LLM4Fin: Fully Automating LLM-Powered Test Case Generation for FinTech Software Acceptance Testing

Zhiyi Xue
Shanghai Key Laboratory of Trustworthy Computing, ECNU
Shanghai, China

Liangguo Li
Shanghai Key Laboratory of Trustworthy Computing, ECNU
Shanghai, China

Senyue Tian
Shanghai Key Laboratory of Trustworthy Computing, ECNU
Shanghai, China

Xiaohong Chen*
Shanghai Key Laboratory of Trustworthy Computing, ECNU
Shanghai, China

Pingping Li
Software Testing Center
Guotai Junan Securities Co. Ltd.
Shanghai, China

Liangyu Chen
Shanghai Key Laboratory of Trustworthy Computing, ECNU
Shanghai, China

Tingting Jiang
Software Testing Center
Guotai Junan Securities Co. Ltd.
Shanghai, China

Min Zhang
Dishui Lake International Software Engineering Institute, ECNU
Shanghai, China

Abstract

FinTech software, crucial for both safety and timely market deployment, presents a compelling case for automated acceptance testing against regulatory business rules. However, the inherent challenges of comprehending unstructured natural language descriptions of these rules and crafting comprehensive test cases demand human intelligence. The emergence of Large Language Models (LLMs) holds promise for automated test case generation, leveraging their natural language processing capabilities. Yet, their dependence on human intervention for effective prompting hampers efficiency.

In response, we introduce a groundbreaking, fully automated approach for generating high-coverage test cases from natural language business rules. Our methodology seamlessly integrates the versatility of LLMs with the predictability of algorithmic methods. We fine-tune pre-trained LLMs for improved information extraction accuracy and algorithmically generate comprehensive testable scenarios for the extracted business rules. Our prototype, LLM4Fin, is designed for testing real-world stock-trading software. Experimental results demonstrate LLM4Fin's superiority over both state-of-the-art LLM, such as ChatGPT, and skilled testing engineers. We achieve remarkable performance, with up to 98.18% and an average of 20% − 110% improvement on business scenario coverage, and up to 93.72% on code coverage, while reducing the time cost from 20 minutes to a mere 7 seconds. These results provide robust evidence of the framework's practical applicability and efficiency, marking a significant advancement in FinTech software testing.

*Xiaohong Chen is the corresponding author, xhchen@sei.ecnu.edu.cn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ISSTA '24, September 16–20, 2024, Vienna, Austria
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0612-7/24/09
https://doi.org/10.1145/3650212.3680388

CCS Concepts

• **Software and its engineering** → **Requirements analysis**; **Software testing and debugging**.

Keywords

Software acceptance testing, test case generation, fintech software, large language model

ACM Reference Format:
Zhiyi Xue, Liangguo Li, Senyue Tian, Xiaohong Chen, Pingping Li, Liangyu Chen, Tingting Jiang, and Min Zhang. 2024. LLM4Fin: Fully Automating LLM-Powered Test Case Generation for FinTech Software Acceptance Testing. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), September 16–20, 2024, Vienna, Austria.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3650212.3680388

1 Introduction

FinTech (Finance Technology) software is indispensable in the financial industry, especially for stock trading, requiring a delicate balance between safety and rapid time-to-market. Before market deployment, these applications must undergo Regulations/Compliance Acceptance Testing (RAT)[18, 31]. This involves a black-box testing procedure to ascertain if the target software meets specified acceptance criteria, typically encompassing regulations, business rules, or business requirements and standards [1, 11]. However, the manual creation of valid test cases from lengthy, language-intensive documents poses a daunting challenge for testing engineers. Given the urgency of market releases and the dynamic nature of rules, automation emerges as an appealing solution [50].

Automating the acceptance testing of FinTech software is a formidable task due to the inherent need for substantial human intelligence. Extracting and comprehending business rules, often described in unstructured natural languages, and crafting comprehensive test cases require a level of understanding that current automated systems struggle to achieve [6]. Human intelligence is indispensable in the initial phases of interpreting rules, navigating through unstructured natural language and domain-specific

terminologies, identifying testable rules, and subsequently composing thorough testing scenarios based on domain knowledge. This complex process further extends to the strategic composition of test cases, emphasizing the essential role of human intelligence in ensuring the effectiveness of the testing process [16, 42].

The emergence of Large Language Models (LLMs) has revolutionized labor-intensive tasks in software engineering, excelling in code generation [17, 27, 48], collaborative high-coverage test case generation with search-based software testing (SBST) [24], and bug reproduction [20]. LLMs, with their profound understanding of natural language, generate coherent and contextually relevant responses, presenting significant advantages, especially when working collaboratively with engineers. They produce almost-ready-to-use content, minimizing the need for substantial human effort in error correction. However, this collaborative capability heavily relies on users' prompting skills and domain knowledge [28, 58].

Despite the prowess of LLMs, Soman and G [43] argue that fine-tuning is inevitable for performance improvements, particularly for domain-specific questions. We conducted a focus group interview with eight senior testing engineers on their experience of using LLMs for testing. We identified three major limitations when using LLMs to generate test cases from business rules described in unstructured natural languages for domain-specific software testing. These limitations include high intellectual demand for composing prompts, uncontrollable and intractable outputs, and limited domain knowledge in LLMs. These constraints pose a central technical challenge: *while LLMs excel at interpreting natural language business rules, generating tractable test cases from unstructured natural language business rules using them remains challenging*. According to the latest survey [49], there is still no research on the use of LLMs in acceptance testing.

Building on the observations mentioned earlier, this paper introduces a novel approach for fully automatic test case generation in FinTech software acceptance testing, leveraging the capabilities of LLMs. Our approach involves fine-tuning two LLMs: one for filtering testable business rules and the other for transforming these rules into formal ones. These formal rules are then algorithmically assembled into testable scenarios based on domain knowledge. Finally, test cases are generated using strategies such as boundary value [37] and equivalence partitioning [54], employing Satisfiability Modulo Theories (SMT)-based constraint-solving algorithms.

Our approach uniquely combines intelligent and algorithmic methods throughout the process, seamlessly integrating versatile LLMs with predictable algorithms. A key insight is that, to fully automate the generation of high-coverage test cases using LLMs, the assistance of predictable algorithms is essential to ensure tractability. The limitations of LLMs can be effectively complemented by algorithms, which are inherently terminating, deterministic, and tractable. This synergistic approach addresses the challenges associated with LLMs, enhancing the efficiency and reliability of automated test case generation for FinTech software.

To showcase the effectiveness of our approach, we have implemented a prototype LLM4Fin and applied it to a real-world stock trading system. Our comprehensive evaluation involves comparing LLM4Fin with end-to-end models like ChatGPT and human testers including three seasoned engineers boasting 5-7 years of extensive experience in testing. The experimental results reveal that LLM4Fin outperforms both ChatGPT and the experienced senior engineers. It achieves remarkable performance, with up to 98.18% and an average of 20% – 110% improvement on business scenario coverage, and up to 93.72% on code coverage, while reducing the time cost from more than 20 minutes to a mere 7 seconds. These findings emphasize the efficiency and efficacy of our approach for automatic test case generation in domain-specific regulation testing. Our tool is available at https://github.com/13luoyu/intelligent-test.

In summary, our work makes three major contributions:

1. We introduce the first fully automated LLM-powered approach, seamlessly integrating artificial intelligence and algorithmic methods to generate high-coverage test cases from business rules expressed in natural language for FinTech software acceptance testing.
2. We implement the proposed approach in a prototype, named LLM4Fin, for a real-world stock trading system. To the best of our knowledge, this marks the first industry-level fully automatic LLM-powered testing application.
3. We conduct an extensive field study to evaluate LLM4Fin, showcasing its significant advancements with up to 98.18% business scenario coverage in about 7 seconds on generating test cases.

2 Motivation

LLMs such as ChatGPT [38] have been increasingly used in testing and other software development endeavors. However, people have gradually realized that there are multiple difficulties and inconveniences to using universal LLMs to generate high-quality test cases. We organized a focus group meeting to learn the difficulties of using LLMs in testing and related activities. We invited software development professionals who have extensive testing experience using LLMs and other tools to share their experiences during the meeting. We first summarized common concerns that emerged from the meeting and then further illustrated them through an example.

2.1 Focus Group Meeting

2.1.1 The Process. The focus group meeting had 8 participants. Two researchers served as moderators. We use FG1 to FG6 to represent the focus group meeting participants. The moderators first introduced the aim and format of the focus group meeting, then gave participants time to share their own experiences and discuss.

FG1 and FG2 were from the technical section of a company focusing on financial services. They raised three concerns. First, they said that the output test cases often did not follow the desired format or routines in the financial service industry. Second, the generated test cases are not rich enough. LLMs usually generate test cases without using known testing strategies such as boundary values. Third, sometimes the generated test cases were merely straightforward expressions of the inputs. FG3 echoed that the output format was often out of control even when the model was explicitly directed to produce specific forms of output. In addition, the information in the model was relatively complex. Sometimes it contained a lot of extra unnecessary information in test cases while not containing the desired information.

