

Article

Automatic Generation of Software Prototype Data for Rapid Requirements Validation

Shuanglong Chang ¹, Juntao Gao ² and Weiru Wang ^{3,*}¹ Institute of Unmanned Systems, Beihang University, Beijing 100191, China; shuanglongchang@buaa.edu.cn² School of Computer and Information Technology, Northeast Petroleum University, Daqing 163318, China; gjt@nepu.edu.cn³ School of Computer, Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China

* Correspondence: weiru.wang@bjut.edu.cn

Abstract

In the early stages of computer software information system development, requirements errors can lead to software system failure and performance degradation and even cause huge security incidents. Traditional requirements verification methods are inefficient and susceptible to human factors when dealing with complex software requirements. Rapid prototyping is an effective requirement validation method, but the generated prototype does not contain any data, and the traditional method requires domain experts to write the data manually, which is time consuming and complicated. In this study, an automatic software prototype data generation method, InitialGPT, is proposed, which automatically generates requirements-compliant prototype data by interacting with users through a requirements model to improve the efficiency and accuracy of requirements validation. We designed a framework containing a prompt generation template, a data generation model, a data evaluation model, and multiple prototype data tools, and validated it on four real-world software system cases. The results show that the approach improves the efficiency of requirements validation by a factor of 7.02, and generates data of similar quality to those written manually, but at a more advantageous cost and efficiency, demonstrating its potential for application in the computer software industry.



Academic Editor: George Angelos Papadopoulos

Received: 19 May 2025

Revised: 23 July 2025

Accepted: 23 July 2025

Published: 1 September 2025

Citation: Chang, S.; Gao, J.; Wang, W. Automatic Generation of Software Prototype Data for Rapid Requirements Validation. *Electronics* **2025**, *14*, 3497. <https://doi.org/10.3390/electronics14173497>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: requirements error; rapid software prototype; data generation; data quality; requirements validation

1. Introduction

Requirements errors are one of the reasons for the failure of computer software projects [1], and in serious cases, they can cause great disasters. With the rapid development of information technology, the requirements of computer software systems become increasingly complex and variable. In the face of this challenge, careful requirements modeling and system verification play a crucial role in reducing the uncertainty of the target system [2,3]. However, this requires developers and stakeholders to work together to define the goals of the system's functionality. A common challenge in this context is communication barriers between developers and stakeholders [4].

Traditional methods of requirements validation, such as the preparation of requirements documents and requirements review meetings, are not only inefficient, but also susceptible to human factors, resulting in insufficient accuracy and completeness of requirements validation. Rapid prototyping, as an effective requirement validation method, helps

development teams and stakeholders to better understand and validate the requirements by rapidly building runnable software prototypes [5]. The current state-of-the-art method RM2PT [4–7] is able to rapidly model software prototypes from formal requirements in UML diagrams by generating software prototype systems, complemented by system operation contracts in Object Constraint Language (OCL), enabling customers to visually validate their requirements, thus easily discovering misunderstandings and uncertainties in the requirements.

However, these generated software prototypes do not contain any initial data [5,8]. Software prototype data generation has been a key bottleneck in the prototyping process, and traditional approaches usually rely on hiring domain experts to manually write the prototype data for requirements validation of software prototypes [9]. This is not only complex, time consuming, inefficient, and costly to manually write [10], but also presents difficulties in meeting the requirements of rapid iteration due to low user feedback iteration efficiency and high tendency to introduce human errors [11].

Therefore, developing a method to automatically and efficiently generate high-quality prototype data has become an urgent problem in the field of software development. Unlike data generation for software testing [12], test data need to represent code logic boundary use cases (e.g., abnormal inputs, stress loads, program fixes, etc.), while prototype data need to represent business scenarios (e.g., user profiles, typical workflows). Initial data generation for computer software prototyping systems is based on requirements validation for demonstrating and verifying software functionality as well as collecting user feedback. Therefore, it is not only reasonably effective for stakeholders, but also focuses more on requirements engineering and user experience in the current state of the system, emphasizing user feedback and representativeness.

Despite the results of existing research, the application of automation in the early requirements validation phase of software development still faces many challenges. On the one hand, most of the existing requirement validation methods focus on the preparation of requirement documents and manual data preparation and review, and lack of automation support for the requirement validation process. On the other hand, although large language models [13] (LLMs) are capable of generating high-quality natural language text by virtue of their powerful performance, e.g., some researchers have already started to explore the potential of using LLMs to generate synthetic data for text categorization [14], Rokas Štrimaitis [15] et al. attempt to synthesize time series data for forecasting expected trends in accounting data, although how to effectively apply them to the generation of data in the requirements validation phase of software prototypes to alleviate the burden on humans is still an urgent problem.

Based on the above challenges, in this paper, we propose a model-driven automatic initial data generation method (InitialGPT) for software prototypes based on large language model (LLM) and intelligent body (agent) workflow, which for the first time realizes automatic generation of high-quality initial data for rapid prototypes in the requirements validation phase in line with business logic and constraints, and significantly solves the core problems of inefficiency, high cost, and error proneness of traditional manual generation methods.

The intelligent model-driven approach (IMDA) combines model-driven development (parsing requirement models) with an LLM-driven intelligent workflow. The core innovations are:

- **Automatic Generation of Requirement Models to Structured Prompt Templates:** A layered prompt generation template (initial instruction layer, entity information layer, user interaction layer, data output layer) is designed to automatically extract key information (entities, attributes, and business rules) from formalized requirement models

(use case diagrams, class diagrams, and OCL constraints) and dynamically generate high-quality LLM prompts by integrating user interaction inputs and structured LLM prompts. This is the key to ensure that the generated data conforms to business logic and constraints.

- **Multi-Agent Collaborative Data Generation Workflow:** A multi-agent collaborative framework is designed that includes data generation agents, data validation, assessment agents, and data format output agents. A dynamic selection strategy (single vs. multiple agents) is innovatively adopted to strike a balance between generation speed and data quality. Multi-intelligence collaboration significantly improves the business compliance, accuracy, and diversity of the data generated through task decomposition, logical reasoning, and consistency checking.

In this paper, we pioneered the application of an LLM’s powerful generative capabilities to initial data generation for software requirement validation scenarios, solving the long-standing pain point of relying on manual labor in this area. The focus is on generating representative data for feature demonstration, user experience, and gathering feedback, rather than test data.

The rest of the paper is structured as follows: Section 2 describes the preliminary work related to the methodology of this paper; Section 3 details the components of the InitialGPT methodology; Section 4 evaluates InitialGPT through four case studies; Section 5 discusses potential threats to the validity of the study; and Section 6 summarizes the full paper’s work and looks forward to future directions of the study.

2. Preliminary

2.1. Requirements Model

Yang et al. [5] developed a requirements model by pruning UML and extracting components that describe traditional software systems’ requirements. Shown in Figure 1 is a requirement model (CoCoME) of a supermarket example system, through which a supermarket prototype system can be quickly generated for requirements validation. The model comprises a system operating contract and three views:

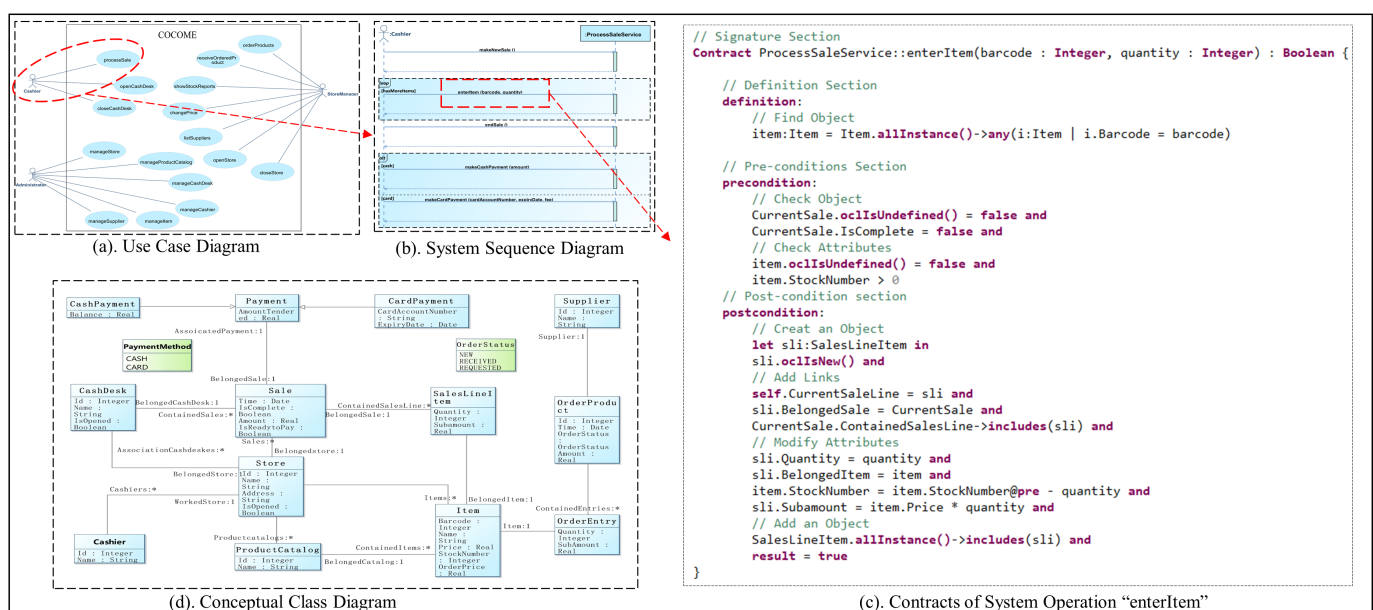


Figure 1. CoCoME requirements model.

Use case diagrams help clients and domain experts define the functional boundaries of a system by describing the interactions between the system and its users. A system

sequence diagram further refines the execution process of a specific use case, showing the sequence of interactions between participants and the system and the triggering process of events within the system. The conceptual class diagram, as the conceptual model of the problem domain, abstracts the core concepts and their structural relationships in the domain through the association between classes and their attributes. Each of the three supports the understanding and expression of system requirements at different levels.

Object Constraint Language (OCL) (<http://www.omg.org/spec/OCL/>, accessed on 22 July 2025) is a declarative language designed to address the limitations of UML diagrammatic notations by enabling precise specification of system design details. As a formal textual language, OCL supports the definition of constraints and object query expressions for any meta-object facility (MOF) model or meta-model, thereby facilitating the description of model elements beyond the expressive capacity of visual diagrams.

The contract of a system operation is described in Object Constraint Language, which specifies the conditions that the system state must fulfill before and after the execution of the system operation. Among them, preconditions define the properties that the system state must satisfy before the execution of a system operation, which are used for state checking during execution; postconditions define the conditions that the system state should satisfy after the successful execution of a system operation, and make clear the state changes that may be brought about by the operation.

2.2. Prototype

As shown in Figure 2, the figure presents an example of manual initialization data construction in the prototype system. The interface in the figure consists of two functional areas on the left and right: the state observation panel of the CoCoME prototype system on the left, and the system function panel on the right. With the help of this prototype interface, the user can complete the initial data entry and configuration through manual interaction. The left side of the figure shows the state observation panel of the CoCoME prototype, which is divided into four main areas. The top left corner shows the names of the current classes in the system and their number of objects, for example, the number of objects in the Store, CashDesk, Sale, and Item classes.

