**Introduction**

The primary objective of this research project is to assess the viability and effectiveness of utilizing ChatGPT, an advanced AI conversational model, in the generation of chaincode for Hyperledger Fabric networks. This investigation aims to explore the potential of ChatGPT to streamline and innovate the process of chaincode development, thereby contributing to the efficiency and robustness of blockchain solutions.

The advent of AI in programming, particularly in automated code generation, has opened new frontiers. Tools like ChatGPT represent a significant leap in this area, offering capabilities to generate, optimize, and even debug code based on conversational inputs. The potential of such tools in blockchain development, especially in the creation of chaincode, is immense. AI-driven programming can not only accelerate the development process but also introduce new levels of abstraction and efficiency, potentially reducing the risk of human error and enhancing the quality of the code.

In this exploratory phase, our research delves into the practicality and effectiveness of ChatGPT in the automated programming of chaincode for Hyperledger Fabric. We aim to establish a methodology for creating reliable, efficient, and secure chaincode using AI tools, marking a significant step forward in the integration of AI with blockchain technology.

**Methodology**

The approach for generating chaincode using ChatGPT for end-to-end chaincode involves several key steps and considerations:

Understanding Specific Requirements: The process begins with a clear understanding of the requirements for the chaincode. This involves identifying the specific business logic and network details that the chaincode needs to address. For instance, the supply chain management chaincode is designed to track packages from origin to delivery, including payment release mechanisms​​.

Defining Data Structures and Chaincode Functions: The chaincode is structured around key data structures and functions relevant to the application. For example, in the supply chain management scenario, structures like Asset, Handler, and Payment are defined, along with functions to create and update assets, acknowledge receipt, and release payments​​.

Tailoring to Specific Industries: The chaincode is customized for specific industry needs, such as pharmaceuticals. This involves adding features like batch numbers, expiry dates, storage conditions, recall handling, and counterfeit protection​​.

Event Handling and Additional Scenarios: Implementing event handling for various conditions like package loss, damage, or holding issues, and addressing additional scenarios specific to the industry, such as storage condition validations and handling counterfeit medications​​.

Incorporating Token and NFT Functionality: Extending the chaincode to include functionalities for managing tokens and non-fungible tokens (NFTs). This includes creating and managing token balances, minting and transferring tokens or NFTs, and querying balances or NFT details, similar to functionalities in blockchain systems like Ethereum​​.

Input Assumptions and Validations: Making assumptions about the format of key inputs and implementing validations to ensure that the data provided to the chaincode meets expected standards. This step is crucial for maintaining data integrity and smooth operation of the chaincode​​.

Adapting to Real-World Applications: Recognizing that the provided chaincode examples are basic representations and need further details and robustness for real-world applications. This includes considering more granular details, business-specific logic, intricate condition checks, and external system integrations​​.

We evaluated the generated chaincodes by using the following test cases and scenarios:

* We generated eight different chaincodes named DEX AMM, DEX Orderbook, First Price Bid, Mint NFT Supplychain, NFT Collectibles, NFT Event Ticket, NFT Real Estate, and NFT Second Price Bid by providing ChatGPT with the necessary information and instructions for each chaincode.
* We copied the generated chaincodes into a “Hyperledger Fabric” project generated by “Hyperledger Fablo”. Fablo had “Fablo REST” and “Blockchain Explorer” set up for the project.
* We used the APIs for each chaincode generated by Fablo REST to test the functionality and performance of the chaincodes. We checked the input and output parameters, the response time, and the error handling of the APIs.
* We used Blockchain Explorer to test the generated transactions and blocks for each successful API call. We verified the transaction ID, the block number, the payload, and the endorsement policy of the transactions and blocks.

**Test Results**

*Successes*: ChatGPT successfully generated functional chaincodes for most of the test cases and scenarios, noting the specificity and completeness of the provided instructions. Some of the successes are:

* ChatGPT generated chaincodes that followed the Hyperledger Fabric programming model and used the contractapi package for the smart contract interface.
* ChatGPT generated chaincodes that implemented the basic CRUD operations (create, read, update, delete) for the assets and participants in the network.
* ChatGPT generated chaincodes that used the appropriate data types and structures for the assets and participants, such as structs, arrays, and maps.
* ChatGPT generated chaincodes that used the correct syntax and formatting for the Go language, such as indentation, comments, and naming conventions.
* ChatGPT generated chaincodes that included some validation and error-handling logic for the input parameters and the API calls.

*Shortcomings*: ChatGPT also had some shortcomings that were identified during testing, such as package name adjustments, and handling of contractapi.Contract, and incomplete business logic. Some of the shortcomings are:

* ChatGPT generated chaincodes all of which had the same package name main, which did not match the package name of the Hyperledger Fabric project. We had to manually change the package name to the correct one before copying the chaincode.
* ChatGPT generated chaincodes all of which had a main method. However, if we already had a main method under the same package, it was causing compilation errors. We had to remove the redundant main method and keep only one main method that invoked the contractapi.Contract.
* ChatGPT generated chaincodes that used the shim and peer package from GitHub, which was deprecated and not recommended for the latest version of Hyperledger Fabric. We had to instruct ChatGPT to only use the contractapi package and avoid using the shim and peer package.
* ChatGPT generated chaincodes that did not use the contractapi.Contract struct for some chaincodes, which prevented the chaincode from being registered and invoked by the contractapi package. We had to modify the chaincode to use the contractapi.Contract struct as shown below.

Wrong ->

type SmartContract struct {

}

func (s \*SmartContract) Init(APIstub shim.ChaincodeStubInterface) sc.Response {

return shim.Success(nil)

}

func (s \*SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) sc.Response {

// ...

}

Right ->

type SmartContract struct {

contractapi.Contract

}

func (s \*SmartContract) Init(ctx contractapi.TransactionContextInterface) error {

return nil

}

func (s \*SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) sc.Response {

// ...

}

* ChatGPT generated chaincodes that were not aligned with the latest update of some used packages, such as the contractapi package. For example, a method that previously returned only one single value has been updated in the latest release so that it returns two values, the main value and an error object. But the chaincode only received one value from the method assuming the previous version, which threw an error as it was not receiving the second object which was an error object. We had to fix this by receiving the second object as “\_” and then ignoring it.

Wrong ->

...

assetAsBytes := ctx.GetStub().GetState(assetID)

...

Right ->

...

assetAsBytes, \_ := ctx.GetStub().GetState(assetID)

...

* ChatGPT generated chaincodes that were not complete as they were missing some business logic. Instead, ChatGPT just wrote some comments in those places and instructed the coder to write the logic. For example, for the DEX Orderbook chaincode, ChatGPT did not implement the logic for matching the orders and executing the trades. We had to write the logic ourselves or provide ChatGPT with more specific information and instructions for the business logic.

Wrong ->

...

// TODO: write the logic for matching the orders and executing the trades

...

Right ->

...

// loop through the buy orders and the sell orders

for i := 0; i < len(buyOrders) && j < len(sellOrders); i++ {

// compare the prices and quantities of the orders

if buyOrders[i].Price >= sellOrders[j].Price && buyOrders[i].Quantity <= sellOrders[j].Quantity {

// execute the trade by transferring the tokens between the buyers and sellers

err := s.transferTokens(ctx, buyOrders[i].Buyer, sellOrders[j].Seller, buyOrders[i].Quantity, buyOrders[i].Price)

if err != nil {

return err

}

// update the orders by reducing the quantities or removing them if they are filled

buyOrders[i].Quantity = 0

sellOrders[j].Quantity -= buyOrders[i].Quantity

if sellOrders[j].Quantity == 0 {

j++

}

}

}

...

* ChatGPT generated chaincodes that did not return the result of the ctx.GetStub().PutState() method for some chaincode methods, which made it difficult to check the success or failure of the state update. We had to return the result of the ctx.GetStub().PutState() method for those chaincode methods.

Wrong ->

...

func (s \*SmartContract) CreateAsset(ctx contractapi.TransactionContextInterface, assetID string, assetName string, assetValue int) error {

// ...

ctx.GetStub().PutState(assetID, assetAsBytes)

return nil

}

...

Right ->

...

func (s \*SmartContract) CreateAsset(ctx contractapi.TransactionContextInterface, assetID string, assetName string, assetValue int) error {

// ...

return ctx.GetStub().PutState(assetID, assetAsBytes)

}

...

**Analysis**

*Effectiveness in Different Scenarios*: ChatGPT showed different levels of effectiveness in chaincode generation depending on the complexity and specificity of the scenarios. Some of the observations are:

* ChatGPT excelled in chaincode generation for standard operations, such as creating, reading, updating, and deleting assets and participants in the network. ChatGPT was able to generate chaincode methods that performed these operations with the correct parameters and logic.
* ChatGPT also performed well in chaincode generation for template generation, such as creating a basic smart contract structure, defining the asset and participant structs, and invoking the contractapi.Contract. ChatGPT was able to generate chaincode templates that followed the best practices and conventions of “Hyperledger Fabric” and “Go” programming.
* ChatGPT struggled in chaincode generation for complex business logic, such as implementing the logic for decentralized exchanges, auctions, and non-fungible tokens. ChatGPT often left some parts of the logic incomplete or incorrect, requiring manual intervention and modification. ChatGPT also had difficulty in handling the edge cases and exceptions for the business logic.
* ChatGPT also had challenges in chaincode generation for specific network configurations, such as setting the endorsement policy, the collection policy, and the chaincode lifecycle. ChatGPT did not generate any code for these configurations, which had to be done separately by using the Hyperledger Fabric CLI or SDK.