FG4 and FG5 said that using LLMs in testing was very demanding to users. Since natural language was ambiguous, it was difficult

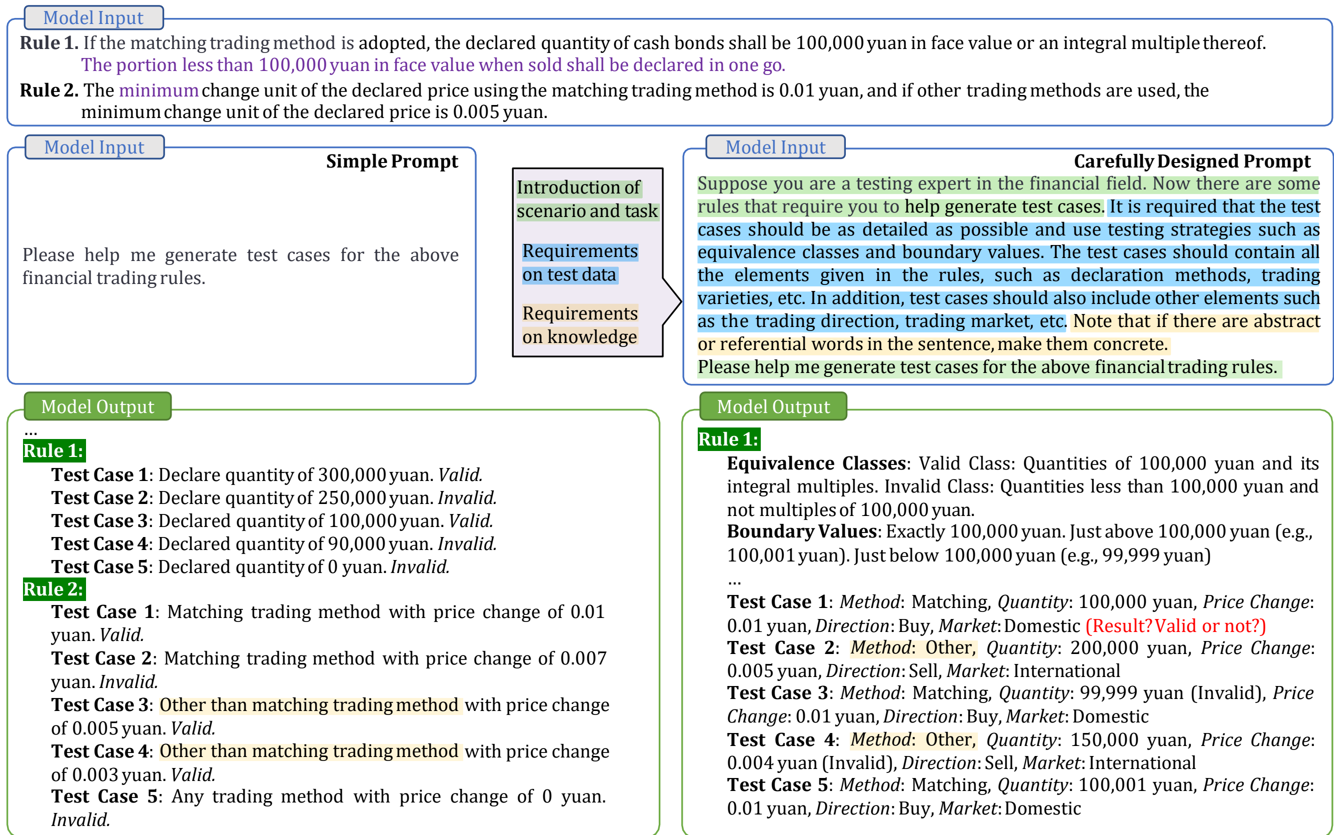

Figure 1: Using ChatGPT (GPT-4) [38] with simple and carefully designed prompts to generate test cases from financial business rules. Purple-highlighted rules in the input are those that do not produce any test cases. Yellow-highlighted test cases deviate from the prompts (also in yellow).

for LLMs to understand our requirements. To use an LLM well meant that the user had to write well-designed prompts, which required the user to read documents and keep trying. The learning curve was quite steep. FG6 offered his view from a knowledge perspective. LLMs contained rich knowledge, but we did not know how to prompt them to use their knowledge to complete tasks. Moreover, some knowledge was not available to these models, and it tended to fabricate some answers, which further aggravated the uncertainty in using these models.

2.1.2 The Outcomes. Based on participants' narratives in the focus group meeting, we summarized three common problems that people in different fields encounter when using LLMs for testing, as follows:

(1) **Intellectually-demanding.** People who want to make use of LLMs effectively in tasks such as testing should master both specific knowledge and prompt skills.
(2) **Uncontrollable and intractable outputs.** The outputs of LLMs are uncontrollable and intractable, making it difficult to reproduce and manage generated test cases.
(3) **Limited domain-knowledge.** The models have limited fintech domain knowledge and blind spots, depending on the training data and the knowledge fed into them.

2.2 A Concrete Example

To highlight the challenges with LLMs, we conducted experiments using ChatGPT (GPT-4) as an illustrative example, exploring its performance in generating test cases for financial business rules described in natural language [38].

For Problem 1, we randomly selected business rules related to trading quantity and price. We designed two prompts to assess ChatGPT's responsiveness. The first prompt was a straightforward request to generate test cases, while the second prompt was meticulously crafted, incorporating information about task participants, test case requirements, testing strategies, and other relevant details. The carefully designed prompt aimed to optimize ChatGPT's outputs. The rules, prompts, and outputs are detailed in Figure 1.

In response to Prompt 1, ChatGPT focused solely on the test points related to declaration quantity in Rule 1 and price changes in Rule 2, generating both successful and unsuccessful test cases. However, with Prompt 2, ChatGPT expanded its focus to include additional test points such as trading markets and trading methods. It demonstrated the ability to employ diverse testing strategies, resulting in a more extensive set of test cases.

This experimentation underscores the significance of well-crafted prompts in utilizing ChatGPT effectively. Notably, effective prompts

often require professionals with domain knowledge to iteratively refine them, exemplified by the success of Prompt 2 above.

In the response to Rule 1, we observe Problem 2 in both prompts. ChatGPT generated test cases for only the first half of the sentence, considering both integer multiples of 100,000 yuan and non-integer multiples, while neglecting the second half of the sentence. Even with explicit reminders about the overlooked rule and clarification that it pertained to "*The port less than 100,000 yuan in face value when solid shall be declared in one go*," ChatGPT struggled to generate valid test cases for this rule. This lack of control over the model's output represents a significant challenge.

For Rule 2, we encounter Problem 3 in both prompts. The objective of Rule 2 is to assess the model's knowledge and ability to enumerate *other trading methods* beyond the matching trading. Despite using Prompt 2 and explicitly instructing the model to specify abstract expressions like *other*, the model appears to lack knowledge in this domain. Consequently, the generated test cases still include only matching tradings and vaguely refer to "other trading methods." The model does not possess the necessary information to specify other trading methods, such as click tradings, bidding tradings, and negotiation tradings. To assist LLMs in this task, users have to provide such domain-specific knowledge by prompting. However, such interaction reduces the automation and increases extra burdens to human users.

In addition to these technical challenges, non-technical obstacles hinder the application of LLMs in fields like finance. Strict permissions and data security regulations within financial institutions mandate the isolation of sensitive information, limiting the online use of general-purpose LLMs.

Overcoming these challenges requires a systematic approach that combines versatile yet intractable LLMs with traditional algorithms to produce deterministic, reproducible, and high-quality test cases fully automatically from unstructured documents.

3 LLM4Fin

3.1 The Architecture in a Nutshell

LLM4Fin comprises three fundamental steps: *LLM-powered rule extraction*, *knowledge-guided test scenario generation*, and *test data generation*, delineated in Figure 2. The primary tasks for each step are elucidated below. LLM4Fin takes a business rule document described in natural language as input and yields a suite of test cases designed to cover all scenarios defined by the rules.

Step I - LLM-Powered Rule Extraction: LLM4Fin filters non-testable rules, extracts testable rules from the document, and transforms them into formal business rules. AI models are employed for testable rule classification and extraction, ensuring the automated identification of rules suitable for testing.

Step II - Knowledge-Guided Test Scenario Generation: Formal rules generated in Step I are interpreted, and their relations are mined under domain knowledge guidance to create *test scenarios*. Test scenarios encapsulate all necessary information for generating a test case and are derived through an interpretation process guided by domain expertise.

Step III - Test Data Generation: Test cases are generated from assembled test scenarios using data enumeration and constraint solving. Test generation strategies, such as boundary value [37] and equivalence partitioning [54], are applied to ensure test coverage.

The entire process is fully automated, eliminating the need for human intervention. This three-step workflow provides a systematic and automated approach for generating test cases from unstructured business rules, showcasing the potential of LLMs in collaboration with domain knowledge for efficient and accurate FinTech software testing.

3.2 Step I: Rule Extraction

Step I consists of three tasks, i.e., *Rule Filtering*, *Rule Element Extraction*, and *Formal Rule Assembly*. By rule filtering, those rules that do not need testing are filtered out by a pre-trained LLM which is fine-tuned for this specific classification task. The rules that are classified to be testable are then fed into another LLM that is finedtuned for element extraction. The extracted elements are assembled algorithmically to be formal rules, following a concrete syntax.

3.2.1 *Rule Filtering.*

There are basically three kinds of sentences in a business rule document. One kind of sentence describes testable business rules that software systems are expected to comply with. The other two kinds of sentences can be definitions of terminologies that are considered knowledge, or explanations of background and other information, which all do not need to be tested. The rules in Example 1 are two typical instances. Apparently, **Rule 1** should be tested, while **Rule 2** cannot. Therefore, we first need to filter out those rules that cannot be tested in a document.

> **Example 1: Testable and Untestable Rules.**
>
> ***Rule 1***: *For trades conducted through the click trading method, the declared quantity should be 100,000 yuan or its multiples.*
> ***Rule 2***: *Click trading is a trading method in which the quoting party submits a quote, and the receiving party clicks on the quote, upon which the trading system confirms the transaction or matches it automatically based on the relevant rules specified in this regulation.*

Rule filtering is essentially a classic sentence classification task and thus can be achieved using classification models in NLP. We achieve this task by constructing a dedicated corpus, where business rules are manually annotated according to their testability, and finetuning a pre-trained LLM. Corpus construction and LLM fine-tuning are going to be detailed in Section 4.

3.2.2 *Rule Element Extraction.*

The objective of this task is to extract the basic elements from testable sentences. The extracted elements are the principal ingredients for defining formal business rules. For instance, **Rule 1** contains two elements about the trading method and declared quantity regulations on this type of trading. The extracted elements can be represented as key-value pairs, as shown in Example 2.

> **Example 2: Extracted Information from Rule 1.**
>
> ***Trading Method***: *click trading method*
> ***Declared Quantity:*** *100,000 yuan or its multiples*

Element extraction can be naturally achieved as an NLP task known as Named Entity Recognition (NER) [25]. Named entities in unstructured texts are classified into corresponding pre-defined

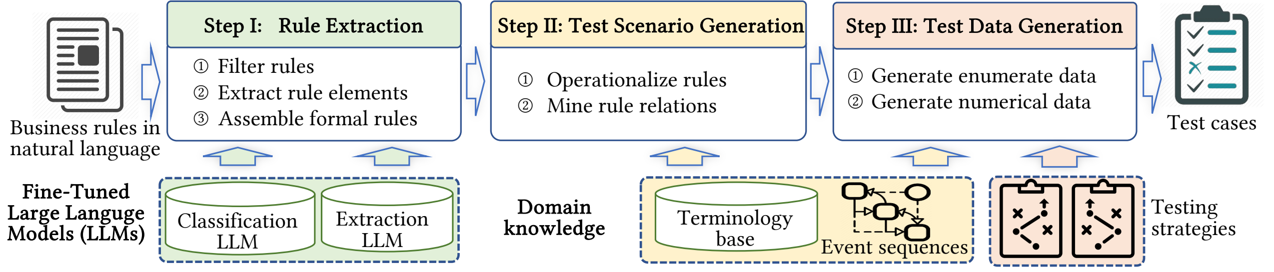

Figure 2: The three-step workflow of LLM4Fin for incorporating LLMs with algorithms for test case generation.

categories. NER is essential for various applications, including information retrieval [4], question answering systems [34], and sentiment analysis [8]. We leverage this technique to assemble formal business rules from unstructured texts. Recent studies [51] show that LLMs exhibit a greater ability in the low-resource and few-shot setups for NER. To enhance its accuracy, we devise a key-value annotation method to construct a dedicated corpus and fine-tune a pre-trained LLM on it. The details will be presented in Section 4.

3.2.3 Formal Business Rule Assembling. Utilizing extracted elements from business rules, we construct structured formal rules adhering to the syntax outlined by Knauf et al. [21]. Formal business rules take the form of `if-then` statements, incorporating conjunctive and/or disjunctive conditions regarding an attribute's value in the condition part. Additionally, these rules include an assignment of such a value to an attribute in the conclusion part. This structured representation ensures clarity and consistency in expressing the logic encapsulated within the business rules.

Each formal rule consists of several basic units which are called *clauses*. For example, "*trading method is click trading*" is a clause, stating that the trading method of interest in the rule is *click trading*. The clause is a pair of a label "Trading Method" and a value "click trading". Clauses can be connected together logically with logical operators such as "`and`" and "`or`", as shown in Example 3.

Example 3: Formal Representation for Rule 1.

```
if Trading Method is "click trading" and
   Declared Quantity is "100,000 yuan or its multiples"
then Result is "success"
```

Algorithm 1 outlines the process of assembling formal rules from extracted elements. It iterates through each label-element pair, composing them into clauses (Lines 2-3). The algorithm addresses two special cases and one common case during this task. Typically, it forms a clause of the form "label is 'element' " and saves it in a clause array (Lines 11-13). When encountering a duplicate label "Operator", the algorithm sets the label of the patient as "Operational Target" and composes the corresponding clause (Lines 4-6). In the case of a duplicate label "Operation", it merges the two operations and updates the operation clause (Lines 7-10). The generated clauses are then connected with "and" after "if" and "then" to construct formal business rules (Line 14). If a rule has conflicting labels, the algorithm divides it into a set of non-conflicting rules (Lines 15-16). Ultimately, the algorithm returns the assembled business rules.

This algorithm's handling of special cases in composing clauses enhances the clarity and conciseness of rules, optimizing comprehension and analysis of interconnected operational activities.

Algorithm 1: Formal Business Rule Assembly

Input : E: extracted rule elements; L: *element* labels.
Output: $BRs = [BR_1, BR_2, \cdots, BR_n]$: the assembled BRs.

```
1  C ← [], BRs ← [] ;  // Initialize clause array C and formal rule array BRs
2  for each element e_i in E do
3      l_i ← GetLabel(L, e_i);
4      if l_i == "Operator" and "Operator" ∈ C then        // multiple operator
5          c_i ← "Operational Target is 'e_i' ";
6          C ← Append(C, c_i) ;                // Add the clause to the end of C
7      else if l_i == "Operation" and "Operation" ∈ C then // multiple operation
8          c_0 ← GetClauseByLabel(C, l_i);
9          c_i ← MergeOperation(c_0, e_i);
10         C ← UpdateClauseByLabel(C, l_i, c_i);
11     else
12         c_i ← "l_i is 'e_i' ";
13         C ← Append(C, c_i);
14 BR_i ← ComposeClauses(C);                           // Compose a formal rule
15 if Conflict(BR_i) then                           // Cope with conflicting labels
16     BRs ← Divide(BR_i);              // Divide into a set of non-conflict subrules.
17 else
18     BRs ← Append(BRs, BR_i);
19 return BRs;
```

Additionally, the conflict detection and resolution process ensures the accuracy and feasibility of the resulting business rules.

3.3 Step II: Test Scenario Generation

Structured formal rules often capture specific aspects of a transaction scenario. To facilitate the generation of test cases for a given scenario, it becomes essential to establish connections among all rules related to that scenario and concretize them. We refer to a comprehensive sequence of concretized formal rules as a *test scenario*. This ensures that the rules collectively represent a cohesive and complete description of the transaction scenario, laying the groundwork for effective and comprehensive test case generation.

3.3.1 Domain Knowledge Representation. The generation of test scenarios relies heavily on domain knowledge. To systematically incorporate this knowledge into the process, a formal representation of domain knowledge is essential. This representation will then be seamlessly integrated into the algorithms responsible for mining relationships among formal rules.

We categorize domain knowledge into three distinct types. The first type signifies the *is-a* relation, elucidating hierarchical connections between categories, where one category serves as a subtype of another. For instance, in the statement "Bidding trading is a trading method in which the seller sells bonds to the single or multiple bidders with the best bid", bidding trading is categorized as a subtype of trading method. The second type embodies the *has-a* relation, denoting that an entity or object possesses or encompasses another entity as one of its components or attributes. In

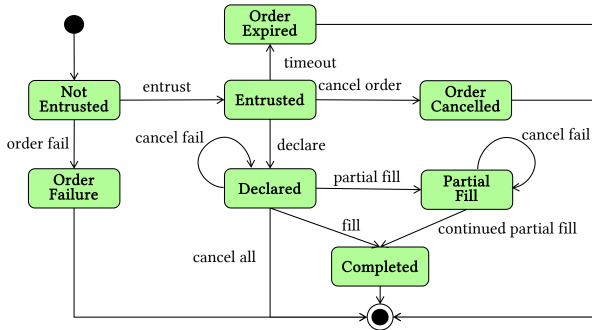

Figure 3: A state diagram for bond trading.

the statement "The elements of Price-limited Order shall include the securities account number, securities code, trading direction, quantity, price, etc.", the securities account number, securities code, etc, are considered components of the elements of Price-limited Order. The third category involves *event dependency*, capturing the logical interdependency of events and behaviors. An illustrative example is presented in Figure 3, where each node represents a state, and the edge symbolizes an operation. These three types of domain knowledge are represented in JSON format and stored in a terminology base. The acquisition of this knowledge is effectively achieved through a combination of document extraction and judicious manual supplementation. The rules classified as domain knowledge in Section 3.2.1 will be incorporated into the base.

By formalizing domain knowledge, we enhance the algorithms' capability to interpret and connect formal rules in a manner consistent with the domain's intricacies. This incorporation ensures that the generated test scenarios not only adhere to the specified formal rules but also align with the nuanced understanding provided by domain expertise. The synergy between formal representation and algorithmic processing strengthens the overall effectiveness and accuracy of the test scenario generation process.