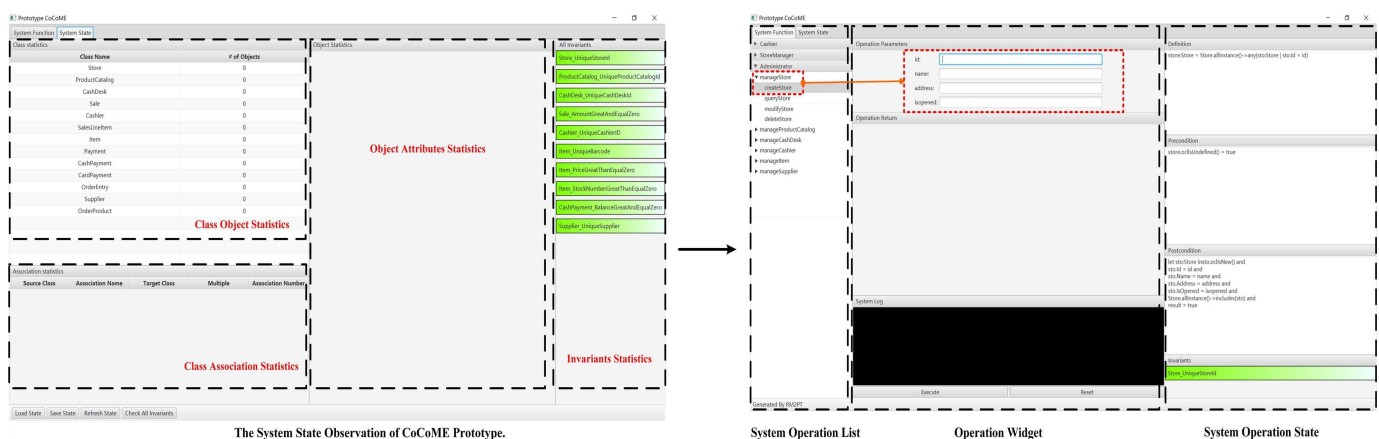


Figure 2. Example of manual initial data creation for prototype.

The lower left section of the panel displays the association status, including the source object, target object, association name, the number of associated objects, and whether the association is one-to-one or one-to-many. When a class entry on the left is clicked, the corresponding property and association statuses are shown in the center and lower left areas of the panel. On the right side, all system invariants are listed; if any invariant is violated, it is highlighted in red, similar to the display in the System Functions panel.

The rest of this section demonstrates how the generated prototype can be used to validate system requirements.

On the right is the System Functions panel of a prototype system, which is divided into three main parts: the list of system operations on the right, the operation widgets in the center, and the status of the system operation on the right, which is shown in green when the operation is executed successfully, and in red when there is an error in the operation.

Therefore, in the requirements model, if you want to add initial data, you need to model the data functions and the stakeholders corresponding to the data functions (e.g., administrators), and then manually create the data in the prototyping system, which can be observed in the system status bar after successful creation (left). However, requirements modeling is often time consuming and costly, and manually writing data also possesses the complexity mentioned above, and thus greatly hinders the efficiency of requirements validation.

2.3. The Necessity of Generating Software Prototype Data

As illustrated in Figure 1, the requirements model provides a structured representation of system specifications through formal modeling techniques, encompassing functional requirements, behavioral patterns, and entity relationships. Stakeholders can rapidly validate requirements by interacting with a prototype system generated from this requirements model. However, as shown in Figure 2, the prototype system typically lacks operational data for validation purposes. Firstly, it needs to be manually prepared and created by the user at the administrator, and for the stakeholders (e.g., administrators) in this part of the data functionality, it also requires spending a lot of time and cost to model it. Manually preparing the data also possesses the same time-consuming cost and complexity, and therefore represents a great impediment to the efficiency of the requirements validation. Manual preparation of initial data remains mandatory in many application scenarios. However, manually writing initial data for prototypes remains a very time consuming, costly, and complex process for stakeholders. This complexity is caused by the following interrelated attributes [16–18]:

- Complexity of manually authoring data: The complexity of manually authoring data is not only related to the creation of the data, but also to the need for an in-depth understanding of the real-world scenarios of the domain behind the data and the fact that the process of writing the data may involve multiple teams. Therefore, generating the data efficiently is a complex process.
- Diversity and complexity of initial data: Initial data generation for requirements validation is different from database data generation, and focuses more on requirements engineering and user experience, as well as helping in the demonstration and validation of business functionality. Therefore, it is a challenge to ensure that the initial data used in the prototyping system has sufficient diversity and applicability, reality, etc., which requires the simulation of complex business logic and user behavior.
- Diversity of specific domains: Initial data generation is used to serve prototype systems for different domains and is generated for specific domains. Therefore, every time the data are generated, they need to rely on the knowledge and experience of domain experts to ensure that the data are highly representative and specialized. This process is often time consuming and costly, as it requires experts to deeply understand and reflect the complexities and nuances within a given domain.
- Inefficient user feedback mechanism and low user interaction: The process of manually writing data is usually not a one-time process. Depending on the feedback and practical applications, the dataset may need to be optimized and iterated, but the inefficient user feedback mechanism makes it a key challenge to effectively integrate user feedback and reduce the burden on the user in the data generation process.

- Automated data generation and management: Currently, generating data requires more manual writing and adding to prototypes manually, and there is a lack of actionable data integration tools, so it is also a challenge to develop efficient automated tools to manage and integrate prototype data and reduce manual intervention.

2.4. Large Language Model and Prompt Engineering

Here are the large language models and prompts engineering studies related to Initial-GPT's work.

Large language models based on the Transformer architecture [19] have made significant progress in the field of natural language processing.

The utilization of bi-directional context in the bidirectional encoder representations from Transformers (BERT) model [20] has already achieved excellent performance in tasks such as text classification and named entity recognition (NER). Building on this success, the GPT family of OpenAI models (including models such as GPT-2 [21], the massive GPT-3 [22] with 175 billion parameters, and the latest GPT-4 (OpenAI, 2023)) have made huge breakthroughs in training parameters. These models have made breakthroughs in generating high-quality text [23–25], demonstrating inference [26] (Wei et al., 2021), translation [22], synthetic data generation [27], and code generation [28]. Recently, researchers have investigated how to improve the robustness and performance of models in different domains through prompt engineering [29–35]. Gwern Branwen et al. [36] suggested that the prompt engineering model can be a new interaction paradigm. Users only need to know how to prompt the model to get the targeted output needed to accomplish downstream tasks. Ge et al. used BERT to generate prompts [37] and then prompt LLMs to generate visual descriptions of blended images to help users with visual fusion. Liu et al. proposed a paradigm [38]: prompt-based learning. This paradigm is based on language models directly modeling the probability of the text compared to traditional supervised learning. It allows pre-training of the language model on massive amounts of raw text, and by defining new prompt functions, the model is able to perform little or even zero learning, adapting to new scenarios with little or no labeled data. In this study, we focus on exploring the capabilities and limitations of the state-of-the-art GPT-4 and GPT-3.5-Turbo models, using them to generate initial data in a requirements validation prototype system.

2.5. Generative AI in Data Generation

In recent years, researchers have explored the use of generative AI techniques for data generation. For instance, Wang et al. [39] suggest that using GPT-3 for dataset labeling is more cost effective than manual annotation, and the resulting labeled datasets can be used to train smaller, more efficient models. Similarly, Ding et al. [40] introduce three methods for generating training data with GPT-3: unlabeled data annotation (generating labels for known examples), training data generation (generating both examples and labels), and assisted training data generation (using Wikidata as additional context). When used to fine-tune a smaller BERT model for text classification and named entity recognition tasks, these methods achieved performance comparable to or slightly worse than GPT-3. Dai et al. [41] propose AugGPT, which uses ChatGPT (GPT-3.5) to augment each example in a small base dataset with six rephrased synthetic examples, which are then used to fine-tune a specialized BERT model. Other studies have also investigated the capability of language models to generate synthetic data for various text classification tasks [42–45]. However, to date, there has been no research applying generative AI to data generation for software prototypes used in requirements validation.

3. InitialGPT

In this section, we first present the prompt generation template designed for InitialGPT as input. Subsequently, we elaborate on the core data generation model and the data evaluation model. Following this, we describe several data post-processing tools that have been developed. Finally, we introduce the template utilized for automatic code generation.

3.1. Overview

As shown in Figure 3, InitialGPT is an intelligent model-driven approach for automatically generating prototype initialization data to support rapid requirements validation. The purpose of this section is to describe the software architecture of InitialGPT. As shown in Figure 1, InitialGPT takes a requirements model as input and then automatically generates initial data for a prototype system. This process is divided into three distinct phases:

The first phase is to parse the requirements model and automatically generate a large language model prompt template. The second phase is to automatically generate the initial data for the prototyping system, and a multi-agent workflow containing several intelligent agents is designed. The third phase is data evaluation and post-processing, which helps users to quickly understand the quality of the data through data benchmarking and supports rapid iteration, and includes several data tools to improve the quality of the data, and lastly, InitialGPT automatically generates the code to be integrated into the prototyping system for the efficient automated generation of prototype data. The four-phase approach provides a structured methodology for efficiently and accurately generating initial data for prototyping systems from requirements models to facilitate stakeholder requirements validation.

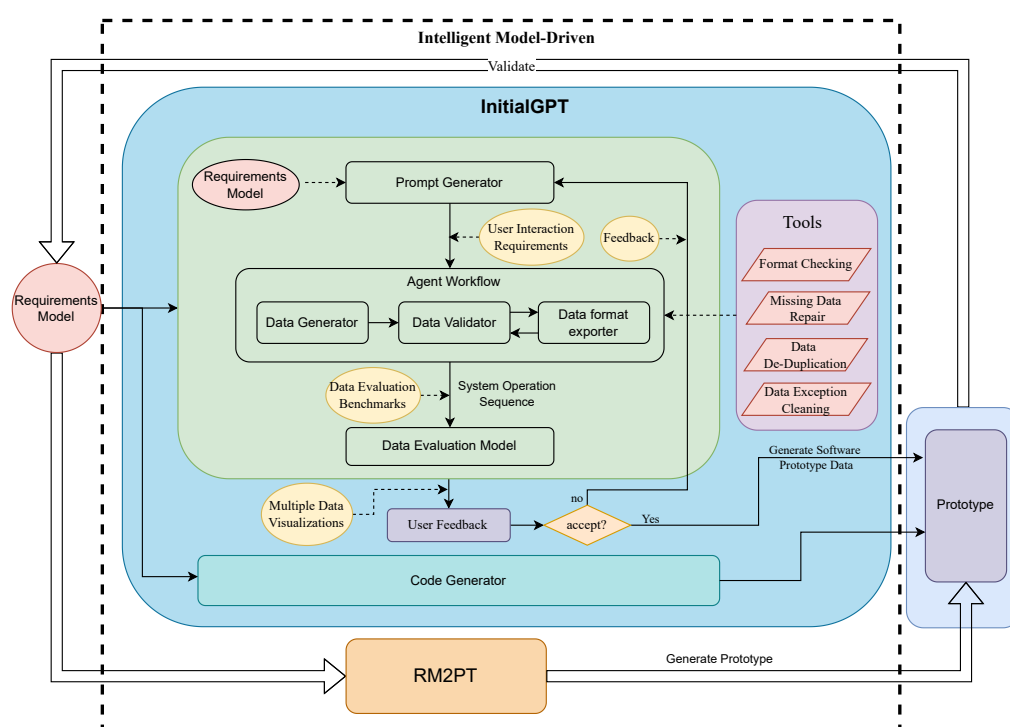


Figure 3. Overview of InitialGPT.

3.2. Architecture Design Based on IMDA Paradigm

IMDA (intelligent model-driven architecture), as the top-level architectural principle guiding the design of InitialGPT, emphasizes the requirement model as the driving source (model-driven), and realizes the automated closed-loop of data generation–

validation–evaluation through intelligent agents. IMDA is not a specific tool but a paradigm framework that integrates the model-driven engineering and artificial intelligence paradigm framework.

InitialGPT, as an automated software prototype data generation method proposed in this paper, has the core objective of generating initial data that conforms to business rules to support rapid requirements validation through the interaction of a large language model and a requirements model. Its innovative kernel is embodied in threefold capabilities: (1) dynamic generation of prompt templates based on requirements model parsing; (2) closed-loop data generation–validation–evaluation that integrates large language modeling and multi-intelligence collaboration; and (3) a quantitative data quality optimization mechanism following the ISO/IEC 25012 standard. The methodology has the core objective of improving the efficiency of requirements validation and replacing the traditional manual construction process through structured data generation.

The IMDA paradigm constitutes the meta-architectural paradigm that guides the design of InitialGPT. The theoretical framework of the IMDA paradigm, with model-driven engineering as the cornerstone and the introduction of intelligent agents to automate the process, can be formally defined as a triad: $IMDA \triangleq \langle M, A, F \rangle$, where $\mathcal{M}$ denotes the requirements model parsing layer, $\mathcal{A}$ is the intelligent agents collaboration layer, and $\mathcal{F}$ stands for the feedback optimization layer. IMDA is not a specific technical implementation, but a cross-domain generalized intelligent system design principle.

3.3. Requirements Model Parsing

The automatic parsing of requirements models is a fundamental step in the InitialGPT approach. In this study, a model-driven engineering (MDE) approach is adopted to transform textual requirements into a structured model conforming to the EMF meta-model by means of a domain-specific language (DSL) defined in Xtext. Firstly, domain entities, attributes, and association relationships in the requirements model are formalized as ECore meta-model elements whose abstract syntax is bounded by Xtext syntax rules (e.g., ‘entity’ name = ID “(attributes+=Attribute)*”). This phase realizes the text-to-model conversion by means of a lexical analyzer (Lexer) and a syntactic analyzer (Parser) generated by Xtext.

The extraction of domain entities is realized through the EMF model traversal algorithm. Figure 4 shows part of the code template used to convert domain conceptual information into entity class parsing, while Figure 5 is an example of the Xtend-generated entity class code. First, the Xtend generator loads the parsed EMF model instance and calls the `eAllContents()` method to perform a depth-first traversal to filter out all model elements of type entity. For each entity, its attributes and associative relationships are recursively collected and a lightweight `EntityInfo` data transfer object is constructed. Attribute mapping extracts the attributes of entity types via an `entity.attributes.map`, and encapsulates the domain concepts corresponding to each class into the corresponding entity parsing class; the final generated collection of domain entities will be used as the input for subsequent hint template generation. Compared with traditional methods such as regular expressions or template matching, this method ensures syntactic consistency through formal meta-models and supports type-safe model manipulation. We further encapsulate the template code written by Xtend into a scripting tool to form execution scripts applied to the prototyping system platform. The execution scripts can automatically generate the final tool code embedded in the prototyping system, which ultimately realizes the automatic generation of prototype data. Experiments show (see Section 4.1) that the method achieves 98.7% entity extraction accuracy in a requirement model containing 100+ entities.

```

def compileEntity(Entity entity) {
...
    package entities;

    import services.impl.StandardOPs;
    import java.util.List;
    import java.util.LinkedList;
    import java.util.ArrayList;
    import java.util.Arrays;
    import java.time.LocalDate;
    import java.io.Serializable;
    import java.lang.reflect.Method;

    public class «entity.name»«IF entity.superEntity != null» extends «entity.superEntity.name»«ENDIF» implements Serializable {
        /* all primary attributes */
        «FOR attribute : entity.attributes»
            private «attribute.type.compileType» «attribute.name»;
        «ENDFOR»
    }
...
}

```

Figure 4. Xtend code template.

```

package entities;

import services.impl.StandardOPs;
import java.util.List;
import java.util.LinkedList;
import java.util.ArrayList;
import java.util.Arrays;
import java.time.LocalDate;
import java.io.Serializable;
import java.lang.reflect.Method;

public class Item implements Serializable {

    /* all primary attributes */
    private int Barcode;
    private String Name;
    private float Price;
    private int StockNumber;
    private float OrderPrice;
}

```

Figure 5. Example of generated entity class code.

3.4. Prompt Template Generation

In the text generation process based on large language models, the quality of input prompt words directly determines the accuracy, diversity, and business adaptability of the generated data [29]. Since large language models rely on natural language input to reason and generate data, how to construct high-quality prompt words that conform to business logic and cover diverse data requirements is a key challenge in the data generation process.

To address this issue, this paper proposes an automated prompt template generation method, which aims to dynamically generate high-quality input prompts based on the requirement model, ensuring that the data generated by the model can meet the business requirements with a high degree of structuring, consistency, and adaptability.

The prompt generation template designed in this framework uses a layered architecture to realize the semantic mapping of requirement information to natural language instructions.

The template consists of the following four parts: Figure 6 shows InitialGPT's prompt template rules, and Figure 7 illustrates an example of a prompt for the CoCoME supermarket system, constructed using the small-scale template rules. The initial instruction defines the basic scenario paradigm of the generation task, the entity information layer injects structured descriptions of domain concepts extracted from the EMF meta-model (including entity names, attribute data types, and so on), the user interaction layer incorporates personalized generation parameters (e.g., number of instances, range of values), and the output format layer specifies the structured representation of the data (e.g., YAML fields and indentation rules).

Id	Sample of linguistic patterns	Examples of linguistic patterns
Patterns related to initial prompts: <IP Ptn>		
1	Small amount of data	System Role Information... + You only need to generate the ones in the square brackets, and you don not need to generate the ones notlisted...: + <the additional prompts>
2	Large amount of data	Generate 40 entity examples of the system based on these <entity_name> information, try to generate reasonable and realistic data...: + <the additional prompts>
Patterns related to user interaction information: <UI Ptn>		
3	Small amount of data	please generate the number of entity instances according to the given values: + User interaction entity quantity requirement information...
4	Large amount of data	The following is to give all the entity information, please do not generate all of them, please follow my request: select <entity_name> entity information from these entity information to generate, just generate the 40 example.
Patterns related to entity information: <EI Ptn>		
5	Small / Large amount of data	<entity_name1>(attribute1(type),attribute2(type), ...), <entity_name2>(attribute1(type),attribute2(type),...)...
Patterns related to output format information: <OFI Ptn>		
6	Small amount of data	The entity data is directly returned in the format of a yaml file, and the yaml data is in this format: Item: - Barcode: 125564890 Name: Backpack Price: 49.99 StockNumber: 20
7	Large amount of data	The entity data is directly returned in the format of a yaml file, and the template of the generated data as follows (Please omit the entity name) : - Xxx: xxxxx Xxx: xxx Xxx: xxx

Figure 6. Prompt template.

The dynamic generation process is realized through a model traversal algorithm: firstly, the entity elements in the requirement model are extracted based on the Xtext parser (`Model.eAllContents.filter(Entity)`), and then the entity tuples <name, attributes, constraints> are populated into predefined slots by the string template engine. When the user specifies the generation scale, the system dynamically updates the quantity parameter using a regular substitution strategy, e.g., replacing N with a specific value. This hierarchical dynamic mechanism ensures that the cue word conforms to the formal requirements specification while flexibly responding to interactive adaptation needs.

Initial Instruction Layer: This layer defines the specific business scenario. Its core functions include: defining the domain scope of the generation task (e.g., “Generate the initial data of the supermarket management system”), declaring the role of the LLM (e.g., “You are a professional data generation expert”), and constructing the reasoning logic framework underlying the task (e.g., “Comply with the entity-relationship constraints”). The text of this layer adopts solidified templates to realize the injection of domain knowledge and to ensure that the generation target is semantically aligned with the business vision.

Entity Information Layer: This layer defines the data generation target entities and attributes. Based on the model-driven engineering technology of the EMF meta-model, it realizes the structured transformation of domain concepts: through the Xtend code template engine, the tuples <entity name, attribute set, constraint set> are dynamically filled into the predefined positions to form the domain knowledge descriptions in line with the business logic.

User Interaction Layer: This layer defines information such as the target number of data generation. By interacting with users, it collects their personalized requirements, provides flexible configuration options, and optimizes the generation of hints. Although the big language model has a strong generalization ability, in specific business scenarios, users may want to have fine-grained control over the data generation method, such as specifying the quantity or format of the generated data (e.g., generating 100 pieces of customer data in YAML format), or needing a specific range of values for certain fields (e.g., the order amount must be in the range of 100–500), attaching comments, etc. The user interaction layer provides a flexible configuration option to optimize the generation of prompt information. The main functions of the user interaction layer include: collection of

requirement information, customized prompts for feedback, etc. The model is divided into two parts.

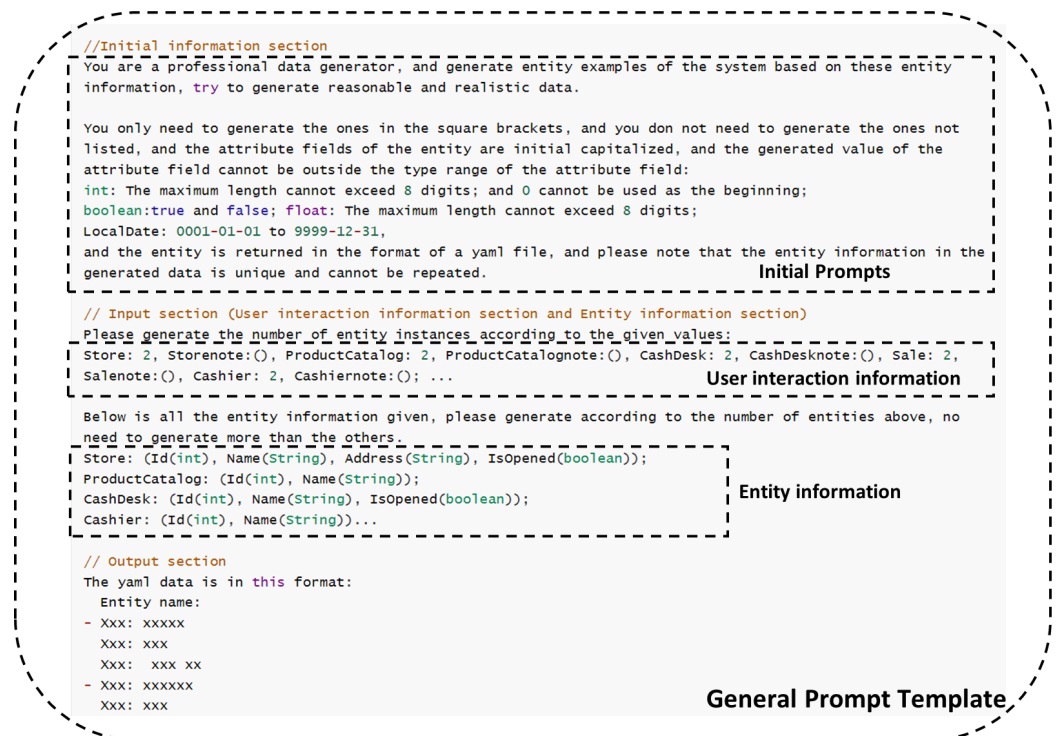


Figure 7. A prompt example for a supermarket system.

Output Format Layer: This layer defines information such as the format of the generated data. Output format information is shown in Figure 6. Through the sample specification LLM output, the goal is to guide the data generation model to generate data in accordance with the YAML format.

In the process of prototype data generation, due to the existence of a context length (token) limit, requests per minute (RPM) limit, and number of tokens processed per minute (tokens per minute, TPM) limit in the large language models, in order to ensure the efficiency and quality of the data generation, we divide the prompt templates into two categories: small-scale prompt templates and large-scale prompt templates, and the specific design principles are as follows:

- Small-scale prompt templates are used for rapid iteration and early stage validation, applicable to scenarios where stakeholders provide rapid feedback on core entities. This category of templates focuses on generating a small number (less than 40 in total) but diverse entities to fulfill the need for rapid validation and feedback. For example, in the early stages of requirements analysis, we may ask the model to generate 20 different types of supermarket merchandise entities in order to quickly validate that the data format and entity attributes meet expectations.
- Large-scale prompt templates are used for batch generation of a specific type of entity. This type of template is optimized for a specific entity type to improve generation efficiency and data consistency. For example, in the system testing phase, we may need to generate 300 supermarket product entities (Item) at one time, in order to simulate the data scale and business logic in real scenarios.

Figure 8 shows an example of a prompt in the CoCoME system that generates a large number of single-class entities individually, constructed using large-scale prompt rules. The large-scale prompt template is similar to the small-scale prompt template,

but it is generated for a single entity, 40 entities at a time, and post-processed until the final generated data meet the requirements, at which point it stops. With this template, the model is able to efficiently generate a large amount of structured data in a single request, meeting the needs of system validation, data simulation, and other scenarios.

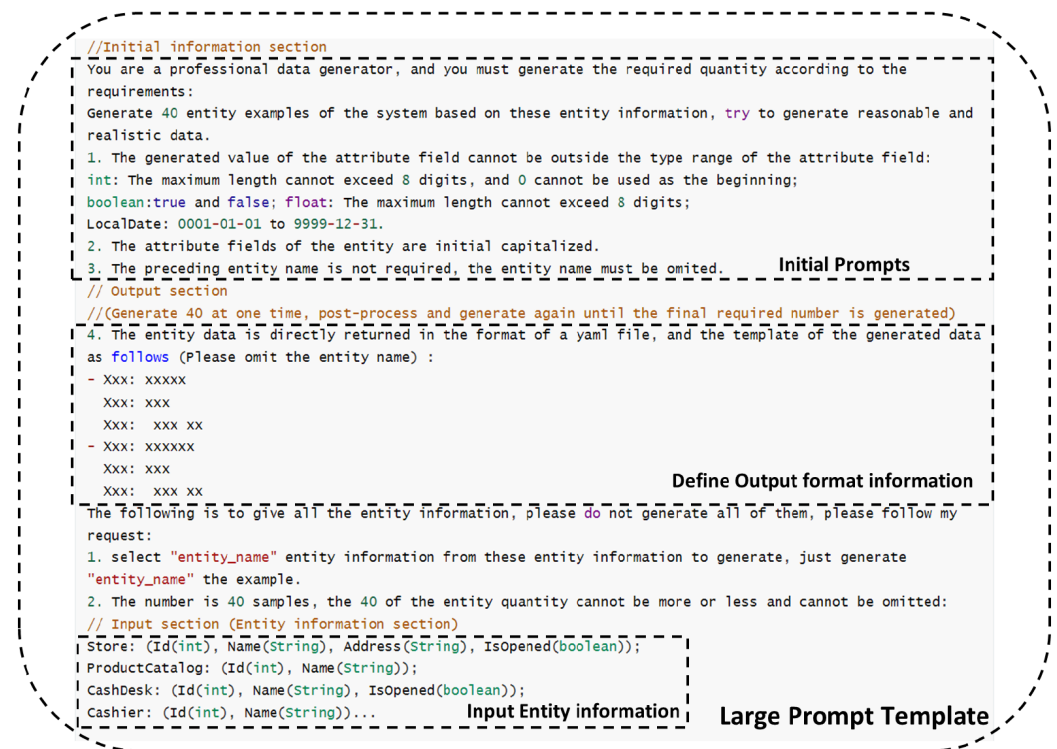


Figure 8. A large-scale prompt example for a supermarket system.

Through the above classification design, we are able to balance the efficiency and quality of data generation under the premise of meeting the performance constraints of the model.

3.5. Data Generation

The data generation method is the core module of the framework, which is responsible for generating specific data content based on the prompt template. The data generation model is divided into an engine layer and a base layer. The engine layer is the execution module of data generation, based on the generation capability of the large language model, combined with the template selection strategy, parallel scheduling strategy, and adaptive intelligent body strategy to achieve efficient data generation. It mainly includes:

Multi-Agent Strategy: In the process of data generation, there are differences in the speed and quality requirements for data generation in different application scenarios. In order to balance the trade-off between fast generation and high-quality generation, this paper proposes a multi-agent selection strategy, which is able to select single-agent mode (Single-Agent) or multi-agent workflow collaboration mode (Multi-Agent) according to the user requirements and business needs to meet the requirements of data generation in different scenarios. Each agent is responsible for different sub-tasks, such as the data generation agent, data validator agent, data evaluation agent, and data format output agent. The single agent is suitable for rapid data generation for initial validation of requirements or small-scale data filling to ensure that usable data are generated in a short period of time. The multi-agent is suitable for generating high-quality data by collaborating with multiple agents to perform data parsing, logical reasoning, and consistency, checking to ensure that

the data conforms to business constraints and to improve the completeness and accuracy of the data.

The multi-agent collaboration framework designed in this study adopts a pipelined workflow architecture, which realizes the prototype data generation and optimization through the orderly collaboration of four types of intelligent bodies, namely data generation (Generator Agent), data validation (Validator Agent), data evaluation (Evaluator Agent) and data format export (Exporter Agent).

The data generation agent generates raw JSON data streams conforming to the domain semantics based on dynamically constructed cue templates prompting a large language model (e.g., GPT-4o). The data validation agent performs constraint checking of the three types of data errors, namely missing data, data duplication, and data exceptions, on the generated data through the dual validation of set rules and agents, and feeds back the exception data that violate the business rules to the generation node to trigger the regeneration process. The data evaluation agent introduces the ISO/IEC 25012 data quality model, performs multi-dimensional quantitative analysis on the validated dataset (including 10 indicators such as completeness, accuracy, diversity, etc.), generates structured evaluation reports (e.g., “Completeness”: 95%, “OutlierRate”: 1.2%), and dynamically optimizes parameter configurations of prompt templates through feedback loops.

Ultimately, due to the instability of the large language model, the output content format is often uncontrollable, so we have set up a data format agent to convert high-quality data into a YAML format that can be loaded by the target system, and realize the integration with the RM2PT prototyping platform with the help of a control interface, in which the data exchange format follows the unified JSON Schema specification to ensure that the type of data for the whole process is safe and traceable.

Prompt Template Selection Strategy: We propose an automatic prompt template selection strategy. This strategy dynamically selects the appropriate prompt template based on the number of entities to be generated, specifically: (1) if the number of entities of a single type exceeds 40, the system automatically switches to the large-scale prompt template to ensure efficient batch generation; (2) if the total number of diverse entities does not exceed 40, the system uses the small-scale prompt template, which is designed for rapid iteration and early-stage validation. This adaptive selection mechanism allows us to balance between generation efficiency and output quality, depending on the scale and purpose of the data generation task.

Parallel Scheduling Strategy: In order to improve the generation efficiency (due to API rate limits imposed by large language model providers—for example, OpenAI’s free tier allows only three requests per minute) and give full play to the performance of the large language model, the engine layer supports increasing the throughput of model-generated data through task splitting and parallelization. Specifically, we utilize data splitting and multi-threading techniques to generate multiple types of data entities simultaneously, significantly reducing overall generation time.

The system employs a cyclic multi-threading approach to maximize the utilization of the large language model’s performance. Multiple threads concurrently submit generation requests, effectively bypassing rate limitations and improving throughput. After generation, data collected from various threads undergoes deduplication, merging, and quality assurance processes to ensure the final dataset is accurate, consistent, and of high quality. This parallel scheduling strategy effectively balances API limitations with generation efficiency, enabling rapid and scalable data generation suitable for diverse application scenarios.

Foundation Layer: The foundation layer provides underlying support for data generation, including model management, format matching, tool invocation, etc., to ensure that

the generated data conforms to the business model, as well as to ensure the stability and scalability of the generation process, and to support large-scale data generation tasks.

3.6. User Interaction Mechanism

InitialGPT realizes user-oriented closed-loop optimization through the interaction interface, which is divided into two key phases: pre-generation configuration and post-generation evaluation. In the pre-generation phase, the system automatically builds an initial prompt template by parsing the demand model, and the user can intervene in multiple dimensions through the visual interface (as shown in Figure 9), including but not limited to setting the number of entity instances (e.g., generating 200 Customer objects), defining attribute constraints (e.g., the OrderDate needs to be in the period of 2020–2023), and selecting the generation strategy (either single-intelligence rapid generation or multi-intelligence precise generation). At the same time, as shown in Figure 10, the system provides prompt word editing functions to allow users to directly modify the automatic prompt template, such as adding domain-specific business rules (“VIP customer order amount should be greater than 500 dollars”). This design transforms the traditional black-box LLM generation process into a transparent and controllable interactive process, effectively reducing the cognitive load on users.

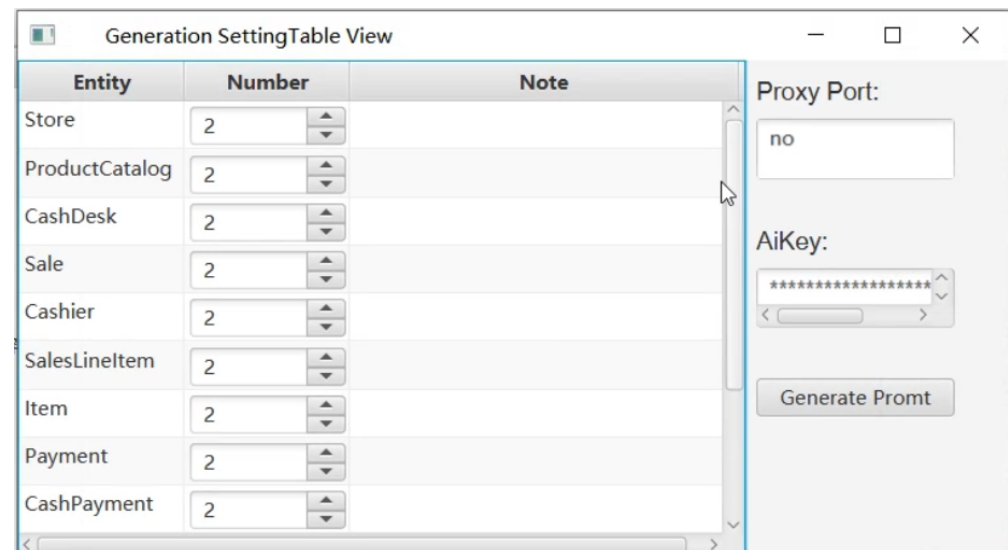


Figure 9. User interface.

In the post-generation evaluation phase, the system presents the generation results through a multi-dimensional visualization interface and supports in-depth interactive validation. Specifically, the system presents the automatically generated initial data in a structured form (e.g., YAML tree diagram or relational table, as shown in Figure 11), along with a quality assessment report based on the ISO/IEC 25012 standard, which contains 10 indicators such as completeness (e.g., entity coverage) and accuracy (e.g., outlier ratio). The interactive panel allows the user to perform three types of key operations: (1) data filtering and sorting to quickly locate potential problems; (2) manual correction of outlier data values; and (3) overall acceptance/rejection decisions, which can be regenerated when rejection is selected.

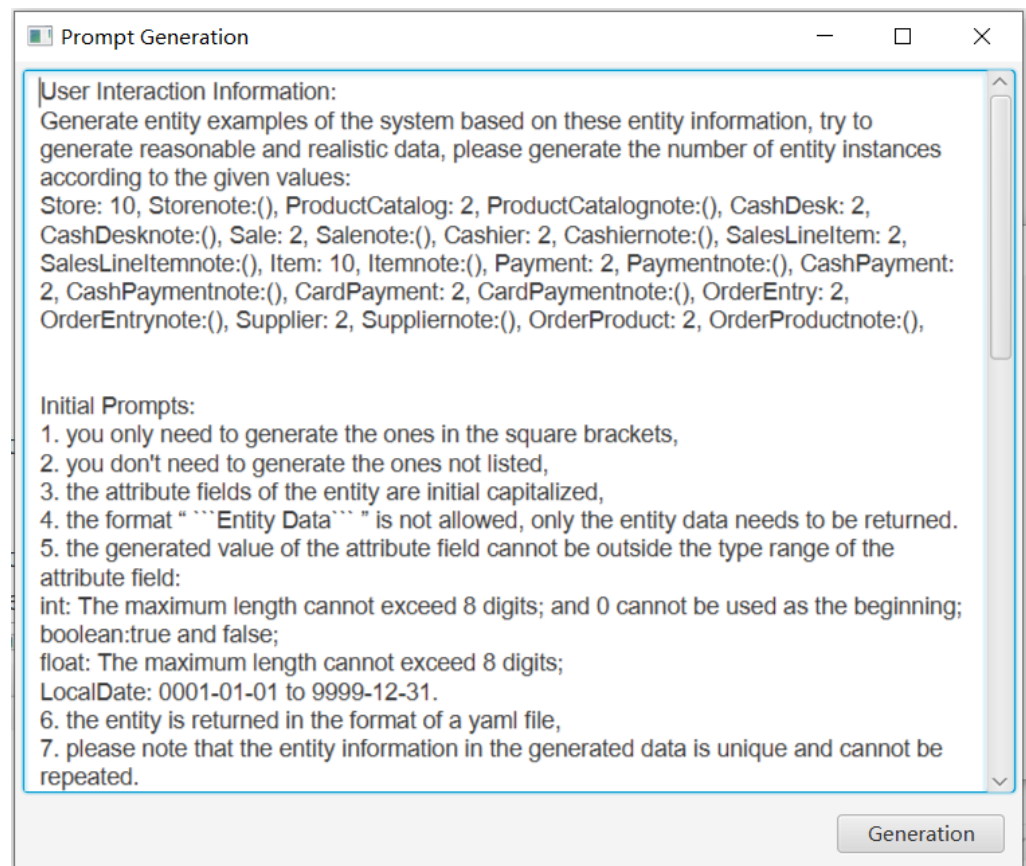


Figure 10. Prompt for dynamic generation.

Initial data generation

Entity	Barcode	Name	Price	StockNumber	OrderPrice
Sale	123456789	Laptop	999.99	10	899.99
CardPayment	234567890	Smartphone	699.99	15	649.99
OrderProduct	345678901	Headphones	199.99	30	179.99
ProductCatalog	456789012	Keyboard	49.99	50	44.99
CashPayment	567890123	Mouse	29.99	60	24.99
Store	678901234	Monitor	299.99	20	279.99
CashDesk	789012345	Printer	149.99	25	139.99
Cashier	890123456	Tablet	399.99	18	379.99
Item	901234567	Camera	499.99	12	459.99
Payment	123456780	Speaker	89.99	40	79.99
OrderEntry					
Supplier					
SalesLineItem					

Current State: Generation is complete!

Model: gpt-4o
Generation Setting

View: Yaml View, Table View

Evaluation: Automatic Evaluation

Load Data

Figure 11. Table view of generated data.

3.7. Automatic Data Evaluation

Data evaluation aims to help users quickly measure the validity of the prototype data through a series of automated data quality assessment methods and make adjustments to the prompt words, generation logic, or the data based on the assessment results, in order to achieve closed-loop optimization. The method undertakes the important tasks of data quality assessment, automated feedback adjustment, and secondary data generation in the data generation framework.

Quality Evaluation: After data generation, automatic quantitative evaluation of the generated data is performed based on predefined evaluation metrics. In the absence of

actual data in the prototype data generation stage, there are currently no effective data quality evaluation metrics applicable to software prototype system requirement validation in the industry. Based on this status condition, this paper proposes the data quality evaluation metrics for software prototype requirements validation. This metric is intended to fill the gap in the existing evaluation system and provide a set of systematic evaluation criteria for the data quality of software prototypes, so as to support a more accurate and reliable requirement validation process.

The evaluation metrics are based on the international standard ISO/IEC 25012:2008 “Soft-ware engineering—Software product Quality Requirements and Evaluation (SQuaRE)—Data quality model” (<https://iso25000.com/index.php/en/iso-25000-standards/iso-25012>, accessed on 22 July 2025), these standards are a series of specifications and standards developed to ensure the quality of software engineering and to improve the efficiency and reliability of development. Part 12 of the model focuses on the quality of data as a component of a computer system and defines 15 quality characteristics of target data used by people and systems. The methodology introduces these quality characteristics and proposes data quality evaluation metrics applicable to software prototype systems oriented to requirements validation, which is based on the 15 data quality characteristics, and constructs a prototype data quality evaluation framework containing 10 evaluation dimensions and the corresponding 15 evaluation metrics. As shown in Table 1, the prototype data quality evaluation framework contains evaluation dimensions, evaluation indicators, hierarchical categorization, evaluation indicator weights, and indicator meanings.

Table 1. The complexity of requirements models.

Case Study	Actor	Use Case	SO	PO	Entity Class	Association	INV
ATM	2	6	16	118	2	2	3
CoCoME	3	16	43	326	13	20	10
ERPSystem	2	8	34	204	8	7	5
Examxx	4	8	21	153	7	9	3
JForum2	3	10	46	282	6	6	0
JPetStore	2	6	19	130	6	5	2
LibraryMS	7	19	45	535	11	17	25
LoanPS	5	10	35	206	8	8	12
SpringBlog	3	7	23	145	5	6	0
ZBlog	1	9	43	183	9	12	0
Sum	32	99	325	2282	75	92	60

The data quality evaluation framework proposed in this study divides the evaluation metrics system into two categories: numerical evaluation metrics (discrete quantization) and continuity evaluation metrics (semantic validation), which cover a total of 15 specific metrics.

The numerical indicators (nine items) include number of entities coverage, missing entity attributes, outlier rate, format consistency, units consistency, diversity, continuity, units consistency. scale size, duplication, and timeliness rate. The overall quality of the initial data generation is evaluated across ten inter-locking dimensions whose combined weights total 100%. **Completeness**, weighted at 15% overall, first verifies that every conceivable scenario is covered by ensuring the dataset contains a sufficient number of entities such as items, categories, suppliers and shoppers (10%), and second checks that no indispensable attributes—item name, price, stock quantity, etc.—are left empty, quantifying the exact percentage of missing values (5%). **Accuracy**, contributing 9%, measures realism by detecting illogical numeric outliers in fields like price and weight (7%) and by judging whether the entire body of data is truthful enough to mirror a real-world

case (2%). **Consistency**, worth 10%, enforces uniform representation of the same entity wherever it appears across tables or systems (6%), insists on a single, coherent format for all date-time stamps (2%), and guarantees that currencies, units of measure and sizes are standardized throughout (2%). **Relevancy** accounts for 10% and focuses on the correctness of the relationships that bind different entities—such as the link between a product and its supplier—through a relationship-accuracy rate. **Diversity**, also 10%, ensures the dataset spans a broad spectrum of merchandise categories, supplier profiles and shopper types so that downstream prototypes can confront varied scenarios. **Scalability** contributes 8%: 5% assesses whether the total dataset volume is large enough to emulate realistic scale, while 3% penalizes excessive duplication or clutter of non-essential records. **Expandability**, weighted at 5%, evaluates how readily the data schema can accommodate future additions or modifications. **Usability** (5%) gauges how easily analysts can query, read and interpret the data. **Timeliness** (5%) measures the freshness of the data by comparing the last update timestamp to the current date. Finally, **Security** (5%) inspects the dataset for any leakage of sensitive information such as private customer details and verifies that appropriate encryption or desensitization measures are in place.

These metrics are calculated in real time by an automatic statistical algorithm with the following core formula:

Number of Entity Coverage:

$$EC = \frac{N_{\text{gen_entities}}}{N_{\text{req_entities}}} \times 100\%$$

If the demand model defines 13 entity classes and generates 12 classes, then $EC = 92.3\%$.

Missing Entity Attributes:

$$MAR = \frac{\sum_{i=1}^n \delta_i}{\alpha} \times 100\%$$

In notation, δ_i represents the number of attributes that are absent (i.e., explicitly marked as missing) for the i -th entity instance, where α denotes the aggregate count of all attribute values present across the entire collection of entity instances. This formulation makes MAR invariant to the mixture of entity types and clearly quantifies the proportion of missing mandatory information.

Format Consistency (FC):

$$FC = \frac{N_{\text{compliant formats}}}{N_{\text{total fields}}} \times 100\%$$

The rest of the numerical metrics are calculated in real time by python (3.11.9) scripts integrated with the Pandas library.

Continuity metrics (six items) include truthfulness, entity consistency, relationship accuracy rate, expansion rate, queryability rate, and leakage rate. These metrics rely on manual expert/LLM validation mechanisms.

Truthfulness (TR):

$$TR = \frac{N_{\text{reasonable instances}}}{N_{\text{total instances}}} \times 100\%$$

This metric assesses the proportion of data that conform to domain common sense (e.g., whether the unit price of goods is in the reasonable range).

Entity Consistency (ENC):

$$ENC = 1 - \frac{N_{\text{Conflicting References}}}{N_{\text{Total References}}}$$

This metric assesses the consistency of cross-system identifiers (e.g., whether OrderID matches in the order and logistics systems).

Relationship Accuracy Rate (RA):

$$RA = \frac{N_{\text{compliant relationships}}}{N_{\text{total relationships}}} \times 100\%$$

This metric assesses the accuracy of relationships (e.g., verifying the rule that “each department must have at least five employees”).

Similar processes are used for expansion rate, queryability rate, and leakage rate.

Feedback Mechanism: According to the assessment results, users can clearly understand the current data quality and determine whether the secondary generation is needed, which can automatically adjust the parameters of the prompt generation and data generation model to achieve continuous optimization of data generation. The role of the data evaluation is not only to detect the data quality, but more importantly, to introduce an optimization mechanism in the data generation process, and to adjust the prompts generated by the large language model by interacting with the user, so as to make the output more in line with the business requirements, thus enhancing the intelligence level of data-driven requirement validation.

3.8. Data Post-Processing Tools

Due to the unstable characteristics of large language models, problems such as irregular format, non-compliance with business constraints, missing fields, etc. may occur in the generated data, so automated data validation and repair mechanisms are needed. In this paper, we design a format conversion tool and a data repair tool.

Format Conversion Tool: The tool is responsible for converting the generated data into the format required by the target system (e.g., YAML format, etc.), which can be easily and clearly read and modified. Through flexible conversion rules, the format conversion tool can meet the needs of different application scenarios.

Data Validation and Repair Tool: The generated data may have errors due to model limitations or instability, such as missing data values, data anomalies, data duplication, etc. The data validation and repair tool automatically detects and repairs errors in the data through predefined rules to ensure data accuracy and consistency.

4. Results

In order to validate the effectiveness of the methodology, a series of experiments have been designed with the aim of evaluating the efficiency of the methodology in prototype data generation, the quality of the data, and the impact on the requirements validation process. The experiments and tool demos have been posted on the webpage (<https://lemon-521.github.io/InitialGPTwebsite/initialgpt.html>, accessed on 22 July 2025). The dataset can be downloaded here (<https://ai4se.com/casestudy>, accessed on 22 July 2025).

4.1. Research Questions

To assess whether InitialGPT effectively addresses the difficulties of the above challenges, the following research questions are posed in this section to evaluate the capabilities and usability of the model.

RQ1: How effective is the InitialGPT approach proposed in this paper compared to the baseline of manually written data for requirements validation in traditional software prototypes?

For RQ1, by introducing four real-world software prototyping system cases [4–7,9], the time efficiency of using InitialGPT was experimented against the current baseline

approach. Time efficiency is a commonly recognized key metric in requirements validation evaluations [14].

RQ2: What is the quality of the prototype data generated by InitialGPT? For RQ2, we invited 10 testers to manually evaluate the quality of the generated initial data based on the evaluation metrics proposed in this paper.

RQ3: How effective is the prompt generation in this paper compared to ordinary prompts? This paper evaluated the effectiveness of the method by comparing the quality of the generated data for InitialGPT cue words with ordinary cue words.

RQ4: How effective is the multi-intelligent-agent workflow in the InitialGPT method proposed in this paper compared to the single-intelligent-agent baseline? For RQ4, this paper evaluated this by comparing the performance of multi-agent and single agent in terms of the quality of data generated for the prototype case.

RQ5: How does the InitialGPT method proposed in this paper compare with other data generation methods? This paper selected representative data generation methods and uses Wasserstein distance, Jensen–Shannon divergence (JSD), Pearson correlation, and Cramér’s V to assess the quality of data generation.

4.2. Case Studies

This section validates the effectiveness and capability of the InitialGPT approach through ten case studies. These case studies, which have been presented at ICSE 2019 [7] and ICSE 2022~2024 [9,46,47], cover systems that are widely used in various domains of daily life, including:

Supermarket System (CoCoME): A business scenario involving a supermarket cashier handling sales operations and a supermarket manager managing merchandise inventory. Library Management System (LibMS): A case study belonging to the education sector, mainly involving the borrowing and returning of books by students. Automated Teller Machine (ATM): A typical case study in the financial sector involving customer withdrawals and bookkeeping. Loan Processing System (LoanPS): Contains use cases for loan application submission, appraisal, new loan booking, and payment; blog systems (Spring-blog, Zb-blog), forum platforms (JForum2), e-commerce applications (JPetStore), online examination systems (Examxx), and ERPSystem are among the various software types.

The ten case studies are broadly representative, cover a wide range of industry sectors, and demonstrate the application of different requirements modeling and prototyping approaches. The complexity of the cases is measured based on factors such as the number of participants in the requirements model, use cases, system operations, entity classes, and their associative relationships. Table 1 shows the requirements complexity statistics for the four cases, which together contain 32 participants, 325 system operations, 99 use cases, 75 entities, and 92 entity class associations. SO and PO are the abbreviations of system and primitive operations, respectively. INV is the abbreviation of invariant. See our GitHub repository for the full case library and requirements modeling details (<https://github.com/maple-fans/CaseStudies>, accessed on 22 July 2025).

4.3. Experimental Setup

The experiment adopts a combination of quantitative and qualitative evaluation methods, and the experimental environment is as follows: Intel Core i5 2.8 GHz processor, DDR3 16 GB RAM, and JDK11 environment (JDK11 has been integrated into RM2PT). RM2PT [4,5,7] is the advanced prototyping method, which can quickly generate a prototype system through the requirement model and improve the efficiency of requirement confirmation.

RQ1: In order to evaluate the effectiveness of InitialGPT, this study invited 10 software engineers to participate in the experiment: 6 of them are master's degree students majoring in software engineering, and the other 4 are from Internet companies with 3 years of software development experience.

Before the start of the experiment, the subjects received a training session, which included the basics of requirements identification using CASE tools. We familiarized the participants with the construction of requirements models and the process of requirements identification using RM2PT, as well as the basic information of each case study. After the training, the participants were randomly divided into two groups for controlled experiments: manual preparation of prototype data for requirements validation, and automatic generation of prototype data for requirements validation using InitialGPT. In addition, this study also explored the effect of the amount of initial data on the efficiency of requirements validation, and the experiments were set up to compare and analyze 25, 100, and 300 pieces of initial data, respectively.

RQ2: Since the prototype data generated by the model is mainly used for requirements validation, its data generation method is different from that of software testing data. Therefore, a user study with 15 subjects was conducted to evaluate the quality of initial data generated by InitialGPT.

Ten of the subjects were master's degree holders in software engineering, and the other five were software engineers. First, this paper conducts experimental design based on the prototype data quality assessment metrics shown in Table 2. All the subjects experience the practical application of manually written prototype data and InitialGPT generated data in the requirement identification process. Subjects need to independently assess the quality of data, including the manually written prototype data and the InitialGPT-generated data.

In this paper, we first conducted systematic research on the current mainstream large language models and screened out GPT-4.5, GPT-4o, O1-preview and O3-mini, GPT-4, GPT-3.5-turbo, and Gemini-2.0-flash after comprehensively considering the performance of the models, the wide range of applications, and the degree of recognition. Claude-3.5-sonnet and Deepseek-r1 models were used as experimental subjects, and manually written data were used as benchmarks for comparative analysis.

Table 2. The time taken to complete the task.

Case Study	Manual			InitialGPT		
	25	100	300	25	100	300
ATM	6.30	24.25	71.01	0.45	3.57	10.47
CoCoME	5.47	33.75	87.58	0.42	1.88	22.50
LibMS	9.15	28.90	81.56	2.18	3.30	14.96
LoanPS	8.96	30.56	83.62	1.73	3.09	11.50
ERPSystem	6.01	23.48	65.73	0.83	2.01	10.21
Examxx	5.82	22.32	61.95	0.62	1.52	7.71
JForum2	7.25	28.56	79.14	1.14	2.82	14.14
JPetStore	4.67	17.95	50.11	0.53	1.30	6.60
SpringBlog	5.11	19.83	55.27	0.59	1.44	7.30
ZBlog	5.26	20.71	58.12	0.74	1.81	9.19
Average	6.40	25.03	69.41	0.92	2.27	11.46

The evaluation was conducted using the 5-Likert [48] evaluation for user research, in which the scoring criteria include: strongly agree (5 points), agree (4 points), neutral (3 points), disagree (2 points), and strongly disagree (1 point). In addition, to ensure the credibility of the experimental results, subjects were required to submit the evaluation results within 2 h. To ensure the accuracy of the experiment, the assessment results of three

independent experiments were counted in this study [49]. Each experiment was evaluated separately for the data generated by each model of manual writing and InitialGPT, and the average score of the three experiments was finally calculated as the final analysis result.

RQ3: The experiment aims to evaluate the difference in data generation quality between InitialGPT prompt (specially designed cue generation model) and ordinary prompt (directly written basic prompt) to measure the advantages of InitialGPT in prompt optimization.

The experimental object is the CoCoME supermarket system software case dataset, the requirements are confirmed by using ordinary prompt words and InitialGPT prompt words, and GPT-4 is adopted as the data generation model. The experiment is divided into two groups: the first group uses ordinary prompt words and directly inputs GPT-4 for data generation; the second group uses the prompt word template generated by InitialGPT and prompts GPT-4 to generate data. During the experiment, all generated data were based on the same demand model to ensure comparability. Ten subjects (six master's degree students and four software engineers) were invited to the study and independently scored using a five-point Likert scale, and the average quality scores of the data in the two groups were calculated to verify whether the prompt word templates could significantly improve the quality of the data generated through statistical analysis.

RQ4: This experiment compares the performance of the multi-agent workflow with the single-agent model in terms of data generation quality in order to assess the advantages of multi-agent workflow in data generation. The experiment, also based on the above case study, used GPT-4 as the data generation model and used the single agent and the multi-agent collaborative model for data generation, respectively. The single-agent mode is used by GPT-4 to complete all the data generation tasks independently, while the multi-agent mode is used by multiple agent to collaborate, including the data generation agent, data validation and cleaning agents, and data format output agent. The results of the experiments were independently evaluated several times and averaged to verify whether the multi-agent workflow outperforms the single-agent body in terms of data generation quality.

The total number of participants in the experiment was 35. The student group (15) consisted of 10 master's degree students in computer science (all with 2–3 years of experience in software development) and 5 undergraduate students in computer science (all having completed at least two software engineering courses and 1 year of development experience). The average age was 24.1 years (Standard Deviation = 1.8).

The professional group (20) were all working software engineers, recruited through an industry collaboration platform. The inclusion criteria were: at least 3 years of software development experience, more than 1 year of which had to involve a field directly related to the evaluation task (e.g., machine learning system development, API design, etc.). The average length of experience was 5.2 years, and the age range was 26–45 years (mean = 32.4, standard deviation = 4.3). All participants volunteered to participate in the experiment and signed an informed consent form. Familiarity with the assessment task was confirmed by a questionnaire prior to the experiment (all participants indicated “familiar” or “very familiar”).

RQ5: In this experiment, various types of representative data generation methods were selected, including data generation based on random selection methods (random selection generation, RSG), data generation based on statistical distributions (statistical distribution generation, SDG), and data generation methods based on deep learning (conditional tabular generative adversarial network, CTGAN), to be compared with the InitialGPT method for comparison.

The dataset is adopted from publicly available datasets covering various fields such as business, finance, and healthcare, including a supermarket sales dataset, loan approval dataset, adult census dataset, and medical cost personal dataset, which are widely used for scientific research and data analysis.

For generic datasets that lack formal requirement models (e.g., UCI adult income dataset), in order to evaluate the performance of InitialGPT on real datasets, this paper first reverse parses its metadata structure, extracts entity-attribute mapping relationships, and constructs a domain information format similar to the one extracted from the formal requirement model. Subsequently, it is injected into InitialGPT prompt templates to drive the large language model to generate data streams conforming to the domain characteristics. The generated results are evaluated against ISO quality standards together with the baseline method to ensure that the comparison experiments are conducted under a unified quality framework. The method solves the method adaptation problem under the scenario of missing formal requirements and provides theoretical feasibility proof for cross-domain data generation.

For data evaluation metrics, this paper uses four evaluation metrics, Wasserstein distance, Jensen–Shannon divergence (JSD), Pearson correlation, and Cramér’s V, to evaluate the quality of data generation. Wasserstein distance and Jensen–Shannon divergence are mainly used to assess the distribution similarity, which is categorized into continuous and discrete distributions. Pearson correlation and Cramér’s V are used to assess the data feature correlation, which is mainly categorized into numerical and categorical data evaluation.

Wasserstein distance is a metric that measures the difference between two probability distributions. It is based on the theory of optimal transmission and calculates the minimum amount of “work” required to “move” one distribution to another, and is used to assess the similarity of continuously distributed numerical data between the generated data and the real data. The equation is defined as follows:

$$W(p, q) = \inf_{\gamma \in \Pi(p, q)} \int_{\mathbb{R}^d \times \mathbb{R}^d} \|x - y\| d\gamma(x, y) \quad (1)$$

where p and q are the two probability distributions, $\Pi(p, q)$ is the set of all joint distributions whose marginals are p and q , and γ is a joint distribution that describes how the mass in p is transported to q . The distance $\|x - y\|$ represents the cost of moving a unit mass from x to y . The Wasserstein distance is the minimum total cost of transporting the mass from p to q over all possible joint distributions γ .

Jensen–Shannon divergence (JSD) scatter is a symmetric, non-negative measure of the similarity of categories of data in discrete distributions in the generated data. The JSD is defined as follows:

$$\text{JSD}(p, q) = \frac{1}{2} D_{\text{KL}}(p \parallel m) + \frac{1}{2} D_{\text{KL}}(q \parallel m), \quad (2)$$

where $m = \frac{1}{2}(p + q)$ is the average distribution of p and q , and $D_{\text{KL}}(p \parallel q)$ is the Kullback–Leibler divergence, given by

$$D_{\text{KL}}(p \parallel q) = \sum_x p(x) \log \frac{p(x)}{q(x)}. \quad (3)$$

The JSD has several desirable properties for our evaluation purposes. It is symmetric, meaning that $\text{JSD}(p, q) = \text{JSD}(q, p)$, and it is bounded between 0 and $\log 2$. The value of JSD is zero if and only if the two distributions are identical. This makes JSD a suitable metric for assessing the quality of generated data in comparison to real data.

Pearson's correlation coefficient (PCC) is used to measure the linear correlation between two variables and is used to evaluate the generation of continuous numerical patterns. For two random variables X and Y , the Pearson correlation coefficient is defined as follows:

To quantify the linear relationship between two continuous variables, we employ Pearson's correlation coefficient (PCC), denoted as P , which is a widely used statistical measure of the linear dependence between two variables. PCC provides a value between -1 and 1 , where 1 indicates a perfect positive linear relationship, -1 indicates a perfect negative linear relationship, and 0 indicates no linear relationship. The PCC is defined as follows:

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}, \quad (4)$$

where x_i and y_i are the individual sample points indexed with i , and $\bar{x}$ and $\bar{y}$ are the sample means of X and Y , respectively. The numerator represents the covariance between X and Y , while the denominator is the product of the standard deviations of X and Y .

Cramér's V is a statistic based on the chi-square test used to measure the strength of association between two categorical variables for category type variables. Cramér's V is particularly useful for analyzing contingency tables and provides a value between 0 and 1 , where higher values indicate stronger associations. Cramér's V is defined as follows:

$$V = \sqrt{\frac{\chi^2}{n \cdot \min(k-1, r-1)}}, \quad (5)$$

where χ^2 is the chi-square statistic, n is the total sample size, k is the number of columns, and r is the number of rows in the contingency table. The term $\min(k-1, r-1)$ is used to normalize the statistic, ensuring that the value of V lies within the interval $[0, 1]$.

In this study, we applied four different data generation methods to implement data generation operations on four specific datasets. Subsequently, the quality of the generated data is systematically compared and analyzed with that of the real data based on the four preset assessment indicators, aiming to comprehensively assess the effectiveness of the data generation methods and the reliability of the generated data.

4.4. Experimental Results

Experiment 1: It mainly consists of statistics on the time consumed by the two methods of requirements validation. According to the data in Table 2, when developers manually write data for 100 initial entities, the average time consumed is 25.03 min; while using InitialGPT to automatically generate data, it only takes 2.27 min, and with the increase in the number of initial data entities, the time gap between the two methods will be more obvious.

The "Manual" column shows the time taken to manually write the corresponding amount of initial data, while the "InitialGPT" column records the time taken to generate the data using InitialGPT (both in minutes).

By calculating the comparisons, Table 3 shows the time efficiency improvement brought about by utilizing InitialGPT for requirements validation. The results show that the InitialGPT approach can improve the efficiency of requirements validation by an average of 7.02 times compared to the traditional manual initial prototyping approach.

Table 3. The improvement efficiency of requirement validation.

Task	25	100	300	Average
Manual	6.40	25.03	69.41	
InitialGPT	0.92	2.27	11.46	
Improvement	5.96 times	10.03 times	5.06 times	7.02 times

Improvement = (Manual – InitialGPT)/InitialGPT.

Experiment 2: As shown in Table 4, this paper counts the results of three manual assessments of the quality of prototype data generated by each model. From the experimental data, the average data quality score of manually written data is 4.42, which is the best performance among all test subjects. The average data quality score of GPT-4-generated data is 4.33, which is the closest performance to the manual level, followed by GPT-4.5 with an average score of 4.30, which is also more outstanding. The average data quality score of Gemini-2.0-flash and o1- preview are at a medium level. Gpt-3.5-turbo has an average score of 4.02, which is an average performance. In summary, the comparative analysis reveals that GPT-4 and GPT-4.5 are more outstanding in terms of the quality of automatically generated prototype data, which are closest to the quality level of manually written data and have better practical value. In this experiment, despite the technical upgrades and optimizations of high-end models such as GPT-4.5, Deepseek-r1, O1-preview, and O3-mini, GPT-4 still outperformed these models in terms of data generation quality. This result may be attributed to a number of factors. First, it may be related to the complexity of the inference models. The O1-preview and O3-mini models may have over-complicated the inference process, leading to over-inference on specific tasks, which affected the data quality. In addition, the new versions of the models may not be as well adapted as expected on specific tasks, despite their optimization in other areas. This lack of fitness may lead to a decrease in the quality of the generated data, reflecting the delicate balance between model complexity and task matching.

Table 4. Quality evaluation of prototype data generation.

Task	First	Second	Third	Average
Manual	4.05	4.68	4.53	4.42
GPT-4	4.2	4.43	4.35	4.33
GPT-4.5	4.4	4.15	4.35	4.30
o1-preview	3.85	4.35	4.25	4.15
Gemini-2.0-flash	4.45	3.74	4.10	4.10
GPT-3.5-turbo	4.43	3.9	3.74	4.02
Deepseek-r1	4.17	3.92	3.85	3.98
GPT-4o	3.85	4.00	3.95	3.93
o3-mini	3.86	3.9	4.0	3.92
Claude-3-5-sonnet	3.75	3.85	3.68	3.76

Experiment 3: In Experiment 3, we compared the effect of an unadjusted ordinary prompt with the InitialGPT prompt. The results, as shown in Table 5, show that the prompt of the InitialGPT method significantly improve the data quality by 12.18%, which indicates that the method has high adaptability and reliability in prompt generation.

Table 5. Quality of data generated from different prompts.

Task	First	Second	Third	Average
Ordinary Prompt	3.87	4.05	3.67	3.86
InitialGPT Prompt	4.2	4.43	4.35	4.33
Improvement	8.53%	9.38%	18.53%	12.18%

Experiment 4: Table 6 shows the results of Experiment 4, where “Single” denotes single-agent and “Multi” denotes multi-agent. The results show that the multi-intelligence workflow significantly outperforms the single-intelligence mode in terms of data generation quality, with an average improvement of 12.38%.

In several evaluations, the multi-agent performs well, especially in complex tasks, and its advantage is more obvious, indicating that multi-agent collaboration can effectively improve the quality of data generation. However, in the experimental process, the generation efficiency of multi-agent is not as good as that of a single agent, and more time and resources are needed to coordinate the collaboration of different agents, which suggests that multi-agent collaboration has high application value when pursuing high-quality data generation but needs to be weighed against the efficiency.

Table 6. Comparison of data generation model agent.

Task	Exp 1	Exp 2	Exp 3	Exp 4	Exp 5	Average
Single	3.89	4.15	4.18	3.96	4.03	4.04
Multi	4.65	4.42	4.72	4.34	4.56	4.54
Improvement	19.54%	6.51%	12.92%	9.59%	13.15%	12.38%

Experiment 5: Table 7 shows the results of Experiment 5. In this comparison experiment, the InitialGPT method demonstrated its significant advantages in the similarity assessment between generated and real data, especially in the distributional similarity and variable correlation between generated and real data. The experimental results show that the Jensen–Shannon divergence (JSD) value of InitialGPT is only 0.001, which is much lower than that of other methods, indicating that its generated data are highly consistent with the real data in terms of distribution. In addition, the Cramér’s V value of InitialGPT is 0.277, which is higher than that of other methods, indicating that the data it generates are not only similar in distribution, but also better able to reflect the characteristics of the real data in terms of the relationships between variables. This ability is particularly important in tasks that require preserving the internal structure and relationships of the data, and thus InitialGPT has a significant advantage in generating high-quality synthetic data that can provide more representative and usable data for a variety of application scenarios.

InitialGPT’s performance in terms of Pearson correlation coefficients is relatively weak, with a value of 0.174, which is lower than that of the other methods. PCCs are used to measure the linear correlation between two variables, and low PCCs values indicate that InitialGPT-generated data may not fully reproduce the real data with respect to some of the linear relationships. The low PC value indicates that the data generated by InitialGPT may not fully reproduce the characteristics of the real data in some linear relationships. This shortcoming may stem from the fact that InitialGPT focuses more on the overall distribution and complex relationships between variables at the expense of the accuracy of some linear correlations. However, this shortcoming may not be significant in practical applications, especially in scenarios where more attention is paid to data distribution and non-linear relationships. Therefore, this limitation of InitialGPT does not detract from its overall advantage in generating high-quality synthetic data.

Table 7. Comparison of data generation.

Task	RSG	SDG	CTGAN	InitialGPT
Wasserstein Distance	3.85	4.15	3.78	3.96
JSD	0.121	0.002	0.012	0.001
PCCs	0.480	0.472	0.485	0.174
Cramér's V	0.047	0.094	0.218	0.277

Statistical Analysis: This study validated the comparability of the LLM-based data generation method with manually prepared data in terms of quality assessment through systematic analysis. As shown in Table 8, intra-group correlation coefficient analysis revealed some subjective differences among raters ($ICC(1) = 0.340$), but the key model comparison revealed that the quality ratings of the proposed LLM-generated method (GPT-4.5) were highly similar to those of manually-written data (Manual) (4.30 vs. 4.42, with a difference of <3%). The standard deviation analysis of each model further revealed that the LLM generation method (GPT-4.5: $SD = 0.132$) had a similar pattern of rater variability to the manually written data (Manual: $SD = 0.329$), suggesting that the two were comparable in terms of perceived quality.

The analysis of variance (ANOVA) results ($F(9,20) = 0.014$, $p = 0.986$) confirmed that there were no statistically significant differences between all models, which supported the hypothesis of equivalence between LLM-generated and manually written data in terms of quality assessment. Notably, the proposed LLM method (GPT-4.5) outperformed the manual data ($SD = 0.329$) in terms of rater agreement ($SD = 0.132$), suggesting more stability in quality evaluation. Although the highest mean score (4.42) was obtained for the manually prepared data, its small difference ($\Delta = 0.12$) with the LLM method (GPT-4.5: 4.30) did not reach the level of statistical significance ($p > 0.05$), which verified the validity of the LLM-generated method for approaching the manual level of quality dimensions.

Table 8. Comparison of LLM-generated and manual data quality.

Metric	Manual	GPT-4.5	Δ	Statistical Significance
Mean Score	4.42	4.30	−0.12	$p = 0.317$
Rater SD	0.329	0.132	−0.197	–
Score Range	4.05–4.68	4.15–4.40	–	–
Gap to Optimal	0.00	0.12	–	$p > 0.05$

Independent t -test ($t = 1.08$, $df = 4$, $p = 0.317$). SD: standard deviation.

5. Discussion and Limitations

5.1. Discussion

Generality: The InitialGPT method proposed in this paper can automatically generate initial data for software prototypes through the requirement model, but InitialGPT is not limited to this. It can also be used to assist in manual requirements identification based on other domain knowledge, and the InitialGPT method can generate corresponding prototype data for the corresponding domain information, which is advantageous to enhance its generality in the software development process. The InitialGPT method can generate the corresponding prototype data for it as long as the corresponding domain information is given, which is favorable to improve its generality in the software development process.

InitialGPT vs. Traditional Manual: The evaluation results show that the data generated by InitialGPT exhibit comparable competitiveness to manual data, although its performance is slightly lower than that of well-constructed manual data. However, considering the high cost of manual methods in terms of time and labor costs of hiring domain experts, InitialGPT offers an efficient and economical alternative. Therefore, this study, on the

one hand, aims to continuously improve the reasonableness of InitialGPT data based on the data quality model, and on the other hand, asserts that InitialGPT is not intended to completely replace manual conceptualization of input data, but rather to serve as an auxiliary user interaction tool. To this end, a series of user interaction and feedback mechanisms are designed to explore the effective combination of InitialGPT generation and manual feedback, so as to optimize the efficiency and quality of requirements identification in the software prototyping process.

5.2. Threats to Validity

Internal Threats: When assessing the validity of this work, internal threats mainly stem from the fact that the assessment results may be influenced by subjective judgment. Since there is no unified data assessment index in the field of requirements identification, to address this challenge, this paper constructs a software prototype data assessment benchmark for requirements identification based on the international standard ISO/IEC 25012:2008, but the assessment results of data quality may still be biased due to the subjectivity of the user's research. This manual assessment process inevitably has errors. It is worth noting, however, that any assessment effort that relies on manual judgment faces similar challenges. To minimize this threat, this study took a twofold approach: first, the source of manual data versus InitialGPT-generated data was hidden from the assessment process, and objective evaluation of data quality was ensured through multiple assessments; and second, a checking team was established to ensure agreement between the different scenarios. Future research will further explore automated assessment methods to improve the accuracy of demand confirmation.

External Challenges and Coping Strategies: The main external constraints faced by the study stem from the closed-source property of the large language model. Although InitialGPT adopts a multi-model fusion strategy, some of its components (e.g., the GPT family of models) are still non-open-source architectures. However, the generation strategy optimization scheme proposed in this study is specifically designed for the characteristics of large language models, and introduces the current mainstream multi-agent workflow architecture in the evaluation system, and the prompt generation and data preprocessing process of this scheme show strong universality, with strong migration capability, and can be adapted to other language models deployed locally. In this study, when the data generation framework of InitialGPT is ported to different local models such as Llama2-13B, etc., it can maintain stable performance output without large-scale adjustments, which further validates the versatility of the approach. Future research will further enhance the quality of data generation through domain knowledge base and knowledge graph.

5.3. Method Portability and Future Work

We conducted preliminary experiments, firstly, by counting the average values of all model generation performances in Table 4, and secondly, by conducting three experiments on the local open-source model LLaMA2-13B. The results of the experiments are shown in Table 9, which show that when we migrate the InitialGPT framework to a locally deployed open-source model (e.g., LLaMA2-13B), the system still maintains a small difference in the high-performance model's data generation performance without significant adjustments to the prompting structure or process. This further validates the portability of our approach.

However, it must be pointed out that the performance of the closed-source model and open-source model in data generation task has not been systematically compared yet. Therefore, in future work, we plan to carry out comparative experiments for different open-source and local models under the unified evaluation framework, and perform quantitative analyses from multiple dimensions such as generation quality, efficiency, and stability.

In addition, we will further explore the introduction of domain knowledge base and knowledge map to improve the semantic consistency and business relevance of generated data. This not only helps to make up for the deficiency of the open-source model in domain understanding, but also provides a possible path for building higher-quality prototype system data.

Table 9. Quality evaluation of prototype data generation.

Task	First	Second	Third	Average
Models	-	-	-	4.05
LLaMA2-13B	3.83	3.62	3.71	3.72

Models Indicates the average performance of the model.

To summarize, InitialGPT has good versatility and migration ability in method design and can adapt to various large language model environments including open-source models. In the future, we will continue to promote the research on model adaptation and performance comparison, and further enhance the practicability and scalability of the method.

5.4. Limitations of InitialGPT in Unstructured Requirements Scenarios

InitialGPT faces the following main limitations in handling unstructured requirements:

(1) Ambiguity and inconsistency in unstructured input: InitialGPT relies on structured or semi-structured prompts to generate prototype data. When processing highly ambiguous, inconsistent, or incomplete unstructured requirements (such as free-form user stories, informal notes, or vague stakeholder descriptions), the model may struggle to accurately interpret their semantics, resulting in low-quality or irrelevant data. (2) Lack of domain context: Unstructured requirements often lack explicit domain knowledge or context. Without sufficient background information, data generated by InitialGPT may be syntactically correct but semantically inconsistent with actual business logic or domain constraints. (3) Dependence on prompt quality: The effectiveness of InitialGPT highly depends on the clarity and completeness of the input prompts. In unstructured scenarios, designing effective prompts becomes more challenging and may require additional preprocessing or human intervention. (4) Human involvement required: To mitigate the above issues, unstructured requirements typically require human involvement for clarification, refinement, or verification, which may reduce the level of automation that InitialGPT aims to achieve.

We believe that acknowledging these limitations helps to better understand the applicability and limitations of our approach. Although InitialGPT has certain limitations when handling unstructured requests, we are also exploring a strategy that combines reverse extraction and template filling. Specifically, the system can first identify key entities (e.g., products, users, orders) and their attributes (e.g., price, inventory, status) from unstructured text, then automatically fill this information into predefined data generation templates. In this way, even when input requirements are unclear or expressed in an unstructured manner, the system can still effectively generate structured prototype data that aligns with business logic. In future work, to systematically address unstructured requirements, we plan to integrate advanced NLP techniques, including named entity recognition (NER) and relation extraction (RE) models, to automate the identification of key entities and their relationships. Additionally, we will design a semi-automatic template mapping mechanism that combines initial automated extraction with user calibration. This iterative approach will allow for progressive refinement of the system's accuracy and adaptability in processing unstructured inputs.

6. Conclusions

With the rapid development of information technology, the requirements of software systems are becoming increasingly complex and variable, which makes requirements validation more and more important in the software development process. The development of model-driven development and artificial intelligence technology has brought new opportunities for software engineering. This study proposes InitialGPT, a prototype data generation method for requirements validation, to overcome the bottleneck in the traditional prototype data generation process. The method reduces manual intervention by automating the generation of requirements-compliant prototype data. Experimental results show that InitialGPT is able to improve the efficiency of requirements identification by 7.02 times compared with the traditional method, while the quality of the generated data (4.33) is close to the level of manually authored data (4.42), which proves the advantages of InitialGPT in improving the efficiency and accuracy of requirements identification in the field of software engineering.

More methods (e.g., domain knowledge graph combination, multimodal data fusion) will be further explored in future work to generate initial data with higher quality. In addition, this paper will also explore the potential applications of InitialGPT in other domains, such as intelligently generating input data for prototyping system operation during the requirements validation process, to further promote the automation and intelligence of the software development process.

Author Contributions: Conceptualization, J.G.; Methodology, S.C., J.G. and W.W.; Software, S.C.; Validation, S.C., J.G. and W.W.; Investigation, S.C.; Resources, W.W.; Writing—original draft, S.C.; Writing—review & editing, W.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Guangdong Provincial Core Software Tackling Project of China (No. 2024QT022) and the Heilongjiang Provincial Natural Science Foundation of China (No. LH2024F005).

Data Availability Statement: The data used in this study were obtained from a publicly available dataset, which is available through its repository <https://github.com/maple-fans/CaseStudies> (accessed on 22 July 2025), and further information can be obtained from the first or corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Sutcliffe, A.G.; Economou, A.; Markis, P. Tracing requirements errors to problems in the requirements engineering process. *Requir. Eng.* **1999**, *4*, 134–151. [\[CrossRef\]](#)
2. Hofmann, H.F.; Lehner, F. Requirements engineering as a success factor in software projects. *IEEE Softw.* **2001**, *18*, 58. [\[CrossRef\]](#)
3. Sommerville, I. *Software Engineering*, 9/E; Pearson Education Inc.: Saddle River, NJ, USA, 2011.
4. Leopold, H.; Mendling, J.; Polyvyanyy, A. Supporting process model validation through natural language generation. *IEEE Trans. Softw. Eng.* **2014**, *40*, 818–840. [\[CrossRef\]](#)
5. Yang, Y.; Li, X.; Ke, W.; Liu, Z. Automated prototype generation from formal requirements model. *IEEE Trans. Reliab.* **2019**, *69*, 632–656. [\[CrossRef\]](#)
6. Yang, Y.; Ke, W.; Li, X. RM2PT: Requirements validation through automatic prototyping. In Proceedings of the 2019 IEEE 27th International Requirements Engineering Conference (RE), Jeju Island, Republic of Korea, 23–27 September 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 484–485.
7. Yang, Y.; Li, X.; Liu, Z.; Ke, W. RM2PT: A tool for automated prototype generation from requirements model. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Montreal, QC, Canada, 25–31 May 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 59–62.

8. Erichsen, J.F.; Sjöman, H.; Steinert, M.; Welo, T. Protobooth: Gathering and analyzing data on prototyping in early-stage engineering design projects by digitally capturing physical prototypes. *Artif. Intell. Eng. Des. Anal. Manuf.* **2021**, *35*, 65–80. [CrossRef]
9. Chang, S.; Gao, J.; Yang, Y. InputGen: A Tool for Automatic Generation of Prototype Inputs to Support Rapid Requirements Validation. In Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Melbourne, Australia, 14–20 May 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 122–126.
10. Penedo, M.H. Prototyping a project master database for software engineering environments. In Proceedings of the Second ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, New York, NY, USA, 9–11 December 1986; SDE 2; pp. 1–11. [CrossRef]
11. Figma Community. How to Use Figma for Prototyping AI Features in Android Apps? Available online: <https://forum.figma.com/chat-and-connect-27/how-to-use-figma-for-prototyping-ai-features-in-android-apps-42586> (accessed on 22 July 2025).
12. Wang, J.; Huang, Y.; Chen, C.; Liu, Z.; Wang, S.; Wang, Q. Software Testing With Large Language Models: Survey, Landscape, and Vision. *IEEE Trans. Softw. Eng.* **2024**, *50*, 911–936. [CrossRef]
13. Sousa, A.E.; Cardoso, P. Use of Generative AI by Higher Education Students. *Electronics* **2025**, *14*, 1258. [CrossRef]
14. Liu, X.; Yin, J.; Liu, J.; Zhang, S.; Xiao, B. Time Efficient Tag Searching in Large-Scale RFID Systems: A Compact Exclusive Validation Method. *IEEE Trans. Mob. Comput.* **2022**, *21*, 1476–1491. [CrossRef]
15. Štrimaitis, R.; Ramanauskaitė, S.; Stefanovič, P. Automated Selection of Time Series Forecasting Models for Financial Accounting Data: Synthetic Data Application. *Electronics* **2025**, *14*, 1253. [CrossRef]
16. Lichter, H.; Schneider-Hufschmidt, M.; Zullighoven, H. Prototyping in industrial software projects-bridging the gap between theory and practice. *IEEE Trans. Softw. Eng.* **1994**, *20*, 825–832. [CrossRef]
17. Nuseibeh, B.; Easterbrook, S. Requirements engineering: A roadmap. In Proceedings of the Conference on the Future of Software Engineering, Limerick, Ireland, 4–11 June 2000; pp. 35–46.
18. Erfurth, I.; Rossak, W. CUTA4UML: Bridging the gap between informal and formal requirements for dynamic system aspects. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2, Cape Town, South Africa, 1–8 May 2010; pp. 171–174.
19. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 1–11.
20. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
21. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog* **2019**, *1*, 9.
22. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1877–1901.
23. Clark, E.; August, T.; Serrano, S.; Haduong, N.; Gururangan, S.; Smith, N.A. All that’s human is not gold: Evaluating human evaluation of generated text. *arXiv* **2021**, arXiv:2107.00061.
24. Dou, Y.; Forbes, M.; Koncel-Kedziorski, R.; Smith, N.A.; Choi, Y. Is GPT-3 text indistinguishable from human text? SCARECROW: A framework for scrutinizing machine text. *arXiv* **2021**, arXiv:2107.01294.
25. Zhou, J.; Zhang, Y.; Luo, Q.; Parker, A.G.; De Choudhury, M. Synthetic lies: Understanding ai-generated misinformation and evaluating algorithmic and human solutions. In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, Hamburg, Germany, 23–28 April 2023; pp. 1–20.
26. Wei, J.; Bosma, M.; Zhao, V.Y.; Guu, K.; Yu, A.W.; Lester, B.; Du, N.; Dai, A.M.; Le, Q.V. Finetuned language models are zero-shot learners. *arXiv* **2021**, arXiv:2109.01652.
27. Hämäläinen, P.; Tavast, M.; Kunnari, A. Evaluating large language models in generating synthetic hci research data: A case study. In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, Hamburg, Germany, 23–28 April 2023; pp. 1–19.
28. McNutt, A.M.; Wang, C.; Deline, R.A.; Drucker, S.M. On the design of ai-powered code assistants for notebooks. In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, Hamburg, Germany, 23–28 April 2023; pp. 1–16.
29. Wang, C.; Yang, Y.; Gao, C.; Peng, Y.; Zhang, H.; Lyu, M.R. No more fine-tuning? An experimental evaluation of prompt tuning in code intelligence. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Singapore, 14–18 November 2022; pp. 382–394.
30. Liao, J.; Zhao, X.; Zheng, J.; Li, X.; Cai, F.; Tang, J. PTAU: Prompt Tuning for Attributing Unanswerable Questions. In Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, 11–15 July 2022; pp. 1219–1229.
31. Zhou, K.; Yang, J.; Loy, C.C.; Liu, Z. Learning to prompt for vision-language models. *Int. J. Comput. Vis.* **2022**, *130*, 2337–2348. [CrossRef]

32. Wu, H.; Ma, B.; Liu, W.; Chen, T.; Nie, D. Fast and constrained absent keyphrase generation by prompt-based learning. In Proceedings of the AAAI Conference on Artificial Intelligence, online, 22 February–1 March 2022; Volume 36, pp. 11495–11503.
33. Seaman, C.B. Qualitative methods in empirical studies of software engineering. *IEEE Trans. Softw. Eng.* **1999**, *25*, 557–572. [CrossRef]
34. Cui, G.; Hu, S.; Ding, N.; Huang, L.; Liu, