(self.data_path)Ð           validation_messages.append(f"Successfully loaded {len(df)} records")Ð           # Basic data validationÐ           required_cols = self.config['validation_rules']['required_columns']Ð           missing_cols = [col for col in required_cols if col not in df.columns]Ð           Ð           if missing_cols:Ð               raise ValueError(f"Missing required columns: {missing_cols}")Ð           # Clean and transform dataÐ           df = self._clean_data(df)Ð           validation_messages.extend(self._validate_data(df))Ð           return df, validation_messagesÐ       except FileNotFoundError:Ð           msg = f"Data file not found at {self.data_path}"Ð           self.logger.error(msg)Ð           raise FileNotFoundError(msg)Ð       except Exception as e:Ð           msg = f"Error loading data: {str(e)}"Ð           self.logger.error(msg)Ð           raiseÐ   def _clean_data(self, df: pd.DataFrame) -> pd.DataFrame:Ð       """Ð       Clean and prepare the data for processingÐ       Ð       :param df: Raw DataFrameÐ       :return: Cleaned DataFrameÐ       """Ð       # Make a copy to avoid modifying original dataÐ       df = df.copy()Ð       Ð       # Convert date column to datetimeÐ       df['date'] = pd.to_datetime(df['date'])Ð       Ð       # Remove any rows with missing valuesÐ
```

```
                                df = df.dropna(subset=['city', 'value'])Ð
        Ð       # Remove any leading/trailing whitespaceÐ       df['city'] =
df['city'].str.strip()Ð       if 'sector' in df.columns:Ð
df['sector'] = df['sector'].str.strip()Ð       Ð       # Convert values to
float and handle any unit conversions if neededÐ       df['value'] =
pd.to_numeric(df['value'], errors='coerce')Ð       Ð       return dfÐ   def
_validate_data(self, df: pd.DataFrame) -> List[str]:Ð       """Ð
Validate the cleaned dataÐ       Ð       :param df: DataFrame to validateÐ
        :return: List of validation messagesÐ       """Ð       messages = []Ð
        rules = self.config['validation_rules']Ð       Ð       # Check value
rangesÐ       value_min = rules['value_range']['min']Ð       value_max =
rules['value_range']['max']Ð       out_of_range = df[Ð           (df['value']
< value_min) | Ð           (df['value'] > value_max)Ð       ]Ð       Ð
if not out_of_range.empty:Ð           messages.append(Ð               f"Found
{len(out_of_range)} values outside expected range "Ð
f"({value_min}-{value_max})"Ð           )Ð           # Log the problematic
valuesÐ           for _, row in out_of_range.iterrows():Ð
messages.append(Ð                   f"Unusual value for {row['city']}:
{row['value']} "Ð                   f"on {row['date']}"Ð               )Ð
        Ð       # Check for duplicate entriesÐ       duplicates =
df[df.duplicated(['city', 'date', 'sector'], keep=False)]Ð       if not
duplicates.empty:Ð           messages.append(Ð               f"Found
{len(duplicates)} duplicate entries"Ð           )Ð       Ð       # Add
summary statisticsÐ       messages.append(f"Total number of cities:
{df['city'].nunique()}")Ð       messages.append(f"Date range:
{df['date'].min()} to {df['date'].max()}")Ð       messages.append(f"Average
emission value: {df['value'].mean():.2f}")Ð       Ð       return messagesÐ
async def register_city_data(self, city_data=None):Ð       """Ð
Register city data on the blockchainÐ       Ð       :param city_data:
Optional DataFrame to use instead of loading from fileÐ       """Ð       try:Ð
            # Load and validate data if not providedÐ           if city_data
is None:Ð               city_data, validation_messages =
self.load_and_validate_data()Ð               for msg in validation_messages:Ð
                    self.logger.info(msg)Ð           # Validate contract
availabilityÐ           if 'CityRegister' not in self.workflow.contracts:Ð
                raise ValueError("CityRegister contract not loaded")Ð
            contract = self.workflow.contracts['CityRegister']Ð           Ð
            # Process unique citiesÐ           unique_cities =
city_data['city'].unique()Ð           Ð           for city in unique_cities:Ð
                try:Ð                   async with self.transaction_semaphore:Ð
                        # Prepare transaction parametersÐ
                        tx_params = {Ð                           'from':
self.workflow.w3.eth.accounts[0],Ð                           'gas': 2000000Ð
                        }Ð                       Ð                       #
Call contract method with correct string argumentÐ
tx_hash = contract.functions.registerCity(Ð                           city,
# Pass city as a stringÐ
datetime.now().strftime(self.config['validation_rules']['date_format']),Ð
                            'total',  # Default sectorÐ
                            0  # Default valueÐ
                        ).transact(tx_params)Ð                       Ð
                        # Wait for receiptÐ                       receipt =
await self.workflow.w3.eth.wait_for_transaction_receipt(tx_hash)Ð
                        Ð                       # Log transactionÐ
                        self.workflow.log_to_file(Ð
'city_register_logs.json',Ð                           {'city': city},Ð
                            receiptÐ                       )Ð
                        Ð                       self.logger.info(f"Registered
city: {city}")Ð               Ð               except Exception as
record_error:Ð
```

```
                                    self.logger.error(f"Error registering city
record: {record_error}")Ð                    continueÐ       except Exception
as e:Ð           self.logger.error(f"City data registration error: {e}")Ð
            raiseÐÐdef create_city_module(workflow, config=None):Ð   """Factory
method to create CityModule with optional custom configuration"""Ð   return
CityModule(workflow, config)
```

## python\companymodule.py

```
import osÐimport loggingÐimport pandas as pdÐfrom typing import Tuple, List,
Dict, AnyÐfrom web3 import Web3Ðfrom datetime import datetimeÐclass
CompanyModule:Ð    def __init__(self, workflow):Ð        """Ð        Initialize
CompanyModule with workflow contextÐ        Ð        :param workflow:
BlockchainWorkflow instanceÐ        """Ð        self.workflow = workflowÐ
self.logger = logging.getLogger('CompanyModule')Ð        self.data_path =
os.path.join(self.workflow.project_root, 'data',
'companies_data_2025-01-13.csv')Ð        Ð    def load_and_validate_data(self) -
> Tuple[pd.DataFrame, List[str]]:Ð        """Ð        Load and validate the
companies dataÐ        Ð        :return: Tuple of (validated DataFrame, list of
validation messages)Ð        """Ð        validation_messages = []Ð        Ð
        try:Ð            # Load the CSV fileÐ
self.logger.info(f"Loading data from {self.data_path}")Ð            df =
pd.read_csv(self.data_path)Ð
validation_messages.append(f"Successfully loaded {len(df)} records")Ð
            # Required columnsÐ            required_cols = ['company_name',
'registration_date', 'sector', 'emissions_baseline']Ð            missing_cols
= [col for col in required_cols if col not in df.columns]Ð            Ð
            if missing_cols:Ð                raise ValueError(f"Missing
required columns: {missing_cols}")Ð            # Clean and transform dataÐ
            df = self._clean_data(df)Ð
validation_messages.extend(self._validate_data(df))Ð            return df,
validation_messagesÐ        except FileNotFoundError:Ð            msg = f"Data
file not found at {self.data_path}"Ð            self.logger.error(msg)Ð
            raise FileNotFoundError(msg)Ð        except Exception as e:Ð
            msg = f"Error loading data: {str(e)}"Ð
self.logger.error(msg)Ð            raiseÐ    def _clean_data(self, df:
pd.DataFrame) -> pd.DataFrame:Ð        """Ð        Clean and prepare the data
for processingÐ        Ð        :param df: Raw DataFrameÐ        :return:
Cleaned DataFrameÐ        """Ð        # Make a copy to avoid modifying original
dataÐ        df = df.copy()Ð        Ð        # Convert registration date to
datetimeÐ        df['registration_date'] =
pd.to_datetime(df['registration_date'])Ð        Ð        # Remove any rows with
missing valuesÐ        df = df.dropna(subset=['company_name',
'emissions_baseline'])Ð        Ð        # Remove any leading/trailing whitespaceÐ
        df['company_name'] = df['company_name'].str.strip()Ð
df['sector'] = df['sector'].str.strip()Ð        Ð        # Convert emissions
baseline to floatÐ        df['emissions_baseline'] =
pd.to_numeric(df['emissions_baseline'], errors='coerce')Ð        Ð
return dfÐ    def _validate_data(self, df: pd.DataFrame) -> List[str]:Ð
"""Ð        Validate the cleaned dataÐ        Ð        :param df: DataFrame to
validateÐ        :return: List of validation messagesÐ        """Ð
messages = []Ð        Ð        # Check value ranges for emissions baselineÐ
        value_min = 0Ð        value_max = 1000  # Adjust as neededÐ
out_of_range = df[Ð            (df['emissions_baseline'] < value_min) | Ð
            (df['emissions_baseline'] > value_max)Ð        ]Ð        Ð        if
not out_of_range.empty:Ð            messages.append(Ð                f"Found
{len(out_of_range)} values outside expected range "Ð
f"({value_min}-{value_max})"Ð            )Ð            # Log the problematic
valuesÐ            for _, row in out_of_range.iterrows():Ð
messages.append(Ð                    f"Unusual emissions baseline for
{row['company_name']}: {row['emissions_baseline']}"Ð                )Ð        Ð
        # Check for duplicate entriesÐ        duplicates =
df[df.duplicated(['company_name', 'registration_date'], keep=False)]Ð
if not duplicates.empty:Ð            messages.append(Ð                f"Found
{len(duplicates)} duplicate entries"Ð            )Ð        Ð        # Add
summary statisticsÐ        messages.append(f"Total number of companies:
{df['company_name'].nunique()}")Ð        messages.append(f"Date range:
{df['registration_date'].min()} to {df['registration_date'].max()}")Ð
```

```
messages.append(f"Average emissions baseline:
{df['emissions_baseline'].mean():.2f}")Đ       Đ       return messagesĐ
async def register_company_data(self, company_data=None):Đ       """Đ
Register company data on the blockchainĐ       Đ       :param company_data:
Optional DataFrame to use instead of loading from fileĐ       """Đ       try:Đ
            # Load and validate data if not providedĐ           if
company_data is None:Đ               company_data, validation_messages =
self.load_and_validate_data()Đ               for msg in validation_messages:Đ
                    self.logger.info(msg)Đ           # Validate contract
availabilityĐ           if 'CompanyRegister' not in self.workflow.contracts:Đ
                raise ValueError("CompanyRegister contract not loaded")Đ
            contract = self.workflow.contracts['CompanyRegister']Đ           Đ
            # Convert to list of dictionaries if DataFrameĐ           if
isinstance(company_data, pd.DataFrame):Đ               company_data =
company_data.to_dict('records')Đ           Đ           # Process each company
recordĐ           for record in company_data:Đ                 try:Đ
                    # Prepare transaction parametersĐ
tx_params = {Đ                        'from': self.workflow.w3.eth.accounts[0],Đ
                        'gas': 2000000Đ                    }Đ
                    Đ                    # Convert registration date to
timestampĐ                    registration_timestamp =
int(pd.to_datetime(record['registration_date']).timestamp())Đ
                    Đ                    # Prepare company arguments Đ
                    tx_hash = contract.functions.registerCompany(Đ
                        self.workflow.w3.eth.accounts[0],  # Company address Đ
                        self.workflow.w3.eth.accounts[0],  # Owner addressĐ
                        registration_timestamp,  # Registration timestampĐ
                        int(record['emissions_baseline'] * 1000),  # Baseline
emissions (scaled)Đ                        0,  # Longitude (if applicable)Đ
                        0   # Latitude (if applicable)Đ
                    ).transact(tx_params)Đ                   Đ
                    # Wait for receiptĐ                   receipt = await
self.workflow.w3.eth.wait_for_transaction_receipt(tx_hash)Đ                   Đ
                    # Log transactionĐ
self.workflow.log_to_file('company_register_logs.json', record, receipt)Đ
                    Đ                    self.logger.info(f"Registered company
{record['company_name']} in {record.get('sector', 'unknown')} sector")Đ
                Đ                except Exception as record_error:Đ
                    self.logger.error(f"Error registering company record:
{record_error}")Đ                    # Continue processing other recordsĐ
                    continueĐ       except Exception as e:Đ
self.logger.error(f"Error in company data registration: {str(e)}")Đ
raiseĐdef create_company_module(workflow, config=None):Đ   """Đ   Factory
method to create CompanyModuleĐ   Đ   :param workflow: BlockchainWorkflow
instanceĐ   :return: CompanyModule instanceĐ   """Đ   return
CompanyModule(workflow)
```

## python\deploy_contracts.py