3.3.2 Rule Operationalization. This sub-step primarily focuses on concretizing abstract expressions. Abstract expressions refer to business rules that cannot be directly operated or utilized as test input but need to be comprehended and converted manually before application, such as references, complex terms, etc. If left unaddressed, these expressions can pose challenges in the testing process.

To tackle this issue, we leverage domain knowledge and the context of related rules to provide specific alternatives for abstract representations within formal business rules. The process begins by extracting all the values from the domain knowledge base corresponding to the label associated with the abstract clause. We then identify the values that appear in the context of the same rule, excluding those previously mentioned. Subsequently, we instantiate abstract expressions with suitable concrete values, resulting in a new set of operational rules. This approach ensures that abstract expressions are replaced with practical, contextually relevant values, facilitating seamless integration into the testing workflow.

For instance, consider **Rule 1** in Example 4, which is operational, while **Rule 3** is not. The hindrance lies in the undefined nature of "other trading methods" in this context. To render the rule operational entails retrieving corresponding instances and assigning them to the undefined labels. Example 4 exemplifies the impact of operationalization by transforming **Rule 3** into four operational rules. The trading method is instantiated by four distinct trading types based on **Knowledge 1**. This operationalization process ensures the rules applicable within the defined context.

Example 4: Operationlization of Formal Rules.

Knowledge 1: *There are five bond trading methods: i.e., matching trading, click trading, inquiry trading, negotiation trading, and bidding trading.*

Rule 1:

```
if Trading Method is "click_trading" and
   Declared Quantity is "100,000_yuan_or_its_multiples"
then Result is "success"
```

Rule 3:

```
if Trading Method is "other_trading_method" and
   Declared Quantity is "1000_yuan_or_its_multiples"
then Result is "success"
```

After operationalization

Rule 1:

```
if Trading Method is "click_trading" and
   Declared Quantity is "100,000_yuan_or_its_multiples"
then Result is "success"
```

Rule 3.1:

```
if Trading Method is "matching_trading" and
   Declared Quantity is "1000_yuan_or_its_multiples"
then Result is "success"
```

Rule 3.2:

```
if Trading Method is "inquiry_trading" and
   Declared Quantity is "1000_yuan_or_its_multiples"
then Result is "success"
...
```

3.3.3 Relation Mining. This sub-step involves mining dependency relations among different rules to construct test scenarios. A test scenario comprises a sequence of continuous operations with strict sequential relationships between them, and each operation is constrained by several rules. For testing, a system needs to start from an initial state and reach a terminal state after a series of operations. To test a specific rule, the tester needs to execute all preceding rules. This implies that some rules should not be tested separately due to temporal dependencies among them. This feature highlights the necessity for the test scenario to account for rule dependencies.

We employ two methods to mine relations among rules: *explicit extraction using preconditions and postconditions* and *utilizing state diagrams*. For rules with explicit temporal prepositions such as "before", "after", and "until", we extract the temporal relations by transferring postconditions into rules to be executed after the original rule and vice versa for preconditions.

For the rules where temporal relations are not explicit, we employ state diagrams to address this challenge. In a state diagram, each state transition is triggered by an operation. When comparing two rules with the operations in a state diagram, aligned operations indicate identical sequences. For example, consider the scenario illustrated in Figure 4, where two rules are matched with the operations "declare" and "cancel". By referencing the bond trading state machine (Figure 3), we can infer that canceling a declaration must follow the declaration. Leveraging this temporal relationship among operations and analyzing the corresponding state diagram allows us to accurately determine the correct sequence for executing these rules, optimizing the system's functionality and behavior.

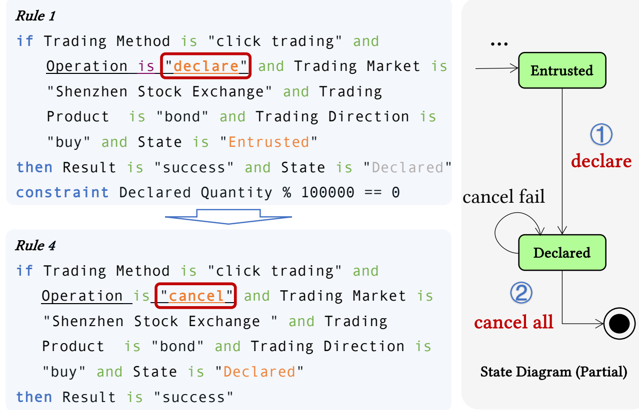

Figure 4: Example: relation mining among rules based on event sequences.

3.4 Step III: Test Data Generation

Once a test scenario is established, algorithmic generation of high-coverage test cases becomes feasible. Formally, a test case serves as a specification encompassing inputs, execution conditions, testing procedures, and expected results, all directed toward accomplishing a specific software objective [1].

Our test case generation approach, outlined in Algorithm 2, systematically processes each formal rule by identifying the type of constraint for each clause (Lines 1-4). If a clause entails an enumeration constraint, the algorithm retrieves all enumeration values for the label from the terminological base, utilizing them as the generated test data (Lines 6-8). Alternatively, if the clause involves a numerical constraint, the algorithm encodes the clause into SMT constraints using diverse testing strategies from the terminological base, such as boundary value and equivalence class. It then employs the Z3 SMT solver [7] to solve these constraints, obtaining corresponding test data (Lines 9-12).

Algorithm 2: Test Case Generation from Formal Rules.

Input : *BRs*: test scenarios; *TB*: the terminological base; *TS*: the test strategy.
Output : *TC*: generated test cases.

```
1  TC ← [];                                   // Initialize TC to be empty
2  for each business rule BR_i in BRs do
3      TD ← {} ;                  // Initialize TD to be empty for each label
4      for each constraint clause c_i in BR_i do
5          l_i ← getLabelFromClause(c_i);
6          if isEnumerateConstraint(c_i) then
7              V ← getEnumerateValue(TB, l_i) ;     // Enumerate values of l_i
8              TD[l_i] ← V;
9          else
10             φ ← getSMTConstraint(c_i, TS);// Encode into SMT constraints
11             V ← solveSMTConstraint(φ);// Solve φ by calling SMT solvers
12             TD[l_i] ← V;
13     TC_i ← CartesianProduct(TD);
14     TC ← Append(TC, TC_i);
15 return TC;
```

Finally, the algorithm computes the Cartesian product of the test data generated for each label to form the final test cases (Lines 13-15). Example 5 illustrates the generated test cases for **Rule 1**.

Example 5: Generated test cases for Rule 1.

	Market	*Product*	*Method*	*Direction*	*Quantity*	*Result*
Case 1	*Shenzhen*	*Bond*	*Click trading*	*Buy*	*100,000*	*Success*
Case 2	*Shenzhen*	*Bond*	*Click trading*	*Buy*	*200,000*	*Success*
Case 3	*Shenzhen*	*Bond*	*Click trading*	*Buy*	*50,000*	*Fail*
Case 4	*Shenzhen*	*Bond*	*Click trading*	*Buy*	*100,001*	*Fail*
Case 5	*Shenzhen*	*Bond*	*Click trading*	*Sell*	*200,000*	*Success*
Case 6	*Shenzhen*	*Bond*	*Matching trading*	*Sell*	*200,000*	*Success*
Case 7	*Shenzhen*	*Stock*	*Click trading*	*Buy*	*200,000*	*Fail*
...	...	...	...	...	...	...

4 LLM Fine-Tuning

In our approach, pre-trained general-purpose LLMs need to be fine-tuned for high accuracy in the tasks of rule filtering and element extraction. The routine of fine-tuning an LLM is standard, basically consisting of corpus construction and model training.

4.1 Corpus Construction

Due to the lack of publicly available training datasets, we need to construct the corpus for training first. We create two corpora for fine-tuning pre-trained LLMs, one for rule filtering and the other for element extraction, accessible at [56].

For the rule filtering task, we compile 18 business rule documents from financial authorities like the Shenzhen Stock Exchange. After spending 10 man-weeks annotating each rule as testable, untestable, or domain knowledge, we acquire 3,334 annotated rules for model training. The element extraction corpus is more intricate to ensure high accuracy in fine-tuning. Given the diverse and free form of rules in unstructured languages, annotation requires both consistency and flexibility. Consistency facilitates formal rule assembly, while flexibility allows extensive element annotation, enhancing fine-tuned model accuracy. To balance both, we introduce meta-model and key-value-based methods for document annotation.

4.1.1 A Meta-Model of FinTech Business Rule Documents. We conduct a systematic analysis of business rule documents to formulate a meta-model that comprehensively captures both syntactical and semantic aspects of unstructured rules. The meta-model, illustrated in Figure 5, divides each rule into two fundamental components: *Condition* and *Consequence*. These components can be linked through logical connectors like "and" or "or". Within each rule, *Condition* and *Consequence* are further subdivided into multiple *ConstraintClauses*.

Each *ConstraintClause* falls into one of six categories: *TimeConstraint*, *PriceConstraint*, *QuantityConstraint*, *TradingMethodConstraint*, *TradingResultConstraint*, or *TradingOperationConstraint*. The first three types focus on constraints related to time, price, and quantity, respectively. For instance, a *TimeConstraint* might be specified as "*9:00 to 11:30 on each trading day*." The *TradingOperationConstraint* can be subdivided into three components: *Action*, *Operator*, and *Operational Object*, based on the syntactic structure within a constraint. Additionally, the *TradingResultConstraint* involves constraints associated with the outcomes of trading operations, covering aspects such as success and failure.

In alignment with the meta-model, we crafted a set of 9 labels for the systematic annotation of corresponding arguments within the rules. These labels encompass crucial elements such as *Trading Product*, *Trading Method*, *Time*, *Price*, *Quantity*, *Result*, *Operator*, *Operation*, and *Operational Object*. Each label represents distinct constraint clauses and is color-coded accordingly in Figure 5.

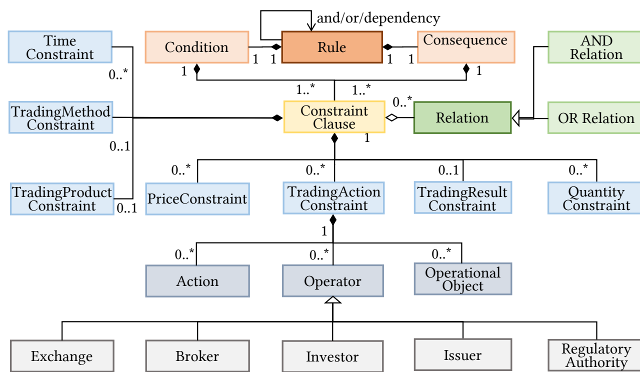

Figure 5: A meta-model for the domain knowledge in the FinTech domain. Blocks in different colors represent different types of categories.

4.1.2 The Key-Value Annotation Method. Key-Value annotation is to deal with concepts that are not referred to in the meta-model. The necessity arises from the fact that some elements only manifest themselves once in the document. They are important to test case generation but cannot be captured by the meta-model. We propose a Key-Value (K-V) annotation method to handle this problem. In our K-V method, a label (called *Key*) and one of its values usually come in pairs. The *Key* enables the annotation of the labels that fall outside the previously defined label sets in the meta-model. Conversely, the *Value* serves to annotate the instances of the label. The *Key-Value* annotation method yields numerous advantages, notably fostering more succinct labeling, maintaining a consistent format, and ultimately reducing the error rate during the annotation process. This method optimizes the annotation workflow, ensuring greater accuracy and facilitating a more streamlined and efficient analysis of the rule documents.

Example 6 showcases the annotated text of **Rule 1** using our annotation approach. The labels of *Trading Method* and *Quantity* are derived from the meta-model, while *Key* is introduced by the Key-Value approach for the new concept of "*declared quantity*".

> **Example 6**: Text Annotation for LLM Fine-Tuning.
>
> **Annotated Rule 1:** For trades conducted through the [click trading]*(Trading Method)* method, the [declared quantity]*(Key)* should be [100,000 yuan or its multiples]*(Quantity)*;

We employ the annotation tool POTATO [40] to thoroughly annotate all sentences in the 18 documents issued by stock authorities, requiring over 8 man-months. This corpus comprises a total of 1,331 sentences, with each sentence having an average of over 5 annotations. Importantly, domain experts have meticulously verified the annotations in our corpus, ensuring their accuracy and reliability.

4.2 Model Selection and Training

We fine-tune the pre-trained Chinese language model Mengzi [60] on our constructed corpora to perform our rule filtering and extraction tasks. Mengzi is a series of lightweight yet powerful models whose backbone model is RoBERTa [29]. As we focus on the financial domain, we choose Mengzi-BERT-base-fin as the base model, which comprises 12 transformer layers and a substantial of $103M$ parameters and has been trained on over $20G$ financial materials.

To train the rule filtering model, the constructed corpus is split into a training dataset and a validation dataset in a 9:1 ratio. The training dataset is augmented using the method in [52], resulting in sizes of 32,945 for training and 333 for validation. The input of the model is a sentence, and the output is a number representing the corresponding class. During training, we use the cross-entropy loss function. With a batch size set to 8, the model is trained for 50 epochs, employing the AdamW [30] as the optimizer. Initially, the learning rate is set to $1e-5$ and linearly decreases to 0 after a 5-epoch warm-up [12]. Our fine-tuned model demonstrates robust performance, achieving up to 99.1% accuracy in classifying rules within the validation dataset.

To train the rule element extraction model, we divided the corpus into training and validation datasets in a 9:1 ratio and applied two data augmentation techniques [2, 52] to the training dataset. The resulting datasets contain 41,215 instances for training and 634 for testing. The input of the model is a sentence, and the output is a sequence of numbers corresponding to the class of each input token. The training setup is similar to the classification model, with the learning rate set to $2e-5$, weight decay to 0.002, training epochs to 20, and warmup epochs to 2. After training, our fine-tuned model achieves an accuracy of up to 87.0%. Both the fine-tuned rule filtering model and rule extraction model are available at [55].

The hyper-parameter settings above are those that achieve the highest accuracy on the validation datasets, respectively. When applied to other domains or rule sets, the hyper-parameter settings may need to be adjusted based on factors like the size of the training dataset and the features of the data, while the model structure and fine-tuning process remain unchanged.

5 Experimental Evaluation

To assess the effectiveness and efficiency of LLM4Fin, we conducted comparisons with both expert and non-expert testing engineers, as well as general-purpose LLMs, focusing on the quality of generated (or manually composed) test cases and the time cost. Besides, we also evaluate the impact of the performance of the fine-tuned LLMs on the overall framework.

Metrics. We assess the quality of test cases using two metrics: Business Scenario Coverage (BSC) [19, 59] and Code Coverage (CC) [22]. Business scenarios are modeled as a tree structure, where nodes represent constraints and edges are values corresponding to the constraints. For example, in stock trading, a node might denote a "Trading Method" with edges like "bidding trading" and "block trading". The final leaf node signifies the consequence of business execution, encompassing success and failure. BSC is the ratio of triggered business scenarios to the total number of scenarios, calculated as $BSC = p_t/p_a$, where p_t is the number of triggered scenarios and p_a is the total number. A higher BSC indicates broader coverage of generated test cases from the perspective of software requirements. CC measures the coverage of generated test cases from the source code side. We use widely adopted metrics such as Statement Block Coverage (SBC) [36] and Modified Condition/Decision Coverage (MC/DC) [14] to evaluate the code coverage. Additionally, we consider the time cost of generating or composing test cases as a metric for assessing efficiency.

Competitors. We compare our tool with human testers, including senior testing engineers in the FinTech domain and non-expert graduate students. All the expert testers and non-expert testers work individually. Additionally, we consider two language models, the general-purpose LLM ChatGPT [38], and a state-of-the-art LLM called ChatGLM [9], specifically designed for the Chinese corpus. Table 1 shows the detailed information about these competitors.

Table 1: Details of Competitors.

Competitor	Description
Expert testers	Three senior testing engineers with 5-7 years of experience in FinTech software centers.
Non-expert testers	Four graduate students with approximately one year of software testing knowledge acquired from courses.
ChatGPT [38]	GPT-4, including 1800 billion parameters.
ChatGLM [9]	A LLM fine-tuned in Chinese with 130 billion parameters, leading the c-eval [15] leaderboard.

Datasets. As the target software is commercial and domain-specific, there are no publicly available benchmarks. To ensure fairness, we randomly selected five functionalities from five typical trading categories, with each functionality's related business rules forming a dataset. The details of these five datasets are presented in Table 2. The datasets are publicly available to facilitate reproducibility [56].

Experimental Settings. We conducted all the experiments on a workstation equipped with a 32-core AMD Ryzen Threadripper PRO 5975WX CPU, 256GB RAM, and an NVIDIA RTX 3090Ti GPU running Ubuntu 22.04.

5.1 Experiment I: Business Scenario Coverage

5.1.1 Experiment Design. We provided the five datasets to both expert and non-expert testers and requested them to generate test cases for them and record the time cost. Using the carefully crafted prompts detailed in Section 2, we requested LLMs to generate test cases for each dataset, one rule at a time, and record the time taken for the task. LLM4Fin was fed with the datasets to produce the corresponding outputs without any prompts.

The number of test cases and business scenario coverage were automatically calculated. To accomplish this, we initially constructed business scenarios for each dataset and developed a program to compute the business scenario coverage for each set of test cases. One path in a business scenario is composed of multiple nodes (constraints) connected from root to leaf. The algorithm counts the number of nodes covered by the test case at most and divides it by the number of all nodes in the path to obtain the coverage of the path. The coverage of all paths in the business scenario is computed by the average coverage of each path.

5.1.2 Experimental Results. Table 3 provides a detailed overview of the experiment's results. Notably, our approach excels in generating test cases, surpassing all comparison tools and human testers in terms of BSC. Across all five evaluation datasets, our approach achieved coverage rates exceeding 80%, with Dataset 3 reaching an impressive 98.67%. To further visualize the coverage deviations of the generated test cases, Figure 6 is presented. It illustrates that LLM4Fin exhibits nearly the same deviation as experts, both of which are lower than those of the general-purpose LLMs. The higher deviation of LLMs reflects the inherent challenge of ensuring consistent and high-quality test case generation with LLMs.

Table 2: Details of Evaluation Datasets. #Rules (BS) refers to the number of rules (business scenarios) in the dataset.

Dataset	Sub-domain	# Rules	# BS
Dataset 1	*GEM after-hours pricing trading*: trading method conducted outside regular market hours.	11	12
Dataset 2	*Stock block trading*: buying or selling a large block of shares in a single transaction.	12	40
Dataset 3	*Fund trading*: transaction where investors contribute funds managed by fund managers.	13	37
Dataset 4	*Convertible bond trading*: trading variety with characteristics of both bonds and stocks.	8	24
Dataset 5	*Stock auction trading*: a trading method in which trades are matched by price and time priority.	13	400

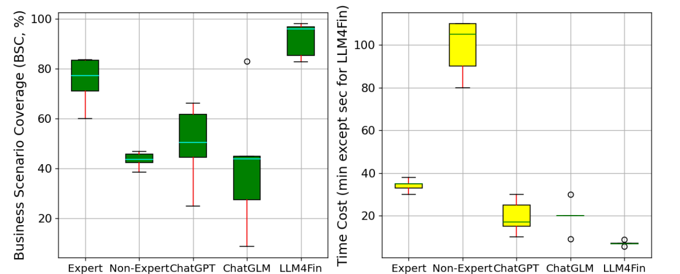

Figure 6: The BSC (left) and time cost (right) distributions and deviations with respect to the generated test cases.

In Figure 6 (right), the time cost and deviation of the five methods are illustrated, providing insights into their efficiency and stability. Notably, LLM4Fin emerges as the epitome of efficiency, showcasing consistently swift execution times and minimal variance across tasks. Following closely is ChatGPT, demonstrating moderate efficiency albeit with some inconsistency and instability. ChatGLM, while requiring more time, exhibits lower deviation than ChatGPT. Unsurprisingly, the manual composition of test cases proves to be the least efficient approach. Moreover, when compared to experts, non-experts require significantly more time, and their deviation is notably higher than both experts and automated tools.

5.2 Experiment II: Code Coverage

5.2.1 Experiment Design. The code coverage can be computed on a cloud test platform called TestStars [44], where the corresponding securities trading system code needs to be uploaded. It executes each test case on this code and records the traversed statement block or condition/decision branch. Because every test case has to be manually input into the target web-based trading system, the experiment is very labor intensive. As an example, we only evaluate on the most complex dataset 5, which has the largest number of business scenarios among all the datasets. Besides, we did not compare with ChatGLM because the BSC by its test cases are too low, i.e., 8.77%. The total number of statement blocks and condition/decisions in Dataset 5 are 1,687 and 2,418, respectively.

5.2.2 Experimental Results. Table 4 shows Statement Block Coverage (SBC) and Modified Condition/Decision Coverage (MC/DC) scores for test cases generated through different methods. Notably, LLM4Fin outshines all other approaches in both SBC and MC/DC, achieving the highest coverage scores of 93.72% and 90.86%, respectively. This demonstrates its superior effectiveness in generating

Table 3: Comparison of the number of test cases (#TC), business scenario coverage (BSC), and time consumption (Time) of generated test cases with experts, non-experts, and general LLMs on 5 evaluation datasets.

Datasets	Experts				Non-Experts				ChatGPT				ChatGLM				LLM4Fin		
	#TC	BSC (%)	Impr. (%)	Time	#TC	BSC (%)	Impr. (%)	Time	#TC	BSC (%)	Impr. (%)	Time	#TC	BSC (%)	Impr. (%)	Time	#TC	BSC (%)	Time
Dataset 1	24	83.43	15.1	33m	22	45.83	109.6	105m	24	44.61	115.3	30m	54	27.48	249.6	20m	218	**96.06**	**7.01s**
Dataset 2	50	83.80	2.5	38m	34	38.58	122.6	90m	42	50.48	70.1	25m	67	45.01	90.8	9m	672	**85.36**	**6.85s**
Dataset 3	168	71.15	38.0	30m	33	46.79	109.8	110m	32	61.81	58.8	17m	33	43.88	123.7	20m	270	**98.18**	**6.82s**
Dataset 4	30	77.34	25.4	35m	29	42.47	128.3	80m	34	25.00	287.8	10m	35	83.02	16.8	30m	88	**96.96**	**5.57s**
Dataset 5	67	60.10	38.0	35m	30	43.64	90.0	110m	41	66.35	25.0	15m	51	8.77	845.4	20m	880	**82.91**	**8.72s**
Average	68	75.16	22.3	34m	30	43.46	111.4	99m	35	49.65	85.1	19m	48	41.63	120.7	20m	426	**91.89**	**6.99s**

Table 4: Code coverage in Statement Block Coverage (SBC) and Modified Condition/Decision Coverage (MC/DC) on Dataset 5.

	Experts	Non-Experts	ChatGPT	LLM4Fin
SBC (%)	86.90 (1466)	71.01 (1198)	87.61 (1478)	**93.72 (1581)**
MC/DC (%)	82.30 (1990)	64.06 (1549)	81.76 (1977)	**90.86 (2197)**

comprehensive test cases. Experts and ChatGPT also perform admirably on code coverage, achieving comparable SBC and MC/DC scores. This result aligns with the findings in Experiment I, where experts and ChatGPT achieve comparable results, i.e., 60.10% and 66.35%, respectively. It is worth mentioning that the increase in code coverage by our tool is not as substantial as in business scenario coverage. This is primarily because the target software under testing does not consider all the business scenarios described by the business rules in Dataset 5, a fact confirmed by the software developer. Similar to the results for BSC, non-experts still lag behind in both CC metrics, compared with other approaches.

5.3 Experiment III: Impact of Back-End LLMs

In this experiment, we explore the impact of fine-tuned LLMs in our approach on the quality of generated test cases and showcase the compatibility of our framework with different back-end LLMs.

5.3.1 Experiment Design. In addition to Mengzi, we choose FinBert [26], and Llama2 (7B) [46] as base models. We choose the three LLMs because they are (i) pre-trained on Chinese corpus and have a good understanding of Chinese content, (ii) open-source and can be fine-tuned to apply to different downstream tasks, and (iii) heterogeneous where FinBert and Mengzi are BERT-based classification models, while Llama2 is a GPT-based generative model.

We fine-tune FinBert with the same method mentioned in Section 4.2. For Llama2, we fine-tune it on a question-answer dataset transformed from the training dataset used by FinBert and Mengzi, where each question includes the prompt for rule extraction and the rule, and the answer is the extracted rule elements and their labels. The accuracy of the trained FinBert, Mengzi, and Llama2 on the validation set is 56.05%, 86.84%, and 94.76%, respectively. Each fine-tuned LLM is used in the rule extraction step.

5.3.2 Experimental Results. We compute the BSC of the generated test cases using different fine-tuned LLMs on the same five datasets employed in Experiment I. The results are shown in Figure 7. We observe that the BSC remains consistently high, exceeding 80% on average, regardless of the LLMs employed. The variation in BSC across different datasets is minimal, never surpassing 8%. This suggests that our approach is compatible with and orthogonal to backend LLMs. An LLM with high accuracy in rule filtering and extraction is substitutable, ensuring our approach's transferability.

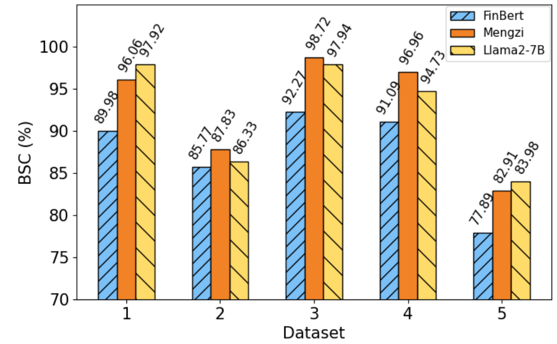

Figure 7: The BSCs of the test cases generated by LLM4Fin.

5.4 Takeaways from the Experimental Results

The experimental results yield valuable insights into the performance of LLMs in the task of generating test cases from unstructured language requirement documents:

Finding I: Crucial Role of Domain Knowledge. The findings underscore the pivotal role of domain knowledge in achieving high-coverage test case generation. Experts and LLM4Fin, both presumably equipped with domain knowledge, achieve significantly higher coverage than non-experts and ChatGPT. This reinforces the idea that domain knowledge plays a crucial role in understanding and covering complex business logic. Our approach, by leveraging domain knowledge in test case generation, surpasses even the experts. This further strengthens the argument for integrating domain knowledge into LLMs and incorporating with traditional algorithms to improve their effectiveness.

Finding II: Non-Expert Level Performance by General-Purpose LLMs. While general-purpose LLMs show promise in test case generation, their effectiveness falls short of expert human testers and our dedicated test case generation tool, as shown in Figure 6 (left). Notably, LLM performance exhibits significantly higher variation in outcome, indicating potential instability and limitations in handling complex test scenarios. These findings suggest that relying solely on general-purpose LLMs for critical software testing tasks may introduce unacceptable risks. However, the potential of LLMs in this domain remains significant. Future research should prioritize investigating the efficacy of sophisticated fine-tuning approaches and high-quality, domain-specific corpora, which hold significant potential to improve the effectiveness and stability of LLMs.

Finding III: Synergistic potential of LLMs and algorithms. The performance of LLM4Fin demonstrates the significant potential for synergy between LLMs and algorithms. Such hybrid leverages the strengths of both paradigms: LLMs excel at capturing complex domain knowledge and language nuances, while algorithms provide efficient and systematic execution. By combining these strengths, we can solve tasks that require both human-like intelligence and mechanical processing, leading to superior outcomes.

5.5 Transferability and Human Efforts

Our LLM4Fin framework is transferable to other application domains, although in this work its effectiveness is showcased only in the FinTech domain. The transferability benefits from its widely-adopted three-step workflow for test case generation, standard LLM fine-tuning and well-established test case generation strategies.

First, our three-step workflow is built on the widely adopted test case generation framework. The three-step workflow is general and has been adopted to many domains such as finance [18, 50], electronic commerce [5], and automation [10]. We follow such architecture while incorporating LLMs to automate those steps that have to be manually achieved by domain experts.

Second, fine-tuning LLMs for rule extraction is applicable to other domains. The core of rule extraction is information extraction, where fine-tuning LLMs has become a widely adopted technique for it in various domains [25, 41, 47]. Our framework incorporates the fine-tuned LLMs into the workflow, which makes it applicable and transferable to other domains, such as road rule testing.

Third, the test case generation strategies are well-established and make no assumption to application domains. They are domain-independent, and have been widely adopted to domains like finance [18, 50] and autonomous driving [23, 45].

Thanks to the transferability of LLM4Fin, it only requires minimal human efforts to adapt the framework to other application domains. The adaptation primarily involves two domain-specific tasks, i.e., fine-tuning LLMs and constructing corresponding domain knowledge bases. For the LLM fine-tuning, one can follow the steps in Section 4.2 to construct high-quality corpus by annotating unstructured texts and labeling testable and untestable sentences and to re-train LLMs until their accuracy is reasonably high, e.g., 86.84% in LLM4Fin. Another manual effort is needed to build domain knowledge bases which are necessary to generate high-coverage test scenarios. It requires human expertise to build precise and compact formal models such as state diagrams that can be efficiently accessed by algorithms.

5.6 Threats to Validity

Our study acknowledges and addresses potential threats to the validity of our work, emphasizing the importance of domain knowledge and the inherent challenges posed by natural language flexibility.

Domain Knowledge Insufficiency: The threat of inadequate domain knowledge is a potential limitation. As aforementioned, the reliance on high-quality, domain-specific corpora for LLM fine-tuning and the creation of comprehensive terminology bases demand considerable human effort. Despite this challenge, we assert that the investment in robust domain knowledge is indispensable, aligning with the recognized value of business rules as crucial assets [33].

Natural Language Ambiguity: The inherent flexibility and ambiguity of natural languages pose another potential concern. While our framework accommodates free-style expression of business rules, well-structured and organized documents enhance precision. Our approach encourages simplicity and clarity in business rule documents, essential for AI-driven methodologies [3, 13, 32, 35, 38, 39].

6 Related Work

Our work builds upon the existing body of research focused on generating test cases from business rules expressed in unstructured natural languages, as highlighted in [13]. Existing approaches often rely on heuristic methods for entity and fact extraction due to the inherent flexibility of natural languages. For instance, Meservy et al. [33] proposed a business rule modeling language and automated test sequence generation, addressing the challenge of engineers lacking expertise in composing formal rules. Willmor and Embury [53] introduced the concept of intensional database test cases containing check-conditions for testing compliance with rules.

The recognition of the importance of domain knowledge in automated test case generation is a growing trend. Yin et al. [57] proposed a domain knowledge-based test case generation approach for space telemetry systems, further emphasizing the significance of domain-specific knowledge in automated testing. Jin et al. [18] proposed FinExpert, a domain-specific test generation tool for FinTech systems that incorporates domain knowledge, including data-field dependencies, exceptional input cases, and test oracles.

The field of applying LLMs to software testing has experienced significant growth in recent times. However, the majority of this focus has centered around unit testing and system testing. According to a recent survey [49], limited research has explored the use of LLMs in acceptance testing, i.e., testing whether software aligns with business requirements, which is the primary focus of our work. While Wang et al. [49] suggested a human-in-the-loop schema with LLMs, our work demonstrates the potential for fully automating the process by incorporating LLMs with classical algorithms.

7 Concluding Remarks

We presented a fully automated methodology that seamlessly integrates LLMs and deterministic algorithms for the purpose of generating high-coverage test cases in FinTech software acceptance testing. Our approach lies in its utilization of versatile LLMs, coupled with traditional algorithms that ensure determinism and tractability. This synergistic blend proves instrumental in overcoming the inherent challenges in interpreting unstructured business rules, rendering the whole process more efficient and tractable.

Our approach also sets the groundwork for a pioneering framework for the deterministic generation of high-quality test cases from unstructured software requirement documents. The impact of our work reaches beyond the FinTech domain, potentially benefiting various domain-specific and safety-critical software applications. Further exploration into finding the optimal balance between LLMs and algorithms is a valuable avenue for future research.

Acknowledgments

This work is supported by NSFC Programs (62161146001, 62372176, 62272166), Shanghai Trusted Software Innovation Center, Huawei, and Shanghai International Joint Lab (22510750100).

References

[1] 2010. ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary. *ISO/IEC/IEEE 24765:2010(E)* (2010). https://doi.org/10.1109/IEEESTD.2010.5733835

[2] 425776024. 2020. NLPCDA. https://github.com/425776024/nlpcda. Accessed: 2023-07-19.

[3] Anthropic. 2023. Meet Claude: A next-generation AI assistant for your tasks, no matter the scale. "https://claude.ai/".

[4] Hsin-Hsi Chen, Yung-Wei Ding, and Shih-Chung Tsai. 1998. Named entity extraction for information retrieval. *Computer Processing of Oriental Languages* 12, 1 (1998), 75–85.

[5] Pavan Kumar Chittimalli, Kritika Anand, Shrishti Pradhan, Sayandeep Mitra, Chandan Prakash, Rohit Shere, and Ravindra Naik. 2019. BuRRiTo: a framework to extract, specify, verify and analyze business rules. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1190–1193. https://doi.org/10.1109/ASE.2019.00134

[6] Jean-Pierre Corriveau, Vojislav Radonjic, and Wei Shi. 2014. Requirements verification: Legal challenges in compliance testing. In *2014 IEEE International Conference on Progress in Informatics and Computing (PIC)*. IEEE, 451–454. https://doi.org/10.1109/pic.2014.6972376

[7] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

[8] Leon Derczynski, Diana Maynard, Giuseppe Rizzo, Marieke Van Erp, Genevieve Gorrell, Raphaël Troncy, Johann Petrak, and Kalina Bontcheva. 2015. Analysis of named entity recognition and linking for tweets. *Information Processing & Management* 51, 2 (2015), 32–49. https://doi.org/10.1016/J.IPM.2014.10.006

[9] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. GLM: General Language Model Pretraining with Autoregressive Blank Infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*. 320–335. https://doi.org/10.18653/V1/2022.ACL-LONG.26

[10] Shuo Feng, Yiheng Feng, Chunhui Yu, Yi Zhang, and Henry X Liu. 2020. Testing scenario library generation for connected and automated vehicles, part I: Methodology. *IEEE Transactions on Intelligent Transportation Systems* 22, 3 (2020), 1573–1582. https://doi.org/10.1109/TITS.2020.2972211

[11] Jannik Fischbach, Julian Frattini, Andreas Vogelsang, Daniel Mendez, Michael Unterkalmsteiner, Andreas Wehrle, Pablo Restrepo Henao, Parisa Yousefi, Tedi Juricic, Jeannette Radduenz, et al. 2023. Automatic creation of acceptance tests by extracting conditionals from requirements: NLP approach and case study. *Journal of Systems and Software* 197 (2023), 111549. https://doi.org/10.1016/J.JSS.2022.111549

[12] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *CoRR* (2017). http://arxiv.org/abs/1706.02677

[13] "Business Rule Group". [n. d.]. "The business rules manifesto". "https://www.businessrulesgroup.org/brmanifesto.htm"

[14] Kelly J Hayhurst and Dan S Veerhusen. 2001. A practical approach to modified condition/decision coverage. In *20th DASC. 20th Digital Avionics Systems Conference (Cat. No. 01CH37219)*, Vol. 1. 1B2/1–1B2/10 vol.1. https://doi.org/10.1109/DASC.2001.963305

[15] Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, Yao Fu, Maosong Sun, and Junxian He. 2023. C-Eval: A Multi-Level Multi-Discipline Chinese Evaluation Suite for Foundation Models. In *Advances in Neural Information Processing Systems (NeurIPS)*. http://papers.nips.cc/paper_files/paper/2023/hash/c6ec1844bec96d6d32ae95ae694e23d8-Abstract-Datasets_and_Benchmarks.html

[16] Simon Holm Jensen, Suresh Thummalapenta, Saurabh Sinha, and Satish Chandra. 2015. Test Generation from Business Rules. In *8th IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 1–10. https://doi.org/10.1109/ICST.2015.7102608

[17] Xue Jiang, Yihong Dong, Lecheng Wang, Qiwei Shang, and Ge Li. 2023. Self-planning code generation with large language model. *CoRR* (2023). https://doi.org/10.48550/ARXIV.2303.06689

[18] Tiancheng Jin, Qingshun Wang, Lihua Xu, Chunmei Pan, Liang Dou, Haifeng Qian, Liang He, and Tao Xie. 2019. FinExpert: Domain-specific test generation for FinTech systems. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 853–862. https://doi.org/10.1145/3338906.3340441

[19] Cem Kaner. 2003. On scenario testing. *Software Testing and Quality Eng. Magazine* (2003), 16–22.

[20] Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. Large Language Models are Few-shot Testers: Exploring LLM-based General Bug Reproduction. (2023), 2312–2323. https://doi.org/10.1109/ICSE48619.2023.00194

[21] Rainer Knauf, Silvie Spreeuwenberg, Rik Gerrits, and Martin Jendreck. 2004. A Step out of the Ivory Tower: Experiences with Adapting a Test Case Generation Idea to Business Rules.. In *FLAIRS Conference*. Citeseer, 343–348. http://www.aaai.org/Library/FLAIRS/2004/flairs04-061.php

[22] Pavneet Singh Kochhar, Ferdian Thung, and David Lo. 2015. Code coverage and test suite effectiveness: Empirical study with real bugs in large systems. In *2015 IEEE 22nd international conference on software analysis, evolution, and reengineering (SANER)*. IEEE, 560–564. https://doi.org/10.1109/SANER.2015.7081877

[23] Thomas Laurent, Stefan Klikovits, Paolo Arcaini, Fuyuki Ishikawa, and Anthony Ventresque. 2023. Parameter coverage for testing of autonomous driving systems under uncertainty. *ACM Transactions on Software Engineering and Methodology* 32, 3 (2023), 1–31. https://doi.org/10.1145/3550270

[24] Caroline Lemieux, Jeevana Priya Inala, Shuvendu K Lahiri, and Siddhartha Sen. 2023. Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 919–931. https://doi.org/10.1109/ICSE48619.2023.00085

[25] Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. 2020. A survey on deep learning for named entity recognition. *IEEE Transactions on Knowledge and Data Engineering* 34, 1 (2020), 50–70. https://doi.org/10.1109/TKDE.2020.2981314

[26] Yu Li and Panpan Hou. 2020. FinBert. https://github.com/valuesimplex/FinBERT.

[27] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. (2023). http://papers.nips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html

[28] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* 55, 9 (2023), 1–35. https://doi.org/10.1145/3560815

[29] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *CoRR* abs/1907.11692 (2019). http://arxiv.org/abs/1907.11692

[30] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *7th International Conference on Learning Representations (ICLR)*. OpenReview.net. https://openreview.net/forum?id=Bkg6RiCqY7

[31] Jelena Madir. 2021. *FinTech: Law and regulation*. Edward Elgar Publishing.

[32] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *6th International Conference on Learning Representations (ICLR)*. OpenReview.net. https://openreview.net/forum?id=rJzIBfZAb

[33] Thomas O Meservy, Chen Zhang, Euntae T Lee, and Jasbir Dhaliwal. 2011. The business rules approach and its effect on software testing. *IEEE software* 29, 4 (2011), 60–66. https://doi.org/10.1109/MS.2011.120

[34] Diego Mollá, Menno Van Zaanen, and Daniel Smith. 2006. Named entity recognition for question answering. In *Proceedings of the Australasian language technology workshop 2006*. Australasian Language Technology Association, 51–58. https://aclanthology.org/U06-1009/

[35] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2574–2582. https://doi.org/10.1109/CVPR.2016.282

[36] Glenford J Myers, Tom Badgett, Todd M Thomas, and Corey Sandler. 2004. *The art of software testing*. Vol. 2. Wiley Online Library. https://doi.org/10.1002/9781119202486

[37] Glenford J Myers, Corey Sandler, and Tom Badgett. 2011. *The art of software testing*. John Wiley & Sons. https://malenezi.github.io/malenezi/SE401/Books/114-the-art-of-software-testing-3-edition.pdf

[38] OpenAI. 2023. ChatGPT: get instant answers, find creative inspiration, and learn something new. "https://openai.com/chatgpt".

[39] OpenAI. 2023. Sage. "https://poe.com/Assistant".

[40] Jiaxin Pei, Aparna Ananthasubramaniam, Xingyao Wang, Naitian Zhou, Apostolos Dedeloudis, Jackson Sargent, and David Jurgens. 2022. POTATO: The Portable Text Annotation Tool. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 327–337. https://doi.org/10.18653/V1/2022.EMNLP-DEMOS.33

[41] Nadeesha Perera, Matthias Dehmer, and Frank Emmert-Streib. 2020. Named entity recognition and relation detection for biomedical information extraction. *Frontiers in cell and developmental biology* 8 (2020), 673. https://doi.org/10.3389/fcell.2020.00673

[42] Suman Roychoudhury, Sagar Sunkle, Deepali Kholkar, and Vinay Kulkarni. 2017. From natural language to SBVR model authoring using structured English for compliance checking. In *2017 IEEE 21st International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 73–78. https://doi.org/10.1109/EDOC.2017.19

[43] Sumit Soman and Ranjani H G. 2023. Observations on LLMs for Telecom Domain: Capabilities and Limitations. (2023), 36:1–36:5. https://doi.org/10.1145/3639856.3639892

[44] Ltd Suzhou Insight Cloud Information Technology Co. 2023. TestStars: A Precision Cloud Test Platform. "http://www.threadingtest.com".

[45] Shuncheng Tang, Zhenya Zhang, Yi Zhang, Jixiang Zhou, Yan Guo, Shuang Liu, Shengjian Guo, Yan-Fu Li, Lei Ma, Yinxing Xue, et al. 2023. A survey on automated driving system testing: Landscapes and trends. *ACM Transactions on Software Engineering and Methodology* 32, 5 (2023), 1–62. https://doi.org/10.1145/3579642

[46] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR* abs/2307.09288 (2023). https://doi.org/10.48550/ARXIV.2307.09288

[47] Amalie Trewartha, Nicholas Walker, Haoyan Huo, Sanghoon Lee, Kevin Cruse, John Dagdelen, Alexander Dunn, Kristin A Persson, Gerbrand Ceder, and Anubhav Jain. 2022. Quantifying the advantage of domain-specific pre-training on named entity recognition tasks in materials science. *Patterns* 3, 4 (2022). https://doi.org/10.1016/J.PATTER.2022.100488

[48] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*. ACM, 332:1–332:7. https://doi.org/10.1145/3491101.3519665

[49] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software testing with large language model: Survey, landscape, and vision. *IEEE Trans. Software Eng.* 50, 4 (2024), 911–936. https://doi.org/10.1109/TSE.2024.3368208

[50] Qingshun Wang, Lintao Gu, Minhui Xue, Lihua Xu, Wenyu Niu, Liang Dou, Liang He, and Tao Xie. 2018. FACTS: Automated black-box testing of FinTech systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 839–844. https://doi.org/10.1145/3236024.3275533

[51] Shuhe Wang, Xiaofei Sun, Xiaoya Li, Rongbin Ouyang, Fei Wu, Tianwei Zhang, Jiwei Li, and Guoyin Wang. 2023. GPT-NER: Named Entity Recognition via Large Language Models. https://doi.org/10.48550/ARXIV.2304.10428

[52] Jason W. Wei and Kai Zou. 2019. EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, 6381–6387. https://doi.org/10.18653/V1/D19-1670

[53] David Willmor and Suzanne M Embury. 2006. Testing the implementation of business rules using intensional database tests. In *Testing: Academic & Industrial Conference-Practice and Research Techniques (TAIC PART'06)*. IEEE Computer Society, 115–126. https://doi.org/10.1109/TAIC-PART.2006.28

[54] Hao Wu. 2012. An effective equivalence partitioning method to design the test case of the WEB application. In *2012 International Conference on Systems and Informatics (ICSAI2012)*. 2524–2527. https://doi.org/10.1109/ICSAI.2012.6223567

[55] Zhiyi Xue, Liangguo Li, et al. 2023. Trained Model. "https://huggingface.co/13luoyu/LLM4Fin".

[56] Zhiyi Xue, Liangguo Li, et al. 2024. Constructed Corpus and Datasets. "https://github.com/13luoyu/intelligent-test/data".

[57] Bin Yin, Xiaohong Chen, Wanyu Li, and Jinyue Tian. 2022. An Incremental Software Automation Testing for Space Telemetry, Track and Command Software Systems Based on Domain Knowledge. *J. Circuits Syst. Comput.* 31, 7 (2022), 2250133:1–2250133:24. https://doi.org/10.1142/S021812662250133X

[58] JD Zamfirescu-Pereira, Richmond Y Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, CHI 2023, Hamburg, Germany, April 23-28, 2023*. ACM, 437:1–437:21. https://doi.org/10.1145/3544548.3581388

[59] Xudong Zhang and Yan Cai. 2023. Building Critical Testing Scenarios for Autonomous Driving from Real Accidents. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 462–474. https://doi.org/10.1145/3597926.3598070

[60] Zhuosheng Zhang, Hanqing Zhang, Keming Chen, Yuhang Guo, Jingyun Hua, Yulong Wang, and Ming Zhou. 2021. Mengzi: Towards Lightweight yet Ingenious Pre-trained Models for Chinese. *CoRR* abs/2110.06696 (2021). https://arxiv.org/abs/2110.06696