DecentraLearn Documentation

Comprehensive Guide to DecentraLearn Framework

# Table of Contents

1. Development Setup Guide

2. Code Style Guide

3. Testing Guide

4. API Documentation

5. Architecture Documentation

6. Tutorials

7. Examples

# Development Setup Guide

# Development Setup Guide  
  
This guide provides detailed instructions for setting up the DecentraLearn development environment.  
  
## Prerequisites  
  
### System Requirements  
  
1. \*\*Operating System\*\*  
 - Linux (recommended)  
 - macOS  
 - Windows (with WSL2)  
  
2. \*\*Software\*\*  
 - Python 3.8 or higher  
 - Git  
 - Virtual environment tool (venv or conda)  
 - Ethereum node (Ganache or local node)  
 - Docker (optional)  
  
3. \*\*Hardware\*\*  
 - 8GB RAM minimum  
 - 20GB free disk space  
 - GPU (optional, for model training)  
  
## Installation Steps  
  
### 1. Clone Repository  
  
```bash  
git clone https://github.com/yourusername/decentralearn.git  
cd decentralearn  
```  
  
### 2. Create Virtual Environment  
  
```bash  
# Using venv  
python -m venv venv  
source venv/bin/activate # Linux/macOS  
.\venv\Scripts\activate # Windows  
  
# Using conda  
conda create -n decentralearn python=3.8  
conda activate decentralearn  
```  
  
### 3. Install Dependencies  
  
```bash  
# Install development dependencies  
pip install -e ".[dev]"  
  
# Install testing dependencies  
pip install -e ".[test]"  
  
# Install documentation dependencies  
pip install -e ".[docs]"  
```  
  
### 4. Configure Development Tools  
  
1. \*\*Pre-commit Hooks\*\*  
 ```bash  
 pre-commit install  
 ```  
  
2. \*\*IDE Configuration\*\*  
 - VS Code: Install Python extension  
 - PyCharm: Configure Python interpreter  
 - Set up linting and formatting  
  
### 5. Setup Blockchain Environment  
  
1. \*\*Local Development\*\*  
 ```bash  
 # Start Ganache  
 ganache-cli  
  
 # Deploy contracts  
 python scripts/deploy\_contracts.py  
 ```  
  
2. \*\*Test Network\*\*  
 ```bash  
 # Configure test network  
 export WEB3\_PROVIDER\_URI="https://ropsten.infura.io/v3/YOUR-PROJECT-ID"  
 ```  
  
## Development Tools  
  
### Code Quality Tools  
  
1. \*\*Black (Code Formatter)\*\*  
 ```bash  
 # Format code  
 black decentralearn tests  
  
 # Check formatting  
 black --check decentralearn tests  
 ```  
  
2. \*\*Pylint (Linter)\*\*  
 ```bash  
 # Run linter  
 pylint decentralearn tests  
  
 # Generate report  
 pylint --output-format=html decentralearn > pylint.html  
 ```  
  
3. \*\*Mypy (Type Checker)\*\*  
 ```bash  
 # Check types  
 mypy decentralearn tests  
  
 # Generate report  
 mypy --html-report mypy-report decentralearn  
 ```  
  
### Documentation Tools  
  
1. \*\*Sphinx\*\*  
 ```bash  
 # Build documentation  
 cd docs  
 make html  
  
 # Serve documentation  
 python -m http.server -d \_build/html  
 ```  
  
2. \*\*Docstring Checker\*\*  
 ```bash  
 # Check docstrings  
 pydocstyle decentralearn tests  
 ```  
  
### Testing Tools  
  
1. \*\*pytest\*\*  
 ```bash  
 # Run tests  
 pytest  
  
 # Run with coverage  
 pytest --cov=decentralearn  
  
 # Run specific tests  
 pytest tests/blockchain/test\_client.py  
 ```  
  
2. \*\*tox\*\*  
 ```bash  
 # Run all environments  
 tox  
  
 # Run specific environment  
 tox -e py38  
 ```  
  
## Project Structure  
  
```  
decentralearn/  
├── decentralearn/ # Main package  
│ ├── blockchain/ # Blockchain integration  
│ ├── models/ # Machine learning models  
│ ├── privacy/ # Privacy mechanisms  
│ └── utils/ # Utility functions  
├── tests/ # Test suite  
├── docs/ # Documentation  
├── examples/ # Example scripts  
└── scripts/ # Development scripts  
```  
  
## Common Development Tasks  
  
### Creating New Features  
  
1. \*\*Create Feature Branch\*\*  
 ```bash  
 git checkout -b feature/new-feature  
 ```  
  
2. \*\*Implement Changes\*\*  
 ```python  
 # Add new functionality  
 def new\_feature():  
 pass  
 ```  
  
3. \*\*Add Tests\*\*  
 ```python  
 def test\_new\_feature():  
 assert new\_feature() == expected\_result  
 ```  
  
4. \*\*Update Documentation\*\*  
 ```markdown  
 # New Feature  
 Description of new feature...  
 ```  
  
### Updating Documentation  
  
1. \*\*Update Docstrings\*\*  
 ```python  
 def function():  
 """Updated documentation."""  
 pass  
 ```  
  
2. \*\*Update Markdown Files\*\*  
 ```bash  
 # Build documentation  
 cd docs  
 make html  
 ```  
  
### Running Tests  
  
1. \*\*Unit Tests\*\*  
 ```bash  
 pytest tests/unit/  
 ```  
  
2. \*\*Integration Tests\*\*  
 ```bash  
 pytest tests/integration/  
 ```  
  
3. \*\*All Tests\*\*  
 ```bash  
 pytest  
 ```  
  
## Troubleshooting  
  
### Common Issues  
  
1. \*\*Dependency Conflicts\*\*  
 ```bash  
 # Create fresh environment  
 rm -rf venv  
 python -m venv venv  
 pip install -e ".[dev]"  
 ```  
  
2. \*\*Blockchain Connection\*\*  
 ```bash  
 # Check connection  
 python scripts/check\_connection.py  
  
 # Reset blockchain  
 python scripts/reset\_blockchain.py  
 ```  
  
3. \*\*Test Failures\*\*  
 ```bash  
 # Run specific test  
 pytest -v tests/specific\_test.py  
  
 # Debug test  
 pytest --pdb tests/specific\_test.py  
 ```  
  
### Debugging  
  
1. \*\*Python Debugger\*\*  
 ```python  
 import pdb; pdb.set\_trace()  
 ```  
  
2. \*\*Logging\*\*  
 ```python  
 import logging  
 logging.basicConfig(level=logging.DEBUG)  
 ```  
  
## Best Practices  
  
1. \*\*Environment Management\*\*  
 - Use virtual environments  
 - Pin dependency versions  
 - Document environment setup  
 - Keep environment clean  
  
2. \*\*Version Control\*\*  
 - Follow git workflow  
 - Write meaningful commits  
 - Use feature branches  
 - Review changes  
  
3. \*\*Testing\*\*  
 - Write comprehensive tests  
 - Maintain test coverage  
 - Test edge cases  
 - Document test requirements  
  
4. \*\*Documentation\*\*  
 - Keep docs up to date  
 - Write clear docstrings  
 - Include examples  
 - Document changes  
  
## See Also  
  
- [Code Style Guide](code\_style.md)  
- [Testing Guide](testing.md)  
- [Development Guide](../development/README.md)  
- [API Documentation](../api/README.md)

# Code Style Guide

# Code Style Guide  
  
This guide outlines the coding standards and best practices for DecentraLearn development.  
  
## Python Style Guide  
  
### General Rules  
  
1. \*\*PEP 8 Compliance\*\*  
 - Follow PEP 8 style guide  
 - Use 4 spaces for indentation  
 - Maximum line length of 88 characters  
 - Use double quotes for strings  
  
2. \*\*Naming Conventions\*\*  
 ```python  
 # Classes: PascalCase  
 class BlockchainClient:  
 pass  
  
 # Functions and variables: snake\_case  
 def register\_client():  
 client\_address = "0x123..."  
  
 # Constants: UPPER\_CASE  
 MAX\_CLIENTS = 100  
 ```  
  
3. \*\*Imports\*\*  
 ```python  
 # Standard library imports  
 import os  
 import sys  
  
 # Third-party imports  
 import torch  
 import web3  
  
 # Local imports  
 from decentralearn.blockchain import client  
 from decentralearn.models import base  
 ```  
  
### Documentation  
  
1. \*\*Docstrings\*\*  
 ```python  
 def register\_client(client\_id: str) -> str:  
 """Register a new client on the blockchain.  
  
 Args:  
 client\_id: Unique identifier for the client.  
  
 Returns:  
 str: Blockchain address of the registered client.  
  
 Raises:  
 BlockchainError: If registration fails.  
 """  
 ```  
  
2. \*\*Type Hints\*\*  
 ```python  
 from typing import List, Dict, Optional  
  
 def get\_model(model\_id: str) -> Optional[BaseModel]:  
 """Get model from blockchain."""  
 ```  
  
3. \*\*Comments\*\*  
 ```python  
 # Use comments sparingly and only when necessary  
 # Explain why, not what  
 ```  
  
### Code Organization  
  
1. \*\*Class Structure\*\*  
 ```python  
 class BlockchainClient:  
 """Client for blockchain interactions."""  
  
 def \_\_init\_\_(self, config: BlockchainConfig):  
 """Initialize client."""  
 self.config = config  
 self.\_setup()  
  
 def \_setup(self):  
 """Setup internal state."""  
 pass  
  
 def public\_method(self):  
 """Public interface."""  
 pass  
 ```  
  
2. \*\*Function Design\*\*  
 ```python  
 def process\_data(  
 data: List[float],  
 threshold: float = 0.5,  
 normalize: bool = True  
 ) -> Dict[str, float]:  
 """Process data with optional normalization."""  
 ```  
  
## Testing Style  
  
1. \*\*Test Naming\*\*  
 ```python  
 def test\_client\_registration():  
 """Test client registration process."""  
  
 def test\_model\_verification\_with\_privacy():  
 """Test model verification with privacy mechanisms."""  
 ```  
  
2. \*\*Test Organization\*\*  
 ```python  
 class TestBlockchainClient:  
 """Test suite for blockchain client."""  
  
 def setup\_method(self):  
 """Setup test environment."""  
 self.client = BlockchainClient()  
  
 def test\_connection(self):  
 """Test blockchain connection."""  
 ```  
  
## Error Handling  
  
1. \*\*Exception Handling\*\*  
 ```python  
 try:  
 client.upload\_model(model)  
 except BlockchainError as e:  
 logger.error(f"Failed to upload model: {e}")  
 raise  
 ```  
  
2. \*\*Custom Exceptions\*\*  
 ```python  
 class BlockchainError(Exception):  
 """Base exception for blockchain operations."""  
  
 class ConnectionError(BlockchainError):  
 """Failed to connect to blockchain."""  
 ```  
  
## Security Guidelines  
  
1. \*\*Input Validation\*\*  
 ```python  
 def validate\_address(address: str) -> bool:  
 """Validate blockchain address."""  
 if not address.startswith("0x"):  
 raise ValueError("Invalid address format")  
 ```  
  
2. \*\*Secure Defaults\*\*  
 ```python  
 class SecurityConfig:  
 def \_\_init\_\_(self, encryption: str = "AES-256"):  
 """Initialize with secure defaults."""  
 ```  
  
## Best Practices  
  
1. \*\*Code Quality\*\*  
 - Write clean, readable code  
 - Use meaningful variable names  
 - Keep functions focused  
 - Document complex logic  
  
2. \*\*Performance\*\*  
 - Optimize critical paths  
 - Use appropriate data structures  
 - Minimize memory usage  
 - Profile code regularly  
  
3. \*\*Maintainability\*\*  
 - Follow DRY principle  
 - Write modular code  
 - Use design patterns  
 - Keep dependencies minimal  
  
4. \*\*Testing\*\*  
 - Write comprehensive tests  
 - Test edge cases  
 - Maintain test coverage  
 - Use fixtures effectively  
  
## Tools and Configuration  
  
1. \*\*Black Configuration\*\*  
 ```toml  
 [tool.black]  
 line-length = 88  
 target-version = ['py38']  
 include = '\.pyi?$'  
 ```  
  
2. \*\*Pylint Configuration\*\*  
 ```ini  
 [MASTER]  
 disable = C0111  
 max-line-length = 88  
  
 [FORMAT]  
 max-line-length = 88  
 ```  
  
3. \*\*Mypy Configuration\*\*  
 ```ini  
 [mypy]  
 python\_version = 3.8  
 warn\_return\_any = True  
 warn\_unused\_configs = True  
 ```  
  
## See Also  
  
- [PEP 8 Style Guide](https://www.python.org/dev/peps/pep-0008/)  
- [Google Python Style Guide](https://google.github.io/styleguide/pyguide.html)  
- [Testing Guide](testing.md)  
- [Development Setup](setup.md)

# Testing Guide

# Testing Guide  
  
This guide provides comprehensive information about testing in DecentraLearn.  
  
## Testing Strategy  
  
### Test Types  
  
1. \*\*Unit Tests\*\*  
 - Test individual components  
 - Isolate dependencies  
 - Fast execution  
 - High coverage  
  
2. \*\*Integration Tests\*\*  
 - Test component interactions  
 - Verify system behavior  
 - Include blockchain interactions  
 - Test privacy mechanisms  
  
3. \*\*Performance Tests\*\*  
 - Measure execution time  
 - Monitor resource usage  
 - Test scalability  
 - Benchmark privacy overhead  
  
## Writing Tests  
  
### Test Structure  
  
1. \*\*Basic Test\*\*  
 ```python  
 def test\_client\_registration():  
 """Test client registration process."""  
 client = BlockchainClient()  
 address = client.register("test\_client")  
 assert address.startswith("0x")  
 ```  
  
2. \*\*Test Class\*\*  
 ```python  
 class TestModelVerification:  
 """Test suite for model verification."""  
  
 def setup\_method(self):  
 """Setup test environment."""  
 self.client = BlockchainClient()  
 self.model = BaseModel()  
  
 def test\_verification\_success(self):  
 """Test successful verification."""  
 result = self.client.verify\_model(self.model)  
 assert result is True  
  
 def test\_verification\_failure(self):  
 """Test failed verification."""  
 invalid\_model = InvalidModel()  
 with pytest.raises(VerificationError):  
 self.client.verify\_model(invalid\_model)  
 ```  
  
### Test Fixtures  
  
1. \*\*Basic Fixture\*\*  
 ```python  
 @pytest.fixture  
 def blockchain\_client():  
 """Create blockchain client for testing."""  
 client = BlockchainClient()  
 yield client  
 client.cleanup()  
 ```  
  
2. \*\*Parameterized Fixture\*\*  
 ```python  
 @pytest.fixture(params=[32, 64, 128])  
 def batch\_size(request):  
 """Test different batch sizes."""  
 return request.param  
 ```  
  
### Mocking  
  
1. \*\*Basic Mock\*\*  
 ```python  
 def test\_model\_upload(mocker):  
 """Test model upload with mocked blockchain."""  
 mock\_client = mocker.Mock()  
 mock\_client.upload\_model.return\_value = "0x123"  
   
 result = upload\_model(mock\_client, BaseModel())  
 assert result == "0x123"  
 ```  
  
2. \*\*Mock with Side Effects\*\*  
 ```python  
 def test\_error\_handling(mocker):  
 """Test error handling in blockchain operations."""  
 mock\_client = mocker.Mock()  
 mock\_client.upload\_model.side\_effect = BlockchainError("Failed")  
   
 with pytest.raises(BlockchainError):  
 upload\_model(mock\_client, BaseModel())  
 ```  
  
## Running Tests  
  
### Basic Commands  
  
1. \*\*Run All Tests\*\*  
 ```bash  
 pytest  
 ```  
  
2. \*\*Run Specific Tests\*\*  
 ```bash  
 pytest tests/blockchain/test\_client.py  
 pytest tests/blockchain/test\_client.py::test\_registration  
 ```  
  
3. \*\*Run with Coverage\*\*  
 ```bash  
 pytest --cov=decentralearn  
 pytest --cov=decentralearn --cov-report=html  
 ```  
  
### Test Configuration  
  
1. \*\*pytest.ini\*\*  
 ```ini  
 [pytest]  
 testpaths = tests  
 python\_files = test\_\*.py  
 python\_classes = Test\*  
 python\_functions = test\_\*  
 ```  
  
2. \*\*coverage.ini\*\*  
 ```ini  
 [run]  
 source = decentralearn  
 omit = tests/\*,setup.py  
  
 [report]  
 exclude\_lines =  
 pragma: no cover  
 def \_\_repr\_\_  
 raise NotImplementedError  
 ```  
  
## Test Categories  
  
### Blockchain Tests  
  
1. \*\*Client Tests\*\*  
 ```python  
 def test\_client\_connection():  
 """Test blockchain connection."""  
 client = BlockchainClient()  
 assert client.is\_connected()  
  
 def test\_transaction\_signing():  
 """Test transaction signing."""  
 client = BlockchainClient()  
 tx = client.sign\_transaction({"to": "0x123"})  
 assert tx["signature"] is not None  
 ```  
  
2. \*\*Contract Tests\*\*  
 ```python  
 def test\_contract\_deployment():  
 """Test smart contract deployment."""  
 client = BlockchainClient()  
 contract = client.deploy\_contract("ModelRegistry")  
 assert contract.address is not None  
 ```  
  
### Privacy Tests  
  
1. \*\*Differential Privacy\*\*  
 ```python  
 def test\_dp\_noise\_addition():  
 """Test differential privacy noise."""  
 data = torch.randn(100)  
 private\_data = add\_dp\_noise(data, epsilon=1.0)  
 assert not torch.allclose(data, private\_data)  
 ```  
  
2. \*\*Homomorphic Encryption\*\*  
 ```python  
 def test\_he\_encryption():  
 """Test homomorphic encryption."""  
 data = torch.tensor([1.0, 2.0, 3.0])  
 encrypted = encrypt\_data(data)  
 assert encrypted.shape == data.shape  
 ```  
  
### Model Tests  
  
1. \*\*Training Tests\*\*  
 ```python  
 def test\_model\_training():  
 """Test model training process."""  
 model = BaseModel()  
 dataset = CustomDataset()  
 trainer = Trainer(model)  
 loss = trainer.train(dataset)  
 assert loss < 1.0  
 ```  
  
2. \*\*Verification Tests\*\*  
 ```python  
 def test\_model\_verification():  
 """Test model verification."""  
 model = BaseModel()  
 verifier = ModelVerifier()  
 result = verifier.verify(model)  
 assert result["valid"] is True  
 ```  
  
## Best Practices  
  
1. \*\*Test Organization\*\*  
 - Group related tests  
 - Use descriptive names  
 - Follow test hierarchy  
 - Maintain test isolation  
  
2. \*\*Test Quality\*\*  
 - Write meaningful tests  
 - Cover edge cases  
 - Test error conditions  
 - Verify expected behavior  
  
3. \*\*Performance\*\*  
 - Minimize test duration  
 - Use appropriate fixtures  
 - Avoid unnecessary I/O  
 - Mock external services  
  
4. \*\*Maintenance\*\*  
 - Keep tests up to date  
 - Remove obsolete tests  
 - Update test data  
 - Document test changes  
  
## See Also  
  
- [pytest Documentation](https://docs.pytest.org/)  
- [Testing Setup](setup.md)  
- [Code Style Guide](code\_style.md)  
- [Development Guide](../development/README.md)

# API Documentation

# API Reference  
  
This section provides detailed documentation for DecentraLearn's API components.  
  