```
import osÐimport jsonÐimport subprocessÐimport loggingÐfrom web3 import Web3Ðclass
ContractDeployer:Ð   def __init__(self, network="development"):Ð       #
Setup loggingÐ       logging.basicConfig(level=logging.INFO)Ð
self.logger = logging.getLogger(__name__)Ð       Ð       # Setup Web3
connectionÐ       if network == "development":Ð           self.w3 =
Web3(Web3.HTTPProvider('http://127.0.0.1:7545'))Ð       else:Ð
raise ValueError(f"Unsupported network: {network}")Ð       Ð       # Project
pathsÐ       self.project_root =
os.path.abspath(os.path.join(os.path.dirname(__file__), '..'))Ð
self.contracts_build_dir = os.path.join(self.project_root, 'build',
'contracts')Ð       Ð   def run_truffle_migrate(self):Ð       """Ð       Run
Truffle migration to deploy contractsÐ       """Ð       try:Ð           #
Ensure we're in the project rootÐ           os.chdir(self.project_root)Ð
            Ð           # Run Truffle migrateÐ           result =
subprocess.run(Ð               ['truffle', 'migrate', '--reset', '--network',
'development'], Ð               capture_output=True, Ð               text=TrueÐ
            )Ð           Ð           if result.returncode != 0:Ð
                self.logger.error("Truffle migration failed")Ð
self.logger.error(result.stderr)Ð               raise Exception("Truffle
migration failed")Ð           Ð           self.logger.info("Truffle migration
completed successfully")Ð       except Exception as e:Ð
self.logger.error(f"Deployment error: {e}")Ð           raiseÐ   def
update_contract_addresses(self):Ð       """Ð       Update contract addresses
in configuration filesÐ       """Ð       try:Ð           # Network ID for
local GanacheÐ           network_id = self.w3.net.versionÐ           Ð
            # Contracts to trackÐ           contract_names = [Ð
'CityRegister', 'CompanyRegister', 'CityEmissionsContract', Ð
'RenewalTheoryContract', 'CityHealthCalculator',Ð
'TemperatureRenewalContract', 'MitigationContract'Ð           ]Ð           Ð
            # Configuration to store addressesÐ           contract_addresses
= {}Ð           Ð           # Read contract addresses from build artifactsÐ
            for contract_name in contract_names:Ð               contract_path
= os.path.join(self.contracts_build_dir, f'{contract_name}.json')Ð
                Ð               if os.path.exists(contract_path):Ð
                    with open(contract_path, 'r') as f:Ð
                        contract_data = json.load(f)Ð                   Ð
                    # Get contract address for the current networkÐ
                    if network_id in contract_data.get('networks', {}):Ð
                        contract_addresses[contract_name] =
contract_data['networks'][network_id]['address']Ð
self.logger.info(f"Loaded address for {contract_name}")Ð
else:Ð                       self.logger.warning(f"No address found for
{contract_name}")Ð           Ð           # Write addresses to a configuration
fileÐ           config_path = os.path.join(self.project_root, 'config',
'contract_addresses.json')Ð           with open(config_path, 'w') as f:Ð
                json.dump(contract_addresses, f, indent=2)Ð           Ð
            self.logger.info(f"Contract addresses saved to {config_path}")Ð
            Ð           return contract_addressesÐ       Ð       except
Exception as e:Ð           self.logger.error(f"Error updating contract
addresses: {e}")Ð           raiseÐdef main():Ð   try:Ð       # Initialize
deployerÐ       deployer = ContractDeployer()Ð       Ð       # Run Truffle
migrationÐ       deployer.run_truffle_migrate()Ð       Ð       # Update and
save contract addressesÐ       contract_addresses =
deployer.update_contract_addresses()Ð       Ð       print("Deployment
completed successfully")Ð       print("Deployed Contract Addresses:",
json.dumps(contract_addresses, indent=2))Ð   Ð   except Exception as e:Ð
        print(f"Deployment failed: {e}")Ðif __name__ == "__main__":Ð   main()
```

## python\emmissionsmodule.py

```
import loggingÐimport pandas as pdÐimport numpy as npÐfrom datetime import datetimeÐimport asyncioÐfrom typing import List, Dict, AnyÐÐclass EmissionsModule:Ð    def __init__(self, workflow, config=None):Ð        """Ð        Initialize EmissionsModule with workflow context and configurationÐ        Ð        :param workflow: BlockchainWorkflow instanceÐ        :param config: Configuration dictionary for module behaviorÐ        """Ð        self.workflow = workflowÐ        self.config = config or self._default_config()Ð        Ð        # Setup specialized loggerÐ        self.logger = self._setup_logger()Ð        Ð        # Transaction rate limitingÐ        self.transaction_semaphore = asyncio.Semaphore(Ð            self.config.get('max_concurrent_transactions', 5)Ð        )Ð    def _default_config(self) -> Dict[str, Any]:Ð        """Ð        Provide default configuration for the moduleÐ        Ð        :return: Default configuration dictionaryÐ        """Ð        return {Ð            'max_concurrent_transactions': 5,Ð            'batch_size': 100,Ð            'validation_rules': {Ð                'required_columns': ['city', 'date', 'sector', 'value'],Ð                'value_range': {Ð                    'min': 0,Ð                    'max': 100  # Adjust based on your emissions scaleÐ                },Ð                'date_format': '%d/%m/%Y'Ð            },Ð            'aggregation_strategy': {Ð                'method': 'sum',Ð                'groupby_columns': ['city', 'date', 'sector']Ð            },Ð            'logging': {Ð                'level': logging.INFO,Ð                'filename': 'emissions_module.log'Ð            }Ð        }Ð    def _setup_logger(self) -> logging.Logger:Ð        """Ð        Set up a specialized logger for the moduleÐ        Ð        :return: Configured loggerÐ        """Ð        logger = logging.getLogger('EmissionsModule')Ð        logger.setLevel(self.config['logging']['level'])Ð        Ð        # File handlerÐ        log_dir = self.workflow.project_root + '/logs'Ð        file_handler = logging.FileHandler(Ð            f'{log_dir}/{self.config["logging"]["filename"]}', Ð            encoding='utf-8'Ð        )Ð        file_handler.setFormatter(logging.Formatter(Ð            '%(asctime)s - %(name)s - %(levelname)s - %(message)s'Ð        ))Ð        logger.addHandler(file_handler)Ð        Ð        return loggerÐ    async def process_emissions_data(self, city_data):Ð        """Ð        Process emissions data on the blockchain with advanced processingÐ        Ð        :param city_data: Emissions dataÐ        """Ð        try:Ð            # Validate contract availabilityÐ            if 'CityEmissionsContract' not in self.workflow.contracts:Ð                raise ValueError("CityEmissionsContract not loaded")Ð            Ð            # Convert to DataFrame if not alreadyÐ            if not isinstance(city_data, pd.DataFrame):Ð                city_data = pd.DataFrame(city_data)Ð            Ð            # Validate dataÐ            validated_data = self.validate_data(city_data)Ð            Ð            # Optional: Aggregate emissions if neededÐ            aggregated_data = validated_data.groupby(['city', 'date'])['value'].sum().reset_index()Ð            Ð            self.logger.info(f"Processing {len(aggregated_data)} aggregated emissions records")Ð            Ð            contract = self.workflow.contracts['CityEmissionsContract']Ð            Ð            for _, row in aggregated_data.iterrows():Ð                try:Ð                    # Prepare transaction parametersÐ                    tx_params = {Ð                        'from': self.workflow.w3.eth.accounts[0],Ð                        'gas': 2000000Ð                    }Ð                    Ð                    # Convert date to timestampÐ                    date_timestamp = int(row['date'].timestamp())Ð                    Ð                    # Process emissions transactionÐ                    tx_hash = await contract.functions.processEmissions(Ð                        row['city'], Ð                        date_timestamp, Ð                        int(row['value'] * 1000)  # Convert to integer (scaled)Ð                    ).transact(tx_params)Ð                    Ð                    # Wait for transaction receiptÐ
```

```
                                                                    receipt
= await self.workflow.w3.eth.wait_for_transaction_receipt(tx_hash)Ð
                    Ð                   # Log transactionÐ
self.workflow.log_to_file('emissions_processing_logs.json', row.to_dict(),
receipt)Ð                   Ð                   self.logger.info(f"Processed
emissions for {row['city']} on {row['date']}")Ð               Ð
except Exception as record_error:Ð
self.logger.error(f"Error processing emissions record: {record_error}")Ð
                    continueÐ           Ð
self.logger.info("Emissions data processing completed")Ð           Ð
            return aggregated_dataÐ       Ð       except Exception as e:Ð
            self.logger.error(f"Comprehensive emissions data processing
error: {e}")Ð           raiseÐ   def validate_data(self, df: pd.DataFrame) ->
pd.DataFrame:Ð       """Ð       Validate input emissions dataÐ       Ð
        :param df: Input DataFrameÐ       :return: Validated DataFrameÐ
"""Ð       # Validate required columnsÐ       required_columns = ['city',
'date', 'sector', 'value']Ð       missing_columns = set(required_columns) -
set(df.columns)Ð       if missing_columns:Ð           raise
ValueError(f"Missing required columns: {missing_columns}")Ð       Ð       #
Ensure date is datetimeÐ       df['date'] = pd.to_datetime(df['date'])Ð       Ð
        # Validate numeric valuesÐ       df['value'] =
pd.to_numeric(df['value'], errors='raise')Ð       Ð       # Check value range
(adjust as needed)Ð       value_min = 0Ð       value_max = 1000Ð
out_of_range = df[(df['value'] < value_min) | (df['value'] > value_max)]Ð
        Ð       if not out_of_range.empty:Ð
self.logger.warning(f"Found {len(out_of_range)} values outside expected
range")Ð           # Optionally, log or filter out these valuesÐ           df
= df[(df['value'] >= value_min) & (df['value'] <= value_max)]Ð       Ð
return dfÐ   Ð   def generate_emissions_report(self, emissions_metrics:
pd.DataFrame) -> Dict[str, Any]:Ð       """Ð       Generate a comprehensive
emissions reportÐ       Ð       :param emissions_metrics: DataFrame with
emissions metricsÐ       :return: Summary report dictionaryÐ       """Ð
try:Ð           # Ensure input is a DataFrameÐ           if not
isinstance(emissions_metrics, pd.DataFrame):Ð               emissions_metrics
= pd.DataFrame(emissions_metrics)Ð           Ð           # Prepare report
dictionaryÐ           report = {Ð               'total_cities_analyzed':
len(emissions_metrics['city'].unique()),Ð               'total_emissions':
emissions_metrics['value'].sum(),Ð               'average_emissions':
emissions_metrics['value'].mean(),Ð               'highest_emission_city':
emissions_metrics.loc[Ð
emissions_metrics['value'].idxmax(), 'city'Ð               ],Ð
'lowest_emission_city': emissions_metrics.loc[Ð
emissions_metrics['value'].idxmin(), 'city'Ð               ],Ð
'emissions_by_city': emissions_metrics.groupby('city')
['value'].sum().to_dict(),Ð               'date_range': {Ð
'start': emissions_metrics['date'].min(),Ð                   'end':
emissions_metrics['date'].max()Ð               }Ð           }Ð           Ð
            self.logger.info("Generated comprehensive emissions report")Ð
            return reportÐ       Ð       except Exception as e:Ð
self.logger.error(f"Error generating emissions report: {e}")Ð
return {}Ðdef create_emissions_module(workflow, config=None):Ð   """Ð
Factory method to create EmissionsModule with custom configurationÐ   Ð
    :param workflow: BlockchainWorkflow instanceÐ   :return: Configured
EmissionsModule instanceÐ   """Ð   return EmissionsModule(workflow, config)
```

## python\healthmodule.py

```
import loggingÐimport pandas as pdÐimport numpy as npÐfrom datetime import
datetimeÐimport asyncioÐfrom typing import List, Dict, AnyÐfrom
sklearn.linear_model import LinearRegressionÐclass HealthModule:Ð   def
__init__(self, workflow=None, config=None):Ð       """Ð       Initialize
HealthModule with workflow context and configurationÐ       Ð       :param
workflow: BlockchainWorkflow instanceÐ       :param config: Configuration
dictionary for module behaviorÐ       """Ð       # Store workflow instanceÐ
        self.workflow = workflowÐ       Ð       self.config = config or
self._default_config()Ð       Ð       # Setup specialized loggerÐ
self.logger = self._setup_logger()Ð       Ð       # Transaction rate limitingÐ
        self.transaction_semaphore = asyncio.Semaphore(Ð
self.config.get('max_concurrent_transactions', 5)Ð       )Ð   def
_default_config(self) -> Dict[str, Any]:Ð       """Ð       Provide default
configuration for the moduleÐ       Ð       :return: Default configuration
dictionaryÐ       """Ð       return {Ð
'max_concurrent_transactions': 5,Ð           'batch_size': 100,Ð
'validation_rules': {Ð               'required_columns': ['city', 'date',
'sector', 'value'],Ð               'value_range': {Ð                   'min':
0,Ð                   'max': 100  # Adjust based on your emissions scaleÐ
               },Ð               'date_format': '%d/%m/%Y'Ð           },Ð
            'health_metrics': {Ð               'total_emissions': 'sum',Ð
               'variance': 'var',Ð               'peak_emission': 'max',Ð
               'emission_trend': 'linear_regression'Ð           },Ð
            'logging': {Ð               'level': logging.INFO,Ð
'filename': 'health_module.log'Ð           }Ð       }Ð   def
_setup_logger(self) -> logging.Logger:Ð       """Ð       Set up a specialized
logger for the moduleÐ       Ð       :return: Configured loggerÐ       """Ð
        logger = logging.getLogger('HealthModule')Ð
logger.setLevel(self.config['logging']['level'])Ð       Ð       # File handlerÐ
        file_handler = logging.FileHandler(Ð           self.config['logging']
['filename'], Ð           encoding='utf-8'Ð       )Ð
file_handler.setFormatter(logging.Formatter(Ð           '%(asctime)s -
%(name)s - %(levelname)s - %(message)s'Ð       ))Ð
logger.addHandler(file_handler)Ð       Ð       return loggerÐ   def
validate_data(self, df: pd.DataFrame) -> pd.DataFrame:Ð       """Ð
Validate input data based on configuration rulesÐ       Ð       :param df:
Input DataFrameÐ       :return: Validated DataFrameÐ       """Ð
validation_rules = self.config['validation_rules']Ð       Ð       # Check
required columnsÐ       missing_columns =
set(validation_rules['required_columns']) - set(df.columns)Ð       if
missing_columns:Ð           raise ValueError(f"Missing columns:
{missing_columns}")Ð       Ð       try:Ð           # Convert date to
specified formatÐ           df['date'] = pd.to_datetime(Ð
df['date'], Ð               format=validation_rules['date_format']Ð
            ).dt.strftime('%d/%m/%Y')Ð           Ð           # Validate
numeric valuesÐ           df['value'] = pd.to_numeric(df['value'],
errors='raise')Ð           Ð           # Check value rangeÐ
value_range = validation_rules['value_range']Ð           invalid_values = df[Ð
               (df['value'] < value_range['min']) | Ð
(df['value'] > value_range['max'])Ð           ]Ð           Ð           if not
invalid_values.empty:Ð               self.logger.warning(f"Found
{len(invalid_values)} invalid value records")Ð               df = df[Ð
                   (df['value'] >= value_range['min']) & Ð
(df['value'] <= value_range['max'])Ð               ]Ð       Ð       except
Exception as e:Ð           self.logger.error(f"Data validation error: {e}")Ð
            raiseÐ       Ð       return dfÐ   def
calculate_health_metrics(self, df: pd.DataFrame) -> pd.DataFrame:Ð       """Ð
        Calculate comprehensive health metrics for citiesÐ       Ð
        :param df: Input DataFrame with city emissions dataÐ
```

```
                                                                    :return:
DataFrame with health metricsÐ       """Ð       # Validate input dataÐ
validated_df = self.validate_data(df)Ð       Ð       health_metrics = []Ð
        Ð       for city in validated_df['city'].unique():Ð
city_data = validated_df[validated_df['city'] == city].copy()  # Create a copyÐ
            Ð           # Basic metricsÐ           total_emissions =
city_data['value'].sum()Ð           variance = city_data['value'].var()Ð
            peak_emission = city_data['value'].max()Ð           Ð           #
Trend analysis (linear regression)Ð           try:Ð               # Convert
dates to numeric for regressionÐ               city_data.loc[:,
'date_numeric'] = pd.to_datetime(city_data['date']).astype(int) // 10**9Ð
                Ð               X =
city_data['date_numeric'].values.reshape(-1, 1)Ð               y =
city_data['value'].valuesÐ               Ð               model =
LinearRegression()Ð               model.fit(X, y)Ð               Ð
                trend_slope = model.coef_[0]Ð               trend_intercept =
model.intercept_Ð           except Exception as trend_error:Ð
self.logger.warning(f"Trend analysis failed for {city}: {trend_error}")Ð
                trend_slope = NoneÐ               trend_intercept = NoneÐ
            Ð           # Compile metricsÐ           city_metrics = {Ð
                'city': city,Ð               'total_emissions':
total_emissions,Ð               'variance': variance,Ð
'peak_emission': peak_emission,Ð               'trend_slope': trend_slope,Ð
                'trend_intercept': trend_interceptÐ           }Ð           Ð
            health_metrics.append(city_metrics)Ð       Ð       return
pd.DataFrame(health_metrics)Ð   async def calculate_city_health(self,
city_data):Ð       """Ð       Calculate and register city health metrics on
the blockchainÐ       Ð       :param city_data: City emissions dataÐ       """Ð
        try:Ð           # Validate contract availabilityÐ           if
'CityHealthCalculator' not in self.workflow.contracts:Ð               raise
ValueError("CityHealthCalculator contract not loaded")Ð           Ð
# Convert to DataFrame if not alreadyÐ           if not isinstance(city_data,
pd.DataFrame):Ð               city_data = pd.DataFrame(city_data)Ð           Ð
            # Calculate health metricsÐ           health_metrics =
self.calculate_health_metrics(city_data)Ð           Ð           # Get the
contract instanceÐ           health_contract =
self.workflow.contracts['CityHealthCalculator']Ð           Ð           #
Process each city's health metricsÐ           for _, row in
health_metrics.iterrows():Ð               try:Ð                   # Interact
with contract to calculate healthÐ                   tx_hash = await
health_contract.functions.calculateCityHealth(Ð
row['city'], Ð                       float(row['total_emissions']), Ð
                    float(row['variance']), Ð
float(row['peak_emission'])Ð                   ).transact({Ð
                    'from': self.workflow.w3.eth.accounts[0],Ð
                    'gas': 2000000Ð                   })Ð
                Ð                   # Wait for transaction receiptÐ
                receipt = await
self.workflow.w3.eth.wait_for_transaction_receipt(tx_hash)Ð                   Ð
                # Log the transactionÐ
self.workflow.log_to_file('city_health_logs.json', row.to_dict(), receipt)Ð
                Ð                   self.logger.info(f"Processed health
metrics for {row['city']}")Ð               Ð               except Exception
as record_error:Ð                   self.logger.error(f"Error processing
health metrics for {row['city']}: {record_error}")Ð                   continueÐ
            Ð           return health_metricsÐ       Ð       except Exception
as e:Ð           self.logger.error(f"Comprehensive city health calculation
error: {e}")Ð           raiseÐ   def generate_health_report(self,
health_metrics: pd.DataFrame) -> Dict[str, Any]:Ð       """Ð       Generate a
comprehensive health reportÐ
```

```
                                Đ       :param health_metrics: DataFrame
with health metricsĐ       :return: Summary report dictionaryĐ       """Đ
        try:Đ           report = {Đ               'total_cities_analyzed':
len(health_metrics),Đ               'global_total_emissions':
health_metrics['total_emissions'].sum(),Đ
'global_average_emissions': health_metrics['total_emissions'].mean(),Đ
                'cities_with_increasing_trend': len(Đ
health_metrics[health_metrics['trend_slope'] > 0]Đ               ),Đ
                'cities_with_decreasing_trend': len(Đ
health_metrics[health_metrics['trend_slope'] < 0]Đ               ),Đ
                'highest_emission_city': health_metrics.loc[Đ
                    health_metrics['total_emissions'].idxmax()Đ
                ]['city'],Đ               'lowest_emission_city':
health_metrics.loc[Đ
health_metrics['total_emissions'].idxmin()Đ               ]['city']Đ
            }Đ           Đ           self.logger.info("Generated
comprehensive health report")Đ           return reportĐ       Đ       except
Exception as e:Đ           self.logger.error(f"Error generating health
report: {e}")Đ           return {}Đ Optional: Configuration customization
exampleĐdef create_health_module(workflow):Đ   """Đ   Factory method to create
HealthModule with custom configurationĐ   Đ   :param workflow:
BlockchainWorkflow instanceĐ   :return: Configured HealthModule instanceĐ
"""Đ   custom_config = {Đ       'max_concurrent_transactions': 3,Đ
'batch_size': 50,Đ       'validation_rules': {Đ           'required_columns':
['city', 'date', 'sector', 'value'],Đ           'value_range': {'min': 0,
'max': 50},Đ           'date_format': '%d/%m/%Y'Đ       }Đ   }Đ   return
HealthModule(workflow, config=custom_config)Đ Example usageĐasync def main():Đ
    # Create workflowĐ   workflow = BlockchainWorkflow()Đ   Đ   # Create
health moduleĐ   health_module = HealthModule(workflow)Đ   Đ   # Sample data
(replace with actual data loading)Đ   sample_data = pd.DataFrame({Đ
'city': ['CityA', 'CityA', 'CityB', 'CityB'],Đ       'date': ['01/01/2023',
'02/01/2023', '01/01/2023', '02/01/2023'],Đ       'sector': ['Energy',
'Energy', 'Transport', 'Transport'],Đ       'value': [10.5, 12.3, 8.7, 9.2]Đ
    })Đ   Đ   # Calculate and register health metricsĐ   health_metrics =
await health_module.calculate_city_health(sample_data)Đ   Đ   # Generate
health reportĐ   report = health_module.generate_health_report(health_metrics)Đ
    print(report)Đif __name__ == "__main__":Đ   import asyncioĐ
asyncio.run(main())
```

## python\renewalmodule.py

```
import loggingÐimport pandas as pdÐimport numpy as npÐfrom datetime import
datetimeÐimport asyncioÐfrom typing import List, Dict, AnyÐÐclass RenewalModule:Ð
    def __init__(self, workflow=None, config=None):Ð        """Ð
Initialize RenewalModule with workflow context and configurationÐ        Ð
        :param workflow: BlockchainWorkflow instanceÐ        :param config:
Configuration dictionary for module behaviorÐ        """Ð        # Store
workflow instanceÐ        self.workflow = workflowÐ        Ð        self.config
= config or self._default_config()Ð        Ð        # Setup specialized loggerÐ
        self.logger = self._setup_logger()Ð        Ð        # Transaction rate
limitingÐ        self.transaction_semaphore = asyncio.Semaphore(Ð
self.config.get('max_concurrent_transactions', 5)Ð        )ÐÐ   def
_default_config(self) -> Dict[str, Any]:Ð        """Ð        Provide default
configuration for the moduleÐ        Ð        :return: Default configuration
dictionaryÐ        """Ð        return {Ð
'max_concurrent_transactions': 5,Ð            'batch_size': 100,Ð
'validation_rules': {Ð                'required_columns': ['city', 'date',
'sector', 'value'],Ð                'value_range': {Ð                    'min':
0,Ð                    'max': 100  # Adjust based on your emissions scaleÐ
                },Ð                'date_format': '%d/%m/%Y'Ð            },Ð
            'renewal_metrics': {Ð                'emissions_reduction_target':
0.1,  # 10% reduction targetÐ                'time_window_days': 365  # Annual
analysisÐ            },Ð            'logging': {Ð                'level':
logging.INFO,Ð                'filename': 'renewal_module.log'Ð            }Ð
        }ÐÐ   def _setup_logger(self) -> logging.Logger:Ð        """Ð        Set
up a specialized logger for the moduleÐ        Ð        :return: Configured
loggerÐ        """Ð        logger = logging.getLogger('RenewalModule')Ð
logger.setLevel(self.config['logging']['level'])Ð        Ð        # File handlerÐ
        file_handler = logging.FileHandler(Ð            self.config['logging']
['filename'], Ð            encoding='utf-8'Ð        )Ð
file_handler.setFormatter(logging.Formatter(Ð            '%(asctime)s -
%(name)s - %(levelname)s - %(message)s'Ð        ))Ð
logger.addHandler(file_handler)Ð        Ð        return loggerÐÐ   def
validate_data(self, df: pd.DataFrame) -> pd.DataFrame:Ð        """Ð
Validate input data based on configuration rulesÐ        Ð        :param df:
Input DataFrameÐ        :return: Validated DataFrameÐ        """Ð        # If
None is passed, return an empty DataFrameÐ        if df is None:Ð
return pd.DataFrame(columns=self.config['validation_rules']
['required_columns'])Ð        Ð        validation_rules =
self.config['validation_rules']Ð        Ð        # Check required columnsÐ
        missing_columns = set(validation_rules['required_columns']) -
set(df.columns)Ð        Ð        # Add missing columns with default values if
necessaryÐ        for col in missing_columns:Ð            if col == 'date':Ð
                df['date'] = pd.Timestamp.now()Ð            elif col == 'city':Ð
                df['city'] = 'Unknown'Ð            elif col == 'sector':Ð
                df['sector'] = 'total'Ð            elif col == 'value':Ð
                df['value'] = 0.0Ð        Ð        try:Ð            # Convert
date to specified formatÐ            df['date'] = pd.to_datetime(Ð
                df['date'], Ð
format=validation_rules['date_format']Ð            ).dt.strftime('%d/%m/%Y')Ð
            Ð            # Validate numeric valuesÐ            df['value'] =
pd.to_numeric(df['value'], errors='coerce').fillna(0)Ð            Ð
# Check value rangeÐ            value_range = validation_rules['value_range']Ð
            invalid_values = df[Ð                (df['value'] <
value_range['min']) | Ð                (df['value'] > value_range['max'])Ð
            ]Ð            Ð            if not invalid_values.empty:Ð
                self.logger.warning(f"Found {len(invalid_values)} invalid
value records")Ð                df.loc[Ð                    (df['value'] <
value_range['min']) | Ð                    (df['value'] > value_range['max']), Ð
                    'value'Ð
```

```
                                        ] = 0Ð       Ð       except
Exception as e:Ð           self.logger.error(f"Data validation error: {e}")Ð
            raiseÐ       Ð       return dfÐ   def
calculate_renewal_metrics(self, city_data: pd.DataFrame, company_data:
pd.DataFrame = None) -> pd.DataFrame:Ð       """Ð       Calculate
comprehensive renewal metricsÐ       Ð       :param city_data: City emissions
DataFrameÐ       :param company_data: Optional company emissions DataFrameÐ
        :return: DataFrame with renewal metricsÐ       """Ð       # Validate
input dataÐ       validated_city_data = self.validate_data(city_data)Ð       Ð
        # Optional company data validationÐ       validated_company_data =
NoneÐ       if company_data is not None:Ð           validated_company_data =
self.validate_data(company_data)Ð       Ð       renewal_metrics = []Ð
renewal_config = self.config['renewal_metrics']Ð       Ð       for city in
validated_city_data['city'].unique():Ð           try:Ð               # City-
specific emissions dataÐ               city_subset =
validated_city_data[validated_city_data['city'] == city]Ð               Ð
                # Calculate total emissions and emissions by sectorÐ
                total_city_emissions = city_subset['value'].sum()Ð
                emissions_by_sector = city_subset.groupby('sector')
['value'].sum()Ð               Ð               # Company emissions for the
city (if available)Ð               company_emissions = 0Ð               if
validated_company_data is not None:Ð                   city_companies =
validated_company_data[validated_company_data['city'] == city]Ð
                    if not city_companies.empty:Ð
company_emissions = city_companies['value'].sum()Ð             Ð
                # Calculate renewal potentialÐ               reduction_target
= total_city_emissions * renewal_config['emissions_reduction_target']Ð
                Ð               # Sector-specific reduction potentialÐ
                sector_reduction_potential = {Ð                   sector:
emissions * renewal_config['emissions_reduction_target']Ð
for sector, emissions in emissions_by_sector.items()Ð             }Ð
                Ð               # Compile renewal metricsÐ
city_renewal_metrics = {Ð                   'city': city,Ð
'total_city_emissions': total_city_emissions,Ð
'emissions_by_sector': emissions_by_sector.to_dict(),Ð
'company_emissions': company_emissions,Ð
'total_reduction_target': reduction_target,Ð
'sector_reduction_potential': sector_reduction_potentialÐ               }Ð
                Ð               renewal_metrics.append(city_renewal_metrics)Ð
            Ð           except Exception as city_error:Ð
self.logger.error(f"Error calculating renewal metrics for {city}:
{city_error}")Ð               continueÐ       Ð       return
pd.DataFrame(renewal_metrics)Ð   async def
calculate_renewal_metrics_workflow(self, city_data, company_data=None):Ð
        """Ð       Main workflow for calculating and registering renewal
metricsÐ       Ð       :param city_data: City emissions dataÐ       :param
company_data: Optional company emissions dataÐ       """Ð       try:Ð
            # Validate contract availabilityÐ           if
'RenewalTheoryContract' not in self.workflow.contracts:Ð               raise
ValueError("RenewalTheoryContract not loaded")Ð           Ð           #
Calculate renewal metricsÐ           renewal_metrics =
self.calculate_renewal_metrics(Ð               pd.DataFrame(city_data), Ð
                pd.DataFrame(company_data) if company_data is not None else
NoneÐ           )Ð           Ð           # Get the contract instanceÐ
            renewal_contract =
self.workflow.contracts['RenewalTheoryContract']Ð           Ð           #
Process each city's renewal metricsÐ           for _, row in
renewal_metrics.iterrows():Ð               try:Ð                   # Prepare
transaction argumentsÐ                   tx_hash = await
renewal_contract.functions.calculateRenewalMetrics(Ð
```

```
row['city'], Đ                        float(row['total_city_emissions']), Đ
                    float(row['company_emissions']),Đ
                    float(row['total_reduction_target'])Đ
                ).transact({Đ                        'from':
self.workflow.w3.eth.accounts[0],Đ                    'gas': 2000000Đ
                })Đ                     Đ                    # Wait for
transaction receiptĐ                    receipt = await
self.workflow.w3.eth.wait_for_transaction_receipt(tx_hash)Đ                    Đ
                # Log the transactionĐ
self.workflow.log_to_file('renewal_metrics_logs.json', row.to_dict(), receipt)Đ
                Đ                    self.logger.info(f"Processed renewal
metrics for {row['city']}")Đ                Đ                except Exception
as record_error:Đ                    self.logger.error(f"Error processing
renewal metrics for {row['city']}: {record_error}")Đ
continueĐ           Đ           return renewal_metricsĐ       Đ       except
Exception as e:Đ           self.logger.error(f"Comprehensive renewal metrics
calculation error: {e}")Đ           raiseĐ   def
generate_renewal_report(self, renewal_metrics: pd.DataFrame) -> Dict[str,
Any]:Đ       """Đ       Generate a comprehensive renewal reportĐ       Đ
        :param renewal_metrics: DataFrame with renewal metricsĐ
        :return: Summary report dictionaryĐ       """Đ       try:Đ
report = {Đ               'total_cities_analyzed': len(renewal_metrics),Đ
                'global_total_emissions':
renewal_metrics['total_city_emissions'].sum(),Đ
'global_total_reduction_target':
renewal_metrics['total_reduction_target'].sum(),Đ
'cities_with_highest_reduction_potential': renewal_metrics.nlargest(Đ
                3, 'total_reduction_target'Đ                )[['city',
'total_reduction_target']].to_dict(orient='records'),Đ
'sector_reduction_breakdown':
self._aggregate_sector_reduction(renewal_metrics)Đ           }Đ           Đ
            self.logger.info("Generated comprehensive renewal report")Đ
            return reportĐ       Đ       except Exception as e:Đ
self.logger.error(f"Error generating renewal report: {e}")Đ           return
{}Đ   def _aggregate_sector_reduction(self, renewal_metrics: pd.DataFrame) ->
Dict[str, float]:Đ       """Đ       Aggregate sector reduction potential
across all citiesĐ       Đ       :param renewal_metrics: DataFrame with
renewal metricsĐ       :return: Dictionary of sector reduction potentialsĐ
        """Đ       sector_reduction = {}Đ       Đ       for _, row in
renewal_metrics.iterrows():Đ           for sector, reduction in
row['sector_reduction_potential'].items():Đ               if sector not in
sector_reduction:Đ                   sector_reduction[sector] = 0Đ
                sector_reduction[sector] += reductionĐ       Đ       return
sector_reductionĐdef create_renewal_module(workflow):Đ   """Đ   Factory method
to create RenewalModule with custom configurationĐ   Đ   :param workflow:
BlockchainWorkflow instanceĐ   :return: Configured RenewalModule instanceĐ
"""Đ   custom_config = {Đ       'max_concurrent_transactions': 3,Đ
'batch_size': 50,Đ       'renewal_metrics': {Đ
'emissions_reduction_target': 0.15,  # 15% reduction targetĐ
'time_window_days': 365  # Annual analysisĐ       }Đ   }Đ   return
RenewalModule(workflow, config=custom_config)Đ Example usageĐasync def main():Đ
    # Import here to avoid circular importsĐ   from blockchainworkflow import
BlockchainWorkflowĐ   Đ   # Create workflowĐ   workflow = BlockchainWorkflow()Đ
    Đ   # Create renewal moduleĐ   renewal_module = RenewalModule(workflow)Đ
    Đ   # Sample city data (replace with actual data loading)Đ
sample_city_data = pd.DataFrame({Đ       'city': ['CityA', 'CityA', 'CityB',
'CityB'],Đ       'date': ['01/01/2023', '02/01/2023', '01/01/2023',
'02/01/2023'],Đ       'sector': ['Energy', 'Transport', 'Energy',
'Transport'],Đ
```

```
                    'value': [10.5, 12.3, 8.7, 9.2]Đ   })Đ   Đ   # Sample
company data (optional)Đ   sample_company_data = pd.DataFrame({Đ
'city': ['CityA', 'CityB'],Đ       'sector': ['Energy', 'Transport'],Đ
'value': [5.0, 4.5]Đ   })Đ   Đ   # Calculate and register renewal metricsĐ
renewal_metrics = await renewal_module.calculate_renewal_metrics_workflow(Đ
        sample_city_data, Đ       sample_company_dataĐ   )Đ   Đ   # Generate
renewal reportĐ   report =
renewal_module.generate_renewal_report(renewal_metrics)Đ   print(report)Đ
__name__ == "__main__":Đ   import asyncioĐ   asyncio.run(main())
```

## python\sequencerpath.py.py

```
/** * Enhanced Sequencer Precomputation Algorithm *  * This improved algorithm enhances the path calculation with: * - Dynamic weight adjustment based on traffic type * - Historical performance tracking * - Predictive congestion modeling * - Multi-objective optimization */   class EnhancedSequencer {  constructor() {    // Initialize system parameters this.nodes = new Map(); // NID -> NodeData mapping this.historicalPerformance = new Map(); // Connection -> performance history    this.trafficPatterns = new Map(); // Traffic patterns to optimize for this.blockchainInterface = null; // Interface to read/write to blockchain  }  /**   * Calculate optimal path from source NID to destination NIAS   * @param {string} sourceNID - Source Node ID   * @param {string} destNIAS - Destination NIAS ID   * @param {object} trafficRequirements - Requirements for this transmission   * @returns {object} - Optimal path sequence and metadata   */  computeOptimalPath(sourceNID, destNIAS, trafficRequirements) {    // Validate nodes exist and are reachable    if (!this.validateEndpoints(sourceNID, destNIAS)) {      throw new Error(`Invalid endpoints: ${sourceNID} -> ${destNIAS}`);    }         // Get weights based on traffic requirements (VoIP, video, general data, etc.)    const weights = this.dynamicWeightSelection(trafficRequirements);        // Get all active nodes from blockchain    const activeNodes = this.getActiveNodesFromBlockchain();        // Build connectivity graph with weighted edges    const graph = this.buildWeightedGraph(activeNodes, weights, trafficRequirements);        // Calculate the optimal path using enhanced algorithm    const paths = this.multiObjectivePathFinding(graph, sourceNID, destNIAS, weights);        // Select primary and backup paths    const { primaryPath, backupPaths } = this.selectPrimaryAndBackupPaths(paths);    // Prepare path sequence for blockchain recording    const pathSequence = this.preparePathSequence(primaryPath, backupPaths);        // Record path to blockchain for validation    this.recordPathToBlockchain(pathSequence);    return pathSequence;  }     /**   * Dynamically select weights based on traffic requirements   * @param {object} trafficRequirements - Type of traffic and QoS needs   * @returns {object} - Weights for different factors   */  dynamicWeightSelection(trafficRequirements) {    const weights = {      latency: 0.5,      bandwidth: 0.2,      security: 0.2,      reliability: 0.1    };        // Adjust weights based on traffic type    switch (trafficRequirements.type) {      case 'VoIP':        weights.latency = 0.7;        weights.bandwidth = 0.1;        weights.security = 0.1;        weights.reliability = 0.1;        break;      case 'Video':        weights.latency = 0.3;        weights.bandwidth = 0.5;        weights.security = 0.1;        weights.reliability = 0.1;        break;      case 'CriticalData':        weights.latency = 0.2;        weights.bandwidth = 0.1;        weights.security = 0.5;        weights.reliability = 0.2;        break;      // Add more traffic types as needed    }        return weights;  }  /**   * Build a weighted graph representation of the network   * @param {Array} activeNodes - Array of active NIDs   * @param {object} weights - Weights for different factors   * @param {object} trafficRequirements - Traffic requirements   * @returns {object} - Graph with weighted edges   */  buildWeightedGraph(activeNodes, weights, trafficRequirements) {    const graph = {};        // Initialize graph nodes    activeNodes.forEach(node => {      graph[node.id] = { edges: [] };    });        // Build edges between nodes    activeNodes.forEach(nodeA => {      activeNodes.forEach(nodeB => {        if (nodeA.id !== nodeB.id && this.nodesCanConnect(nodeA, nodeB)) {          // Get real-time metrics for this connection          const metrics = this.getConnectionMetrics(nodeA.id, nodeB.id);                    // Get historical performance data          const history = this.getHistoricalPerformance(nodeA.id, nodeB.id);                    // Predict future congestion          const predictedCongestion = this.predictCongestion(nodeA.id, nodeB.id, history);                    // Calculate edge cost based on multiple factors
```