**Requirement for Detailed Inputs**

There are two key areas in the context of using ChatGPT for generating chaincode:

Requirement for Detailed Inputs: The level of detail in instructions significantly impacts the quality and completeness of the generated chaincode. Detailed, specific instructions enable ChatGPT to generate more accurate and complete chaincode that aligns closely with the intended business logic and technical requirements. Conversely, vague or general instructions may result in chaincode that lacks crucial elements or contains inaccuracies, necessitating manual corrections or additions.

Use-Case Suitability: ChatGPT is particularly beneficial in certain use cases or stages of blockchain development:

Prototyping: ChatGPT can rapidly generate prototypes of chaincode, allowing developers to quickly conceptualize and iterate on blockchain solutions. This is especially useful in the early stages of project development when exploring various possibilities and solutions.

Educational Purposes: ChatGPT can be a valuable tool for educational scenarios, helping learners understand blockchain concepts, chaincode structure, and coding practices. It can generate example codes and explain blockchain mechanisms, serving as an interactive learning aid.

In summary, the effectiveness of ChatGPT in generating chaincode largely depends on the specificity of the instructions provided. It excels in scenarios like prototyping and education, where rapid development and conceptual understanding are key objectives​​.

**Conclusion**

Feasibility of Using ChatGPT for Chaincode Generation:

* Overall Assessment: ChatGPT's effectiveness in chaincode generation is influenced by the level of detail in the provided instructions and the specific use cases of the blockchain project. While it shows potential, the success in chaincode generation depends on various factors such as the complexity of the project, the specificity of instructions, and the need for manual interventions or modifications in certain scenarios.

Recommendations for Utilizing ChatGPT in Chaincode Development:

* Focus on Specificity and Detail in Instructions: Given that the quality of the generated chaincode is highly dependent on the detail and specificity of the instructions, it is recommended to provide ChatGPT with clear, detailed, and specific requirements to ensure more accurate and complete chaincode output.
* Best Suited for Prototyping and Early Development Stages: ChatGPT is particularly beneficial during the prototyping phase and early stages of blockchain development. It can rapidly generate prototypes, allowing developers to quickly conceptualize, iterate, and explore various possibilities.
* Educational Tool for Blockchain Concepts and Coding Practices: ChatGPT serves as a valuable educational tool, helping learners understand blockchain concepts, chaincode structure, and coding practices. It can generate example codes and explain mechanisms, making it an effective interactive learning aid.
* Limitations in Complex Business Logic and Specific Configurations: For projects involving complex business logic or specific network configurations, ChatGPT might require supplementary manual intervention. In such cases, it's advised to use ChatGPT in conjunction with expert human oversight.
* Iterative Development Approach: Utilize ChatGPT in an iterative manner, refining and elaborating requirements as the project evolves. This approach can harness the benefits of ChatGPT while accommodating the dynamic nature of blockchain project development.

In summary, while ChatGPT offers promising capabilities in chaincode generation, its effectiveness is contingent upon the clarity and detail of user inputs and the nature of the blockchain project. It is most effective in early development stages, prototyping, and educational settings, with a recommendation for iterative use and expert supervision in more complex scenarios​​.

**Future Considerations**

Based on the research efforts, possible areas for further exploration might include:

Advanced Integration with Development Environments: Investigating deeper integration of ChatGPT with popular blockchain development environments and tools. This could involve creating plugins or extensions that allow developers to directly interact with ChatGPT within their preferred IDEs, streamlining the chaincode development process.

Enhanced Support for Complex Blockchain Architectures: Expanding ChatGPT's capabilities to better understand and generate code for complex blockchain architectures, including multi-chain setups, sidechains, and layer 2 solutions. This would make it a more versatile tool for a broader range of blockchain applications.

Automated Testing and Security Analysis: Developing methodologies for ChatGPT to assist in automated testing of chaincode, including security vulnerability scanning and performance benchmarking. This could significantly enhance the reliability and security of chaincodes.

Customization for Industry-Specific Blockchain Applications: Tailoring ChatGPT's capabilities to suit specific industries such as finance, healthcare, or supply chain, where blockchain is increasingly being adopted. This would involve training or programming ChatGPT to understand industry-specific terminologies, regulations, and business processes​​.