## Core Components  
  
- [Blockchain Client](blockchain\_client.md)  
 - Client registration  
 - Model management  
 - Transaction handling  
 - Contract interaction  
  
- [Model Management](model\_management.md)  
 - Model serialization  
 - State management  
 - Version control  
 - Verification  
  
- [Dataset Management](dataset\_management.md)  
 - Dataset loading  
 - Data splitting  
 - Privacy handling  
 - Custom datasets  
  
- [Configuration](configuration.md)  
 - Blockchain settings  
 - Privacy parameters  
 - Model configuration  
 - Network settings  
  
## Usage Examples  
  
### Basic Setup  
  
```python  
from decentralearn.blockchain.client import BlockchainClient  
from decentralearn.config.blockchain\_config import BlockchainConfig  
  
config = BlockchainConfig(  
 rpc\_url="http://localhost:8545",  
 chain\_id=1337  
)  
client = BlockchainClient(config)  
```  
  
### Model Management  
  
```python  
from decentralearn.models.base import BaseModel  
from decentralearn.privacy import DifferentialPrivacy  
  
model = BaseModel()  
dp = DifferentialPrivacy(epsilon=0.1)  
```  
  
### Dataset Handling  
  
```python  
from decentralearn.datasets import DatasetFactory  
from decentralearn.datasets import DatasetSpliter  
  
factory = DatasetFactory()  
spliter = DatasetSpliter()  
```  
  
## Best Practices  
  
1. Always initialize blockchain connection first  
2. Use appropriate privacy mechanisms  
3. Follow configuration guidelines  
4. Implement proper error handling  
5. Monitor privacy budgets  
  
## See Also  
  
- [Architecture Overview](../architecture/overview.md)  
- [Tutorials](../tutorials/README.md)  
- [Examples](../examples/README.md)

# Architecture Documentation

# Architecture Overview  
  
This document provides a comprehensive overview of DecentraLearn's architecture.  
  
## Table of Contents  
  
1. [System Architecture](#system-architecture)  
2. [Key Components](#key-components)  
3. [Data Flow](#data-flow)  
4. [Security Considerations](#security-considerations)  
5. [Scalability Features](#scalability-features)  
  
## System Architecture  
  
```  
+------------------+ +------------------+ +------------------+  
| | | | | |  
| Client Node 1 | | Client Node 2 | | Client Node N |  
| | | | | |  
+------------------+ +------------------+ +------------------+  
 | | |  
 v v v  
+----------------------------------------------------------+  
| |  
| Blockchain Network |  
| |  
+----------------------------------------------------------+  
 ^ ^ ^  
 | | |  
+------------------+ +------------------+ +------------------+  
| | | | | |  
| Smart Contract | | Model Registry | | Data Registry |  
| | | | | |  
+------------------+ +------------------+ +------------------+  
```  
  
## Key Components  
  
### 1. Client Nodes  
  
- \*\*Local Training\*\*:  
 - Dataset management  
 - Model training  
 - Gradient computation  
 - Model validation  
  
- \*\*Communication\*\*:  
 - Blockchain interaction  
 - Model submission  
 - Gradient sharing  
 - State synchronization  
  
### 2. Blockchain Network  
  
- \*\*Smart Contracts\*\*:  
 - Model registration  
 - Gradient aggregation  
 - Incentive mechanism  
 - Access control  
  
- \*\*Consensus Mechanism\*\*:  
 - Proof of Work/Stake  
 - Transaction validation  
 - Block creation  
 - Network synchronization  
  
### 3. Registry Components  
  
- \*\*Model Registry\*\*:  
 - Model storage  
 - Version control  
 - Access management  
 - Verification system  
  
- \*\*Data Registry\*\*:  
 - Dataset metadata  
 - Access control  
 - Usage tracking  
 - Privacy management  
  
## Data Flow  
  
### 1. Training Process  
  
```  
1. Initialization  
 - Load local dataset  
 - Initialize model  
 - Connect to blockchain  
  
2. Local Training  
 - Compute gradients  
 - Update model  
 - Validate results  
  
3. Blockchain Interaction  
 - Submit gradients  
 - Participate in consensus  
 - Receive rewards  
  
4. Global Aggregation  
 - Collect gradients  
 - Compute average  
 - Update global model  
```  
  
### 2. Model Updates  
  
```  
1. Local Update  
 - Train on local data  
 - Compute gradients  
 - Validate model  
  
2. Blockchain Submission  
 - Sign transaction  
 - Submit to network  
 - Wait for confirmation  
  
3. Global Update  
 - Aggregate updates  
 - Compute new model  
 - Distribute to clients  
```  
  
### 3. Data Management  
  
```  
1. Dataset Preparation  
 - Load data  
 - Preprocess  
 - Split for training  
  
2. Privacy Protection  
 - Apply encryption  
 - Add noise  
 - Mask sensitive data  
  
3. Blockchain Storage  
 - Store metadata  
 - Track usage  
 - Manage access  
```  
  
## Security Considerations  
  
### 1. Data Privacy  
  
- \*\*Encryption\*\*:  
 - End-to-end encryption  
 - Homomorphic encryption  
 - Secure multi-party computation  
  
- \*\*Access Control\*\*:  
 - Role-based access  
 - Permission management  
 - Audit logging  
  
### 2. Model Security  
  
- \*\*Verification\*\*:  
 - Model validation  
 - Gradient checking  
 - Consensus verification  
  
- \*\*Protection\*\*:  
 - Digital signatures  
 - Watermarking  
 - Tamper detection  
  
### 3. Network Security  
  
- \*\*Authentication\*\*:  
 - Public key infrastructure  
 - Digital certificates  
 - Two-factor authentication  
  
- \*\*Communication\*\*:  
 - Secure channels  
 - Message encryption  
 - Integrity checks  
  
## Scalability Features  
  
### 1. Horizontal Scaling  
  
- \*\*Client Nodes\*\*:  
 - Dynamic joining/leaving  
 - Load balancing  
 - Resource management  
  
- \*\*Blockchain Network\*\*:  
 - Sharding  
 - Sidechains  
 - State channels  
  
### 2. Performance Optimization  
  
- \*\*Training\*\*:  
 - Parallel processing  
 - Batch processing  
### 1. Blockchain Layer  
- \*\*Ethereum Network\*\*: Handles model updates and verification  
- \*\*Smart Contracts\*\*:   
 - `FLContract.sol`: Manages client registration and model verification  
 - `ModelRegistry.sol`: Tracks model versions and metadata  
- \*\*Blockchain Client\*\*: Python interface for blockchain interactions  
  
### 2. Federated Learning Layer  
- \*\*Client Management\*\*: Handles client registration and authentication  
- \*\*Model Management\*\*:   
 - Model serialization and deserialization  
 - State tracking and versioning  
 - Parameter aggregation  
- \*\*Dataset Management\*\*:  
 - Built-in datasets (FashionMNIST, CIFAR10, etc.)  
 - Custom dataset support  
 - Data splitting utilities  
  
### 3. Security Layer  
- \*\*Model Verification\*\*: On-chain verification of model updates  
- \*\*Client Authentication\*\*: Secure client registration and management  
- \*\*Data Privacy\*\*: Federated learning ensures data privacy  
- \*\*Smart Contract Security\*\*: Secure contract execution and state management  
  
## Data Flow  
  
1. \*\*Initialization\*\*:  
 - Clients register with the blockchain network  
 - Dataset is split among clients  
 - Initial model is distributed  
  
2. \*\*Training Cycle\*\*:  
 - Clients train on their local data  
 - Model updates are serialized and uploaded  
 - Updates are verified on-chain  
 - Verified updates are aggregated  
  
3. \*\*Verification\*\*:  
 - Model updates are verified against previous versions  
 - Client contributions are tracked  
 - Model state is updated in the registry  
  
## Security Considerations  
  
1. \*\*Model Integrity\*\*:  
 - On-chain verification of model updates  
 - Cryptographic signatures for client authentication  
 - Secure parameter aggregation  
  
2. \*\*Data Privacy\*\*:  
 - Local training ensures data privacy  
 - No raw data is shared or stored on-chain  
 - Secure model parameter transmission  
  
3. \*\*Smart Contract Security\*\*:  
 - Access control for contract functions  
 - State management and versioning  
 - Gas optimization for operations  
  
## Scalability  
  
The architecture is designed to be scalable:  
- Modular components allow for easy extension  
- Smart contracts handle multiple clients efficiently  
- Dataset management supports various data types and sizes  
- Model management can handle complex neural networks  
  
## Next Steps  
  
- [Core Components](components.md): Detailed explanation of each component  
- [Data Flow](data\_flow.md): In-depth look at system data flow  
- [Security Model](security.md): Comprehensive security considerations

# Tutorials

# Tutorials  
  
This section provides step-by-step tutorials for using DecentraLearn.  
  
## Getting Started  
  
- [Quick Start](getting\_started.md)  
 - Installation  
 - Basic setup  
 - First model  
 - Blockchain integration  
  
## Core Features  
  
- [Federated Learning](federated\_learning.md)  
 - Model training  
 - Client coordination  
 - Privacy mechanisms  
 - Blockchain verification  
  
- [Model Training](model\_training.md)  
 - Model creation  
 - Training process  
 - Privacy integration  
 - Performance optimization  
  
- [Custom Dataset](custom\_dataset.md)  
 - Dataset creation  
 - Data loading  
 - Privacy handling  
 - Custom splitting  
  
- [Non-IID Distribution](non\_iid\_distribution.md)  
 - Data distribution  
 - Client assignment  
 - Performance impact  
 - Mitigation strategies  
  
## Advanced Topics  
  
- Privacy Mechanisms  
 - Differential Privacy  
 - Homomorphic Encryption  
 - Zero-Knowledge Proofs  
  
- Blockchain Integration  
 - Smart contracts  
 - Model verification  
 - Incentive mechanisms  
  
- Security Features  
 - Access control  
 - Data protection  
 - Audit logging  
  
## Best Practices  
  
1. Follow the tutorials in order  
2. Complete all code examples  
3. Test thoroughly  
4. Monitor privacy budgets  
5. Check blockchain status  
  
## See Also  
  
- [API Reference](../api/README.md)  
- [Examples](../examples/README.md)  
- [Development Guide](../development/README.md)

# Examples

# Examples  
  
This section provides practical examples of using DecentraLearn.  
  
## Basic Examples  
  
- [MNIST Classification](mnist\_classification.md)  
 - Model definition  
 - Training process  
 - Privacy integration  
 - Blockchain verification  
  
- [Custom Dataset](custom\_dataset.md)  
 - Dataset creation  
 - Data loading  
 - Privacy handling  
 - Model training  
  
- [Non-IID Distribution](non\_iid\_distribution.md)  
 - Data distribution  
 - Client assignment  
 - Performance analysis  
 - Privacy impact  
  
## Advanced Examples  
  
- Privacy Mechanisms  
 - Differential Privacy  
 - Homomorphic Encryption  
 - Zero-Knowledge Proofs  
  
- Blockchain Integration  
 - Smart contracts  
 - Model verification  
 - Incentive mechanisms  
  
- Security Features  
 - Access control  
 - Data protection  
 - Audit logging  
  
## Code Snippets  
  
### Basic Setup  
  
```python  
from decentralearn.blockchain.client import BlockchainClient  
from decentralearn.config.blockchain\_config import BlockchainConfig  
  
config = BlockchainConfig(  
 rpc\_url="http://localhost:8545",  
 chain\_id=1337  
)  
client = BlockchainClient(config)  
```  
  
### Model Training  
  
```python  
from decentralearn.models.base import BaseModel  
from decentralearn.privacy import DifferentialPrivacy  
  
model = BaseModel()  
dp = DifferentialPrivacy(epsilon=0.1)  
# Train model with privacy  
```  
  
### Dataset Handling  
  
```python  
from decentralearn.datasets import DatasetFactory  
from decentralearn.datasets import DatasetSpliter  
  
factory = DatasetFactory()  
spliter = DatasetSpliter()  
```  
  
## Best Practices  
  
1. Study the examples thoroughly  
2. Run the code yourself  
3. Modify parameters  
4. Test different scenarios  
5. Monitor performance  
  
## See Also  
  
- [API Reference](../api/README.md)  
- [Tutorials](../tutorials/README.md)  
- [Development Guide](../development/README.md)