```
                                                const cost =
this.calculateEdgeCost(metrics, history, predictedCongestion, weights,
trafficRequirements);                     // Add edge to graph
graph[nodeA.id].edges.push({            target: nodeB.id,           cost:
cost,           metrics: metrics,           predictedCongestion:
predictedCongestion          });        }      });    });        return graph;
  }     /**   * Calculate the cost of an edge based on multiple factors   *
@param {object} metrics - Current connection metrics   * @param {object}
history - Historical performance   * @param {number} predictedCongestion -
Predicted congestion level   * @param {object} weights - Weights for
different factors   * @param {object} trafficRequirements - Traffic
requirements   * @returns {number} - Weighted cost of the edge   */
calculateEdgeCost(metrics, history, predictedCongestion, weights,
trafficRequirements) {    // Normalize metrics to 0-1 range    const
normalizedLatency = metrics.latency / 100; // Assuming 100ms is worst
acceptable    const normalizedBandwidth = 1 - (metrics.availableBandwidth /
metrics.maxBandwidth);    const normalizedSecurity = 1 -
metrics.securityScore; // Higher security score is better    const
normalizedReliability = 1 - metrics.reliability; // Higher reliability is
better        // Apply congestion prediction (0-1 range)    const
congestionFactor = predictedCongestion * 0.5; // Weight of prediction
    // Historical performance factor (0-1 range)    const historyFactor =
this.calculateHistoryFactor(history);        // Calculate weighted cost
let cost =       weights.latency * normalizedLatency +      weights.bandwidth
* normalizedBandwidth +      weights.security * normalizedSecurity +
weights.reliability * normalizedReliability;        // Add congestion
prediction and history factors    cost = cost * (1 + congestionFactor) * (1 +
historyFactor);        // Add traffic-specific adjustments    if
(trafficRequirements.type === 'VoIP' && normalizedLatency > 0.5) {      //
Penalize high latency routes for VoIP traffic      cost *= 2;    }
return cost;  }     /**   * Multi-objective path finding algorithm   *
Enhanced version of A* that considers multiple optimization objectives   *
@param {object} graph - Network graph   * @param {string} start - Start node
ID   * @param {string} goal - Goal node ID   * @param {object} weights -
Weights for different factors   * @returns {Array} - Multiple candidate paths
   */  multiObjectivePathFinding(graph, start, goal, weights) {    //
Initialize data structures    const openSet = new PriorityQueue();    const
cameFrom = {};    const gScore = {};    const fScore = {};    const
evaluatedPaths = [];        // Initialize starting node    gScore[start] = 0;
    fScore[start] = this.heuristic(start, goal, weights);
openSet.enqueue(start, fScore[start]);        while (!openSet.isEmpty()) {
      const current = openSet.dequeue().element;            // If we've
reached the goal, reconstruct and return the path      if (current === goal) {
        const path = this.reconstructPath(cameFrom, current);
evaluatedPaths.push({          path: path,          score: gScore[current]
        });                // If we have enough paths, return them        if
(evaluatedPaths.length >= 3) {          return evaluatedPaths;        }
        // Continue searching for alternative paths        continue;      }
            // Explore neighbors      graph[current].edges.forEach(edge => {
        const neighbor = edge.target;        const tentativeGScore =
gScore[current] + edge.cost;               if (!(neighbor in gScore) ||
tentativeGScore < gScore[neighbor]) {          // This path is better than
any previous one          cameFrom[neighbor] = current;
gScore[neighbor] = tentativeGScore;          fScore[neighbor] =
gScore[neighbor] + this.heuristic(neighbor, goal, weights);
          if (!openSet.contains(neighbor)) {
openSet.enqueue(neighbor, fScore[neighbor]);          }        }      });    }
        // If no path was found, return empty array    return evaluatedPaths;
  }     /**   * Heuristic function for A* algorithm   * @param {string} node
- Current node ID
```

```
                    * @param {string} goal - Goal node ID   * @param {object} weights - Weights for different factors   * @returns {number} - Heuristic value   */  heuristic(node, goal, weights) {    // Get node data    const nodeData = this.nodes.get(node);    const goalData = this.nodes.get(goal);    if (!nodeData || !goalData) {      return 0;    }        // Calculate geographical distance using Haversine formula    const distance = this.haversineDistance(      nodeData.latitude, nodeData.longitude,      goalData.latitude, goalData.longitude    );        // Normalize distance (assuming max distance is 20,000 km)    const normalizedDistance = distance / 20000;        // Return weighted heuristic    return normalizedDistance * weights.latency;  }     /**   * Calculate geographical distance using Haversine formula   * @param {number} lat1 - Latitude of first point   * @param {number} lon1 - Longitude of first point   * @param {number} lat2 - Latitude of second point   * @param {number} lon2 - Longitude of second point   * @returns {number} - Distance in kilometers   */  haversineDistance(lat1, lon1, lat2, lon2) {    const R = 6371; // Earth radius in kilometers    const dLat = this.deg2rad(lat2 - lat1);    const dLon = this.deg2rad(lon2 - lon1);    const a =       Math.sin(dLat/2) * Math.sin(dLat/2) +      Math.cos(this.deg2rad(lat1)) * Math.cos(this.deg2rad(lat2)) *      Math.sin(dLon/2) * Math.sin(dLon/2);    const c = 2 *      Math.atan2(Math.sqrt(a), Math.sqrt(1-a));    return R * c;  }  deg2rad(deg) {    return deg * (Math.PI/180);  }     /**   * Predict congestion based on historical data and current trends   * @param {string} nodeA - First node ID   * @param {string} nodeB - Second node ID   * @param {object} history - Historical performance data   * @returns {number} - Predicted congestion level (0-1)   */  predictCongestion(nodeA, nodeB, history) {    if (!history || !history.congestionSamples ||      history.congestionSamples.length === 0) {      return 0;    }        // Get current time and day of week    const now = new Date();    const hour = now.getHours();    const dayOfWeek = now.getDay();        // Filter historical samples for similar time and day    const relevantSamples = history.congestionSamples.filter(sample => {      const sampleTime = new Date(sample.timestamp);      return Math.abs(sampleTime.getHours() - hour) <= 1 &&             (sampleTime.getDay() === dayOfWeek);    });        if (relevantSamples.length === 0) {      return 0;    }        // Calculate average congestion for similar times    const avgCongestion = relevantSamples.reduce((sum, sample) => sum + sample.congestion, 0) / relevantSamples.length;        // Current trend (increasing/decreasing congestion)    const recentSamples = history.congestionSamples.slice(-5);    const trend = this.calculateCongestionTrend(recentSamples);        // Combine historical average with current trend    return Math.min(1, Math.max(0, avgCongestion + trend));  }     /**   * Calculate congestion trend from recent samples   * @param {Array} samples - Recent congestion samples   * @returns {number} - Trend value (-0.2 to 0.2)   */  calculateCongestionTrend(samples) {    if (samples.length < 2) {      return 0;    }        // Calculate linear regression slope    let sumX = 0, sumY = 0, sumXY = 0, sumXX = 0;    const n = samples.length;        for (let i = 0; i < n; i++) {      sumX += i;      sumY += samples[i].congestion;      sumXY += i * samples[i].congestion;      sumXX += i * i;    }        const slope = (n * sumXY - sumX * sumY) / (n * sumXX - sumX * sumX);        // Normalize slope to -0.2 to 0.2 range    return Math.max(-0.2, Math.min(0.2, slope));  }     /**   * Select primary and backup paths from candidate paths   * @param {Array} candidatePaths - Array of candidate paths   * @returns {object} - Selected primary and backup paths   */  selectPrimaryAndBackupPaths(candidatePaths) {    if (candidatePaths.length === 0) {      throw new Error("No valid paths found");    }        // Sort paths by score (lower is better)    candidatePaths.sort((a, b) => a.score - b.score);        // Select primary path (best score)    const primaryPath = candidatePaths[0].path;        // Select backup paths (next best, prioritizing path diversity)
```

```
                                const backupPaths = [];        for (let i =
1; i < candidatePaths.length && backupPaths.length < 2; i++) {      const
path = candidatePaths[i].path;            // Check path diversity (how
different is this path from the primary)      const diversity =
this.calculatePathDiversity(primaryPath, path);            // Only select
paths with sufficient diversity      if (diversity > 0.6) {
backupPaths.push(path);      }    }        return { primaryPath,
backupPaths };  }     /**   * Calculate diversity between two paths (0-1,
higher means more diverse)   * @param {Array} path1 - First path   * @param
{Array} path2 - Second path   * @returns {number} - Path diversity score   */
  calculatePathDiversity(path1, path2) {    // Convert paths to sets for easy
comparison    const set1 = new Set(path1);    const set2 = new Set(path2);
    // Count shared nodes    let sharedNodes = 0;    set1.forEach(node => {
      if (set2.has(node)) {        sharedNodes++;      }    });        //
Calculate diversity    const totalUniqueNodes = set1.size + set2.size -
sharedNodes;    return 1 - (sharedNodes / totalUniqueNodes);  }     /**   *
Prepare path sequence for blockchain recording   * @param {Array} primaryPath
- Primary path   * @param {Array} backupPaths - Backup paths   * @returns
{object} - Path sequence object   */  preparePathSequence(primaryPath,
backupPaths) {    const pathId = "SEQ_" + this.generateUniqueId();
return {      path_id: pathId,      source_nid: primaryPath[0],
destination_nias: primaryPath[primaryPath.length - 1],      path_sequence:
primaryPath,      backup_paths: backupPaths,      timestamp: new
Date().toISOString(),      metrics: this.calculatePathMetrics(primaryPath)
    };  }     /**   * Calculate aggregate metrics for entire path   * @param
{Array} path - Path to calculate metrics for   * @returns {object} - Path
metrics   */  calculatePathMetrics(path) {    let totalLatency = 0;    let
minBandwidth = Infinity;    let securityScore = 1;        // Calculate
metrics for each segment and aggregate    for (let i = 0; i < path.length -
1; i++) {      const metrics = this.getConnectionMetrics(path[i], path[i +
1]);            totalLatency += metrics.latency;      minBandwidth =
Math.min(minBandwidth, metrics.availableBandwidth);      securityScore *=
metrics.securityScore; // Multiply security scores (weakest link principle)
    }        return {      totalLatency,      minBandwidth,
securityScore,      hopCount: path.length - 1    };  }     /**   * Record
path to blockchain for validation   * @param {object} pathSequence - Path
sequence object   */  recordPathToBlockchain(pathSequence) {    // Format for
blockchain storage    const blockchainRecord = {      type: "PATH_SEQUENCE",
      data: pathSequence,      hash: this.hashPathSequence(pathSequence),
      timestamp: pathSequence.timestamp    };        // Write to blockchain
(implementation depends on blockchain interface)
this.blockchainInterface.writeRecord(blockchainRecord);  }     /**   * Hash
path sequence for blockchain integrity   * @param {object} pathSequence -
Path sequence object   * @returns {string} - Hash of path sequence   */
hashPathSequence(pathSequence) {    // Simple hash function for example
purposes    // In production, use a cryptographic hash function    return
"0x" + this.simpleHash(JSON.stringify(pathSequence));  }     /**   * Simple
hash function (for demonstration only)   * @param {string} str - String to
hash   * @returns {string} - Hashed string   */  simpleHash(str) {    let
hash = 0;    for (let i = 0; i < str.length; i++) {      const char =
str.charCodeAt(i);      hash = ((hash << 5) - hash) + char;      hash = hash
& hash; // Convert to 32bit integer    }    return
Math.abs(hash).toString(16);  }     /**   * Generate unique ID for path
sequence   * @returns {string} - Unique ID   */  generateUniqueId() {
return Date.now().toString(36) + Math.random().toString(36).substr(2, 5);  }
  /**   * Helper method implementations would go here   * These are
referenced above but not fully implemented for brevity   */
validateEndpoints(sourceNID, destNIAS) {    // Implementation would check if
endpoints exist and are reachable    return true;  }
getActiveNodesFromBlockchain() {
```

```
                                    // Implementation would query blockchain for active nodes    return [];  }    nodesCanConnect(nodeA, nodeB) {    // Implementation would check if nodes can connect    return true;  }    getConnectionMetrics(nodeA, nodeB) {    // Implementation would get real-time metrics for connection    return {      latency: 10,      availableBandwidth: 100,      maxBandwidth: 1000,      securityScore: 0.9,      reliability: 0.95    };  }    getHistoricalPerformance(nodeA, nodeB) {    // Implementation would get historical performance data    return {      congestionSamples: []    };  }    calculateHistoryFactor(history) {    // Implementation would calculate factor based on historical performance    return 0;  }    reconstructPath(cameFrom, current) {    // Implementation would reconstruct path from cameFrom map    return [];  }} /** * Priority Queue implementation for A* algorithm */class PriorityQueue {  constructor() {    this.elements = [];  }    enqueue(element, priority) {    this.elements.push({ element, priority });    this.elements.sort((a, b) => a.priority - b.priority);  }    dequeue() {    return this.elements.shift();  }    isEmpty() {    return this.elements.length === 0;  }    contains(element) {    return this.elements.some(item => item.element === element);  }} // Example usage:/* const sequencer = new EnhancedSequencer();const path = sequencer.computeOptimalPath('NID-1', 'NIAS-12', { type: 'VoIP' }); console.log(path);*/
```

## python\test_n2n_routing.py

```
import osimport jsonimport loggingfrom typing import Dict, Any, Optionalfrom
web3 import Web3                class ContractConfig:    def __init__(self,
address: str, abi: Optional[Dict] = None):        """        Contract
configuration class                :param address: Contract address
        :param abi: Contract ABI (optional)        """        self.address =
address        self.abi = abi                        class N2NContractLoader:
    def __init__(        self,         rpc_url: str =
"http://127.0.0.1:8545",         contract_addresses_path: Optional[str] =
None,        build_contracts_dir: Optional[str] = None    ):        """
        Initialize N2N Contract Loader                :param rpc_url:
Blockchain RPC URL        :param contract_addresses_path: Path to contract
addresses JSON        :param build_contracts_dir: Directory containing
contract build artifacts        """        # Setup logging
logging.basicConfig(            level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s'        )
self.logger = logging.getLogger(__name__)                # Determine project
root and default paths        self.project_root = os.path.abspath(
os.path.join(os.path.dirname(__file__), '..', '..')        )                #
Contract addresses file        self.contract_addresses_path =
contract_addresses_path or os.path.join(            self.project_root,
'config', 'contract_addresses.json'        )                # Build contracts
directory        self.build_contracts_dir = build_contracts_dir or
os.path.join(            self.project_root, 'build', 'contracts'        )
                # Blockchain connection        self.rpc_url = rpc_url
self.w3 = Web3(Web3.HTTPProvider(self.rpc_url))                # Validate
blockchain connection        if not self.w3.is_connected():            raise
ConnectionError(f"Could not connect to blockchain at {self.rpc_url}")
        # Load contract addresses and configurations
self.contract_addresses = self._load_contract_addresses()                #
N2N-specific contracts to load        self.n2n_contract_names = [
'NIDRegistry',             'NIASRegistry',             'ABATLTranslation',
            'SequencePathRouter',             'ClusteringContract'        ]
                # Contracts storage        self.contracts: Dict[str, Any] = {}
        def _load_contract_addresses(self) -> Dict[str, ContractConfig]:
        """        Load contract addresses from JSON file
        :return: Dictionary of contract configurations        """        try:
            with open(self.contract_addresses_path, 'r') as f:
raw_addresses = json.load(f)                        # Convert raw addresses
to ContractConfig objects            contract_configs = {}            for
name, address in raw_addresses.items():                contract_configs[name]
= ContractConfig(address)                        self.logger.info(f"Loaded
contract addresses from {self.contract_addresses_path}")            return
contract_configs        except FileNotFoundError:
self.logger.error(f"Contract addresses file not found at
{self.contract_addresses_path}")            raise        except
json.JSONDecodeError:            self.logger.error(f"Invalid JSON in contract
addresses file at {self.contract_addresses_path}")            raise
def _load_contract_abi(self, contract_name: str) -> Dict[str, Any]:        """
        Load contract ABI from build artifacts                :param
contract_name: Name of the contract        :return: Contract ABI        """
        # Potential ABI file paths        abi_paths = [
os.path.join(self.build_contracts_dir, f"{contract_name}.json"),
os.path.join(self.build_contracts_dir, 'N2N', f"{contract_name}.json"),
            os.path.join(self.build_contracts_dir, 'blockchain',
f"{contract_name}.json"),            os.path.join(self.build_contracts_dir,
'passchain', f"{contract_name}.json"),
os.path.join(self.build_contracts_dir, 'relay', f"{contract_name}.json")
        ]                for abi_path in abi_paths:            try:
                if os.path.exists(abi_path):
```

```
                                                    with
open(abi_path, 'r') as f:                           contract_data = json.load(f)
                    return contract_data['abi']           except
Exception as e:                 self.logger.warning(f"Error reading ABI at
{abi_path}: {e}")                 raise FileNotFoundError(f"No ABI found for
contract: {contract_name}")       def load_n2n_contracts(self) -> Dict[str,
Any]:        """        Load N2N-related contracts                :return:
Dictionary of loaded contracts        """        for contract_name in
self.n2n_contract_names:            try:                # Get contract
configuration                contract_config =
self.contract_addresses.get(contract_name)                                  if
not contract_config or not contract_config.address or contract_config.address
== '0x0':                    raise ValueError(f"Invalid address for contract
{contract_name}")                                   # Load ABI
abi = self._load_contract_abi(contract_name)                                 #
Update contract configuration with ABI                contract_config.abi =
abi                                 # Create contract instance
contract = self.w3.eth.contract(
address=contract_config.address,                    abi=contract_config.abi
                )                                   # Store contract
                self.contracts[contract_name] = contract
                self.logger.info(f"Successfully loaded {contract_name} at
{contract_config.address}")                           except (KeyError,
ValueError, FileNotFoundError) as e:
self.logger.error(f"Failed to load contract {contract_name}: {e}")
                raise                return self.contracts
# Utility functionsdef bytes32_to_hex(bytes32_val: bytes) -> str:
"""Convert bytes32 to hex string"""    return '0x' + bytes32_val.hex()
def hex_to_bytes32(hex_str: str) -> bytes:    """Convert hex string to
bytes32"""    if hex_str.startswith('0x'):        hex_str = hex_str[2:]
return bytes.fromhex(hex_str.zfill(64))
def hash_to_bytes32(text: str) -> bytes:    """Create a bytes32 hash from a
string"""    from web3 import Web3    return Web3.keccak(text=text)[:32]
def generate_random_hash() -> bytes:    """Generate a random bytes32 hash"""
    return hash_to_bytes32(str(random.random()))
def eth_address_to_node_id(address: str) -> bytes:    """Convert Ethereum
address to a node ID"""    return hash_to_bytes32(address)
def node_id_to_eth_address(node_id: bytes) -> str:    """Deterministically
convert node ID to Ethereum address"""    # This is just for demonstration,
in practice you would have a more sophisticated mapping    w3 =
Web3(Web3.HTTPProvider("http://127.0.0.1:8545"))    return
w3.eth.accounts[int.from_bytes(node_id[:4], 'big') % len(w3.eth.accounts)]
# Test parametersNUM_NODES = 100NUM_CLUSTERS = 5NUM_PATHS = 50
TRANSMISSION_SIZE = 1000  # packetsPACKET_SIZE = 1024  # bytesLATENCY_BASE =
10  # msMAX_HOPS = 5            # Test setupdef setup_test_environment() ->
Dict[str, List]:    """Set up test environment with nodes, clusters and
paths"""    print("Setting up test environment...")        # Track registered
entities    registered = {        'nids': [],        'nias': [],
'clusters': [],        'abatl_records': [],        'paths': [],
'path_statuses': []  # Added to track path statuses    }    # Generate
accounts for transactions    accounts = w3.eth.accounts[:10]  # Use first 10
accounts        # Create clusters first    print(f"Creating {NUM_CLUSTERS}
clusters...")    for i in range(NUM_CLUSTERS):        cluster_id = i + 1  #
Cluster IDs start from 1        cluster_type = 0 if i < NUM_CLUSTERS // 2
else 1  # Half NAP, half BGP        valid_until = int(time.time()) + 3600 *
24 * 30  # Valid for 30 days        security_level = random.randint(1, 5)
        max_latency = random.randint(50, 200)        min_bandwidth =
random.randint(10, 100)                tx_hash =
clustering_contract.functions.createCluster(            cluster_id,
            f"Cluster-{cluster_id}",
```

```
                                        cluster_type,
valid_until,            security_level,            max_latency,
min_bandwidth        ).transact({'from': accounts[0]})
w3.eth.wait_for_transaction_receipt(tx_hash)
registered['clusters'].append(cluster_id)        # Create NIDs
print(f"Creating {NUM_NODES // 2} NIDs...")    for i in range(NUM_NODES // 2):
        # Primary ID attributes        primary_id = hash_to_bytes32(f"nid-
primary-{i}")        secondary_id = hash_to_bytes32(f"nid-secondary-{i}")
        security_level = random.randint(1, 5)        cluster_id =
random.choice(registered['clusters'][:NUM_CLUSTERS//2])  # NAP clusters
        node_type = random.choice(["VALIDATOR", "RELAY", "EDGE"])
        tx_hash = nid_registry.functions.registerNode(            primary_id,
            secondary_id,            security_level,            cluster_id,
            node_type        ).transact({'from': accounts[0]})
w3.eth.wait_for_transaction_receipt(tx_hash)
registered['nids'].append(bytes32_to_hex(primary_id))        # Create NIAS
print(f"Creating {NUM_NODES // 2} NIAS...")    for i in range(NUM_NODES // 2):
        # Primary ID attributes        primary_id = hash_to_bytes32(f"nias-
primary-{i}")        secondary_id = hash_to_bytes32(f"nias-secondary-{i}")
        security_level = random.randint(1, 5)        routing_weight =
random.randint(1, 100)        load_balancing_factor = random.randint(1, 100)
        cluster_id = random.choice(registered['clusters'][NUM_CLUSTERS//2:])
# BGP clusters        nias_type = random.choice(["EDGE", "RELAY",
"VALIDATOR"])                tx_hash = nias_registry.functions.registerNIAS(
            primary_id,            secondary_id,            security_level,
            routing_weight,            load_balancing_factor,
cluster_id,            nias_type        ).transact({'from': accounts[0]})
                w3.eth.wait_for_transaction_receipt(tx_hash)
registered['nias'].append(bytes32_to_hex(primary_id))        # Create ABATL
records    print(f"Creating ABATL records...")    for i in range(NUM_NODES //
4):  # Create fewer ABATL records        abatl_id = hash_to_bytes32(f"abatl-
{i}")        nid_id = hex_to_bytes32(random.choice(registered['nids']))
        nias_id = hex_to_bytes32(random.choice(registered['nias']))
cluster_id = random.randint(1, NUM_CLUSTERS)        abatl_type =
random.randint(0, 2)        sender_type = random.choice([0, 1])  # 0 =
NID_SENDER, 1 = NIAS_SENDER                tx_hash =
abatl_translation.functions.registerABATL(            abatl_id,
nid_id,            nias_id,            cluster_id,            abatl_type,
            sender_type        ).transact({'from': accounts[0]})
        w3.eth.wait_for_transaction_receipt(tx_hash)
registered['abatl_records'].append(bytes32_to_hex(abatl_id))                #
Update secondary attributes        qos_level = random.randint(1, 100)
latency = random.randint(10, 200)        bandwidth = random.randint(10, 1000)
        security_level = random.randint(1, 5)                tx_hash =
abatl_translation.functions.updateABATLSecondaryAttributes(
abatl_id,            qos_level,            latency,            bandwidth,
            security_level        ).transact({'from': accounts[0]})
        w3.eth.wait_for_transaction_receipt(tx_hash)       # Create paths
print(f"Creating {NUM_PATHS} paths...")    for i in range(NUM_PATHS):
path_id = hash_to_bytes32(f"path-{i}")        source_nid =
hex_to_bytes32(random.choice(registered['nids']))        destination_nias =
hex_to_bytes32(random.choice(registered['nias']))                # Create
random path sequence        num_hops = random.randint(0, MAX_HOPS)
path_sequence = [source_nid]                # Add intermediate NIDs
for _ in range(num_hops):            intermediate_nid =
hex_to_bytes32(random.choice(registered['nids']))
path_sequence.append(intermediate_nid)                # Add destination NIAS
        path_sequence.append(destination_nias)                # Service class
        service_class = random.choice(["VoIP", "Streaming", "Standard",
"Critical"])
```

```
                            # Create path with all required parameters
tx_hash = sequence_path_router.functions.createPath(            path_id,
            source_nid,           destination_nias,           path_sequence,
            service_class        ).transact({'from': accounts[0]})
        w3.eth.wait_for_transaction_receipt(tx_hash)
registered['paths'].append(bytes32_to_hex(path_id))                # Store
initial path status        path_status = {           'path_id':
bytes32_to_hex(path_id),           'status': 0,  # Pending
'packets_total': 0,           'packets_lost': 0,           'latency': 0,
            'compliance': False        }
registered['path_statuses'].append(path_status)                # Create
disjoint path for some paths        if random.random() < 0.3:  # 30% chance
            # Create disjoint path sequence           disjoint_sequence =
[source_nid]                        # Add different intermediate NIDs
            for _ in range(num_hops):                # Ensure we don't reuse
intermediate nodes from original path                while True:
                    intermediate_nid =
hex_to_bytes32(random.choice(registered['nids']))                    if
intermediate_nid not in path_sequence[1:-1]:                        break
                disjoint_sequence.append(intermediate_nid)
            # Add destination NIAS
disjoint_sequence.append(destination_nias)                        tx_hash =
sequence_path_router.functions.createDisjointPath(                path_id,
                disjoint_sequence            ).transact({'from': accounts[0]})
                        w3.eth.wait_for_transaction_receipt(tx_hash)
print("Test environment setup complete")    return registered# Test functions
def simulate_transmissions(registered_entities: Dict[str, List]) -> Dict[str,
List]:    """Simulate data transmissions and measure performance"""
print("Simulating transmissions...")        results = {        'path_id': [],
        'path_length': [],        'transmission_time': [],
'packets_lost': [],        'latency': [],        'throughput': [],
'success_rate': [],        'path_status': [],  # Added to track final status
        'compliance_check': []  # Added to track QoS compliance    }
accounts = w3.eth.accounts[:10]        for i, path_id_hex in
enumerate(registered_entities['paths']):        path_id =
hex_to_bytes32(path_id_hex)                # Get path details using the new
getPath function        path_data =
sequence_path_router.functions.getPath(path_id).call()        path_sequence =
path_data[3]  # pathSequence is at index 3 in the PathRecord
path_length = len(path_sequence)        service_class = path_data[9]  #
serviceClass is at index 9                # Simulate transmission
characteristics        security_level = random.randint(1, 5)
packets_total = TRANSMISSION_SIZE                # Start transmission with
new parameters        start_time = time.time()        tx_hash =
sequence_path_router.functions.startTransmission(            path_id,
            packets_total,            security_level
        ).transact({'from': accounts[0]})
w3.eth.wait_for_transaction_receipt(tx_hash)                # Simulate
transmission metrics        simulated_latency = LATENCY_BASE * (path_length -
1) + random.randint(-5, 10)        packet_loss_rate = 0.01 * (path_length - 1)
        packets_lost = int(packets_total * packet_loss_rate)
transmission_time = simulated_latency / 1000        throughput =
(packets_total - packets_lost) / transmission_time        success_rate =
(packets_total - packets_lost) / packets_total * 100        compliance_check
= success_rate > 95                # Complete transmission with new parameters
        tx_hash = sequence_path_router.functions.completeTransmission(
            path_id,           packets_lost,           simulated_latency,
            compliance_check        ).transact({'from': accounts[0]})
        w3.eth.wait_for_transaction_receipt(tx_hash)        end_time =
time.time()
```

```
                        # Get updated path status        status_data =
sequence_path_router.functions.pathStatus(path_id).call()        final_status
= status_data[6]  # complianceCheck is at index 6                # Record
results        results['path_id'].append(path_id_hex)
results['path_length'].append(path_length)
results['transmission_time'].append(end_time - start_time)
results['packets_lost'].append(packets_lost)
results['latency'].append(simulated_latency)
results['throughput'].append(throughput)
results['success_rate'].append(success_rate)
results['path_status'].append(final_status)
results['compliance_check'].append(compliance_check)
print(f"Completed transmission {i+1}/{len(registered_entities['paths'])}",
end="\r")        print("\nAll transmissions completed")    return results
def test_node_failure_recovery(registered_entities: Dict[str, List]) ->
Dict[str, List]:    """Test recovery from node failures"""    print("Testing
node failure recovery...")        results = {        'path_id': [],
'original_path_length': [],        'new_path_length': [],
'rerouting_time': [],        'reroute_successful': [],
'used_disjoint_path': []  # Track if disjoint path was used    }
accounts = w3.eth.accounts[:10]        # Test on a subset of paths
test_paths = random.sample(registered_entities['paths'], min(10,
len(registered_entities['paths'])))        for i, path_id_hex in
enumerate(test_paths):        path_id = hex_to_bytes32(path_id_hex)
        # Get original path sequence using new getPath function
path_data = sequence_path_router.functions.getPath(path_id).call()
original_sequence = path_data[3]  # pathSequence is at index 3
original_path_length = len(original_sequence)                # Skip if path
has only source and destination        if original_path_length <= 2:
            continue                # Choose a random intermediate node to
fail        failed_node_index = random.randint(1, original_path_length - 2)
        failed_node = original_sequence[failed_node_index]                #
Check for available disjoint paths        disjoint_path_count =
sequence_path_router.functions.getDisjointPathsCount(path_id).call()
used_disjoint = False                if disjoint_path_count > 0:            #
Use the first disjoint path            used_disjoint = True
disjoint_path = sequence_path_router.functions.disjointPaths(path_id,
0).call()            new_sequence = disjoint_path[1]  # pathSequence is at
index 1 in DisjointPath        else:            # Create new path by
replacing the failed node            new_sequence = list(original_sequence)
            while True:                replacement_node_hex =
random.choice(registered_entities['nids'])                replacement_node =
hex_to_bytes32(replacement_node_hex)                if replacement_node not
in original_sequence:                    new_sequence[failed_node_index] =
replacement_node                    break                # Measure rerouting
time        start_time = time.time()        tx_hash =
sequence_path_router.functions.reroutePath(            path_id,
failed_node        ).transact({'from': accounts[0]})                receipt =
w3.eth.wait_for_transaction_receipt(tx_hash)        end_time = time.time()
                # Verify the reroute        updated_path =
sequence_path_router.functions.getPath(path_id).call()
updated_sequence = updated_path[3]        reroute_successful = failed_node
not in updated_sequence                # Record results
results['path_id'].append(path_id_hex)
results['original_path_length'].append(original_path_length)
results['new_path_length'].append(len(updated_sequence))
results['rerouting_time'].append(end_time - start_time)
results['reroute_successful'].append(reroute_successful)
results['used_disjoint_path'].append(used_disjoint)
print(f"Completed node failure test {i+1}/{len(test_paths)}", end="\r")
```

```
    print("\nAll node failure tests completed")    return results
def test_clustering_efficiency(registered_entities: Dict[str, List]) ->
Dict[str, List]:    """Test the efficiency of node clustering"""
print("Testing clustering efficiency...")        results = {
'cluster_id': [],        'node_count': [],        'avg_latency': [],
'avg_bandwidth': [],        'successful_transmissions': [],
'failed_transmissions': []    }        accounts = w3.eth.accounts[:10]
# Test each cluster    for cluster_id in registered_entities['clusters']:
        # Get cluster members        cluster_members =
clustering_contract.functions.getClusterMembers(cluster_id).call()
node_count = len(cluster_members)                if node_count == 0:
            continue                # Calculate metrics        avg_latency =
random.randint(10, 100)  # Simulated average latency        avg_bandwidth =
random.randint(10, 1000)  # Simulated average bandwidth
avg_security_level = random.randint(1, 5)  # Simulated average security level
        successful_transmissions = random.randint(10, 100)  # Simulated
successful transmissions        failed_transmissions = random.randint(0, 10)
# Simulated failed transmissions                # Update cluster metrics
        tx_hash = clustering_contract.functions.updateClusterMetrics(
            cluster_id,            avg_latency,            avg_bandwidth,
            avg_security_level,            successful_transmissions,
            failed_transmissions        ).transact({'from': accounts[0]})
                w3.eth.wait_for_transaction_receipt(tx_hash)                #
Record results        results['cluster_id'].append(cluster_id)
results['node_count'].append(node_count)
results['avg_latency'].append(avg_latency)
results['avg_bandwidth'].append(avg_bandwidth)
results['successful_transmissions'].append(successful_transmissions)
results['failed_transmissions'].append(failed_transmissions)
print("Clustering efficiency tests completed")    return results
def compare_with_traditional_bgp() -> Dict[str, List]:    """Simulate
comparison with traditional BGP approach (for demonstration)"""
print("Comparing with traditional BGP approach...")        # Simulated data
for comparison    # In a real scenario, this would come from actual BGP
measurements    results = {        'hop_count': list(range(1, 11)),  # 1 to
10 hops        'n2n_latency': [],  # Our approach        'bgp_latency': [],
# Traditional approach        'n2n_throughput': [],  # Our approach
'bgp_throughput': [],  # Traditional approach        'n2n_recovery_time':
[],  # Our approach        'bgp_recovery_time': []  # Traditional approach
    }        # Simulate latency data for both approaches    for hops in
results['hop_count']:        # N2N latency (lower due to precomputed paths)
        n2n_latency = LATENCY_BASE * hops + random.randint(-2, 5)
results['n2n_latency'].append(n2n_latency)                # BGP latency
(higher due to dynamic routing decisions)        bgp_latency = LATENCY_BASE *
hops * 1.5 + random.randint(0, 20)
results['bgp_latency'].append(bgp_latency)                # N2N throughput
(packets per second)        n2n_throughput = 1000 / (n2n_latency / 1000)  #
Convert ms to seconds        results['n2n_throughput'].append(n2n_throughput)
                # BGP throughput (packets per second)        bgp_throughput =
1000 / (bgp_latency / 1000)  # Convert ms to seconds
results['bgp_throughput'].append(bgp_throughput)                # Recovery
time from failure (ms)        # N2N can recover faster due to precomputed
disjoint paths        n2n_recovery = 50 + hops * 10 + random.randint(-5, 15)
        results['n2n_recovery_time'].append(n2n_recovery)                #
BGP recovery time (longer due to global reconvergence)        bgp_recovery =
200 + hops * 50 + random.randint(0, 100)
results['bgp_recovery_time'].append(bgp_recovery)        print("Comparison
completed")    return results                    # Visualization functionsdef
visualize_transmission_results(results: Dict[str, List]):
```

```
                                                    """Visualize
transmission results with new metrics"""    print("Generating transmission
visualization...")        # Create DataFrame    df = pd.DataFrame(results)
    # Plot latency vs path length with compliance status
plt.figure(figsize=(12, 6))    colors = df['compliance_check'].map({True:
'green', False: 'red'})    plt.scatter(df['path_length'], df['latency'],
c=colors)    plt.xlabel('Path Length (Hops)')    plt.ylabel('Latency (ms)')
    plt.title('Latency vs Path Length (Green = Compliant, Red = Non-
compliant)')    plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()    plt.savefig(f"{RESULT_OUTPUT_DIR}/
latency_vs_path_length.png")        # Plot throughput vs success rate
plt.figure(figsize=(12, 6))    plt.scatter(df['throughput'],
df['success_rate'])    plt.xlabel('Throughput (packets/second)')
plt.ylabel('Success Rate (%)')    plt.title('Throughput vs Success Rate')
plt.grid(True, linestyle='--', alpha=0.7)    plt.tight_layout()
plt.savefig(f"{RESULT_OUTPUT_DIR}/throughput_vs_success_rate.png")        #
Plot compliance status distribution    plt.figure(figsize=(10, 6))
df['compliance_check'].value_counts().plot(kind='bar')    plt.xlabel('QoS
Compliance')    plt.ylabel('Count')    plt.title('Distribution of QoS
Compliance Status')    plt.xticks([0, 1], ['Non-compliant', 'Compliant'],
rotation=0)    plt.grid(True, linestyle='--', alpha=0.7)    plt.tight_layout()
    plt.savefig(f"{RESULT_OUTPUT_DIR}/compliance_distribution.png")
print("Transmission visualization complete")
def generate_summary_report(    transmission_results: Dict[str, List],
node_failure_results: Dict[str, List],     clustering_results: Dict[str,
List],    comparison_results: Dict[str, List]):    """Generate summary report
with new metrics"""    print("Generating summary report...")        # Create
DataFrames    transmission_df = pd.DataFrame(transmission_results)
node_failure_df = pd.DataFrame(node_failure_results)    clustering_df =
pd.DataFrame(clustering_results)    comparison_df =
pd.DataFrame(comparison_results)        # Calculate key metrics
compliance_rate = transmission_df['compliance_check'].mean() * 100
disjoint_path_usage = node_failure_df['used_disjoint_path'].mean() * 100 if
len(node_failure_df) > 0 else 0        # Generate report text    report = f"""
# N2N Routing System Performance Report
## 1. Transmission Performance                                    - **Average
Latency**: {transmission_df['latency'].mean():.2f} ms- **Average
Throughput**: {transmission_df['throughput'].mean():.2f} packets/second-
**Average Success Rate**: {transmission_df['success_rate'].mean():.2f}%-
**QoS Compliance Rate**: {compliance_rate:.2f}%
## 2. Node Failure Recovery                                - **Average Rerouting
Time**: {node_failure_df['rerouting_time'].mean():.4f} seconds- **Rerouting
Success Rate**: {node_failure_df['reroute_successful'].mean() * 100:.2f}%-
**Disjoint Path Usage**: {disjoint_path_usage:.2f}%
## 3. Clustering Efficiency                                - **Number of
Clusters**: {len(clustering_df)}- **Average Nodes per Cluster**:
{clustering_df['node_count'].mean():.2f}
## 4. Comparison with Traditional BGP                                          -
**Latency Improvement**: {((comparison_df['bgp_latency'] -
comparison_df['n2n_latency']) / comparison_df['bgp_latency']).mean() *
100:.2f}%- **Throughput Improvement**: {((comparison_df['n2n_throughput'] -
comparison_df['bgp_throughput']) / comparison_df['bgp_throughput']).mean() *
100:.2f}%- **Recovery Time Improvement**:
{((comparison_df['bgp_recovery_time'] - comparison_df['n2n_recovery_time']) /
comparison_df['bgp_recovery_time']).mean() * 100:.2f}%
## 5. Conclusion                The enhanced N2N routing system with sequence
path management demonstrates:- Improved QoS compliance tracking- Efficient
disjoint path utilization- Comprehensive path status monitoring- Better
failure recovery mechanisms"""        # Save report to file    with
open(f"{RESULT_OUTPUT_DIR}/performance_report.md", 'w') as f:
```

```
f.write(report)        print("Summary report generated")    return report
def visualize_node_failure_recovery(results: Dict[str, List]):
"""Visualize node failure recovery results"""    print("Generating node
failure recovery visualization...")        # Create DataFrame    df =
pd.DataFrame(results)        # Filter successful reroutes    df_successful =
df[df['reroute_successful']]        # Plot rerouting time vs original path
length    plt.figure(figsize=(10, 6))
plt.scatter(df_successful['original_path_length'],
df_successful['rerouting_time'])    plt.xlabel('Original Path Length (Hops)')
    plt.ylabel('Rerouting Time (seconds)')    plt.title('Rerouting Time vs
Path Length')    plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()    plt.savefig(f"{RESULT_OUTPUT_DIR}/
rerouting_time_vs_path_length.png")        # Plot original vs new path length
    plt.figure(figsize=(10, 6))
plt.scatter(df_successful['original_path_length'],
df_successful['new_path_length'])    plt.plot([1,
max(df_successful['original_path_length'])], [1,
max(df_successful['original_path_length'])], 'r--')  # Diagonal line
plt.xlabel('Original Path Length (Hops)')    plt.ylabel('New Path Length
(Hops)')    plt.title('Original vs New Path Length After Rerouting')
plt.grid(True, linestyle='--', alpha=0.7)    plt.tight_layout()
plt.savefig(f"{RESULT_OUTPUT_DIR}/original_vs_new_path_length.png")
print("Node failure recovery visualization complete")
def visualize_clustering_efficiency(results: Dict[str, List]):
"""Visualize clustering efficiency results"""    print("Generating clustering
efficiency visualization...")        # Create DataFrame    df =
pd.DataFrame(results)        # Plot node count vs average latency
plt.figure(figsize=(10, 6))    plt.scatter(df['node_count'],
df['avg_latency'])    plt.xlabel('Node Count')    plt.ylabel('Average Latency
(ms)')    plt.title('Average Latency vs Node Count per Cluster')
plt.grid(True, linestyle='--', alpha=0.7)    plt.tight_layout()
plt.savefig(f"{RESULT_OUTPUT_DIR}/avg_latency_vs_node_count.png")       #
Plot node count vs average bandwidth    plt.figure(figsize=(10, 6))
plt.scatter(df['node_count'], df['avg_bandwidth'])    plt.xlabel('Node Count')
    plt.ylabel('Average Bandwidth (Mbps)')    plt.title('Average Bandwidth vs
Node Count per Cluster')    plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()    plt.savefig(f"{RESULT_OUTPUT_DIR}/
avg_bandwidth_vs_node_count.png")        # Calculate success rate
df['success_rate'] = df['successful_transmissions'] /
(df['successful_transmissions'] + df['failed_transmissions']) * 100       #
Plot node count vs success rate    plt.figure(figsize=(10, 6))
plt.scatter(df['node_count'], df['success_rate'])    plt.xlabel('Node Count')
    plt.ylabel('Success Rate (%)')    plt.title('Success Rate vs Node Count
per Cluster')    plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()    plt.savefig(f"{RESULT_OUTPUT_DIR}/
success_rate_vs_node_count.png")        print("Clustering efficiency
visualization complete")                        def
visualize_comparison_with_bgp(results: Dict[str, List]):    """Visualize
comparison with traditional BGP approach"""    print("Generating comparison
visualization...")        # Create DataFrame    df = pd.DataFrame(results)
    # Plot latency comparison    plt.figure(figsize=(10, 6))
plt.plot(df['hop_count'], df['n2n_latency'], 'o-', label='N2N Approach')
plt.plot(df['hop_count'], df['bgp_latency'], 's-', label='Traditional BGP')
    plt.xlabel('Hop Count')    plt.ylabel('Latency (ms)')
plt.title('Latency Comparison: N2N vs Traditional BGP')    plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)    plt.tight_layout()
plt.savefig(f"{RESULT_OUTPUT_DIR}/latency_comparison.png")        # Plot
throughput comparison    plt.figure(figsize=(10, 6))
plt.plot(df['hop_count'], df['n2n_throughput'], 'o-', label='N2N Approach')
```

```
    plt.plot(df['hop_count'], df['bgp_throughput'], 's-', label='Traditional
BGP')    plt.xlabel('Hop Count')    plt.ylabel('Throughput (packets/second)')
    plt.title('Throughput Comparison: N2N vs Traditional BGP')    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.7)    plt.tight_layout()
plt.savefig(f"{RESULT_OUTPUT_DIR}/throughput_comparison.png")        # Plot
recovery time comparison    plt.figure(figsize=(10, 6))
plt.plot(df['hop_count'], df['n2n_recovery_time'], 'o-', label='N2N Approach')
    plt.plot(df['hop_count'], df['bgp_recovery_time'], 's-',
label='Traditional BGP')    plt.xlabel('Hop Count')    plt.ylabel('Recovery
Time (ms)')    plt.title('Recovery Time Comparison: N2N vs Traditional BGP')
    plt.legend()    plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()    plt.savefig(f"{RESULT_OUTPUT_DIR}/
recovery_time_comparison.png")        # Calculate improvement percentages
latency_improvement = ((df['bgp_latency'] - df['n2n_latency']) /
df['bgp_latency']) * 100    throughput_improvement = ((df['n2n_throughput'] -
df['bgp_throughput']) / df['bgp_throughput']) * 100    recovery_improvement =
((df['bgp_recovery_time'] - df['n2n_recovery_time']) /
df['bgp_recovery_time']) * 100        # Plot improvement percentages
plt.figure(figsize=(10, 6))    plt.plot(df['hop_count'], latency_improvement,
'o-', label='Latency Improvement')    plt.plot(df['hop_count'],
throughput_improvement, 's-', label='Throughput Improvement')
plt.plot(df['hop_count'], recovery_improvement, '^-', label='Recovery Time
Improvement')    plt.xlabel('Hop Count')    plt.ylabel('Improvement (%)')
plt.title('Performance Improvement of N2N over Traditional BGP')
plt.legend()    plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()    plt.savefig(f"{RESULT_OUTPUT_DIR}/
performance_improvement.png")        print("Comparison visualization
complete")          def generate_summary_report(    transmission_results:
Dict[str, List],    node_failure_results: Dict[str, List],
clustering_results: Dict[str, List],    comparison_results: Dict[str, List]):
    """Generate summary report of results"""    print("Generating summary
report...")        # Create DataFrames    transmission_df =
pd.DataFrame(transmission_results)    node_failure_df =
pd.DataFrame(node_failure_results)    clustering_df =
pd.DataFrame(clustering_results)    comparison_df =
pd.DataFrame(comparison_results)        # Calculate key metrics
avg_latency = transmission_df['latency'].mean()    avg_throughput =
transmission_df['throughput'].mean()    avg_success_rate =
transmission_df['success_rate'].mean()        avg_rerouting_time =
node_failure_df['rerouting_time'].mean() if len(node_failure_df) > 0 else 0
    rerouting_success_rate = node_failure_df['reroute_successful'].mean() *
100 if len(node_failure_df) > 0 else 0        # Calculate average improvement
over BGP    avg_latency_improvement = ((comparison_df['bgp_latency'] -
comparison_df['n2n_latency']) / comparison_df['bgp_latency']).mean() * 100
avg_throughput_improvement = ((comparison_df['n2n_throughput'] -
comparison_df['bgp_throughput']) / comparison_df['bgp_throughput']).mean() *
100    avg_recovery_improvement = ((comparison_df['bgp_recovery_time'] -
comparison_df['n2n_recovery_time']) /
comparison_df['bgp_recovery_time']).mean() * 100        # Generate report text
    report = f"""# N2N Routing System Performance Report
## 1. Transmission Performance                                    - **Average
Latency**: {avg_latency:.2f} ms- **Average Throughput**: {avg_throughput:.2f}
packets/second- **Average Success Rate**: {avg_success_rate:.2f}%
## 2. Node Failure Recovery                                - **Average Rerouting
Time**: {avg_rerouting_time:.4f} seconds- **Rerouting Success Rate**:
{rerouting_success_rate:.2f}%                                ## 3. Clustering
Efficiency          - **Number of Clusters**: {len(clustering_df)}- **Average
Nodes per Cluster**: {clustering_df['node_count'].mean():.2f}
## 4. Comparison with Traditional BGP
```

-
**Latency Improvement**: {avg_latency_improvement:.2f}%- **Throughput Improvement**: {avg_throughput_improvement:.2f}%- **Recovery Time Improvement**: {avg_recovery_improvement:.2f}%
## 5. Conclusion The Node-to-Node (N2N) multi-layer communication scheme demonstrates significant improvements over traditional BGP in terms of latency, throughput, and failure recovery time. The precomputed path sequencing allows for faster data transmission, while the disjoint path mechanism provides robust failure recovery.
The clustering approach effectively groups nodes based on their attributes, maintaining optimal network performance within each cluster.
Overall, the N2N system provides a more efficient and reliable networking solution compared to traditional BGP."""  # Save report to file  with open(f"{RESULT_OUTPUT_DIR}/performance_report.md", 'w') as f:
f.write(report)  print("Summary report generated")  return report
# Main functiondef main():  try:  # Initialize contract loader
loader = N2NContractLoader()  # Load N2N contracts
n2n_contracts = loader.load_n2n_contracts()  # Access specific contracts  nid_registry = n2n_contracts.get('NIDRegistry')
nias_registry = n2n_contracts.get('NIASRegistry')  abatl_translation = n2n_contracts.get('ABATLTranslation')  sequence_path_router = n2n_contracts.get('SequencePathRouter')  clustering_contract = n2n_contracts.get('ClusteringContract')  # Validate core contract loading  required_contracts = [  nid_registry,  nias_registry,  abatl_translation,  sequence_path_router,  clustering_contract  ]
if not all(required_contracts):  raise ValueError("One or more required N2N contracts failed to load")  print("All N2N contracts loaded successfully!")  # Setup test environment
registered_entities = setup_test_environment()  # Run tests  transmission_results = simulate_transmissions(registered_entities)  node_failure_results = test_node_failure_recovery(registered_entities)  clustering_results = test_clustering_efficiency(registered_entities)  comparison_results = compare_with_traditional_bgp()  # Visualize results
visualize_transmission_results(transmission_results)
visualize_node_failure_recovery(node_failure_results)
visualize_clustering_efficiency(clustering_results)
visualize_comparison_with_bgp(comparison_results)  # Generate summary report  report = generate_summary_report(
transmission_results,  node_failure_results,
clustering_results,  comparison_results  )
print("\nAll tests completed. Results saved to:", RESULT_OUTPUT_DIR)
print("\nSummary:")  print(report)  except Exception as e:
print(f"Error in N2N routing test execution: {e}")  import traceback  traceback.print_exc()  if __name__ == "__main__":  main()

# Table of Contents

migrations\4_initial_migration.js
..... [object Object]

migrations\5_initial_migration.js
..... [object Object]

migrations\6_initial_migration.js
..... [object Object]

migrations\7_initial_migration.js
..... [object Object]

migrations\8_initial_migration.js
..... [object Object]

migrations\9_initial_migration.js
..... [object Object]

next.config.js
..... [object Object]

scripts\deploy_ethereum.js
..... [object Object]

scripts\deploy_polkadot.js
..... [object Object]

test\cllimateintergration_test.js
..... [object Object]

test\uncertainty_analytics_test.js
..... [object Object]

truffle-config.js
..... [object Object]

.eslintrc.json
..... [object Object]

.next\build-manifest.json
..... [object Object]

.next\export-marker.json
..... [object Object]

.next\images-manifest.json
..... [object Object]

.next\package.json
..... [object Object]

.next\prerender-manifest.json
..... [object Object]

.next\react-loadable-manifest.json
..... [object Object]

.next\routes-manifest.json
..... [object Object]

.next\server\chunks\font-manifest.json

..... [object Object]

.next\server\font-manifest.json
..... [object Object]

.next\server\middleware-manifest.json
..... [object Object]

.next\server\next-font-manifest.json
..... [object Object]

.next\server\pages-manifest.json
..... [object Object]

.vscode\settings.json
..... [object Object]

build\contracts\TransactionTypes.json
..... [object Object]

config\contract_addresses.json
..... [object Object]

contract_addresses.json
..... [object Object]

launch.json
..... [object Object]

package.json
..... [object Object]

results\bcadn_analysis\network_analysis.json
..... [object Object]

results\bcadn_analysis\network_results.json
..... [object Object]

tsconfig.json
..... [object Object]

dataanalysis\egovernance_visualization..py
..... [object Object]

dataanalysis\statistical_model.py
..... [object Object]

python\BCADN.py
..... [object Object]

python\blockchainworkflow.py
..... [object Object]

python\climatemodule.py
..... [object Object]

python\companymodule.py
..... [object Object]

python\deploy_contracts.py
..... [object Object]

python\emmissionsmodule.py
..... [object Object]

python\healthmodule.py
..... [object Object]

python\renewalmodule.py
..... [object Object]

python\sequencerpath.py.py
..... [object Object]

python\test_n2n_routing.py
..... [object Object]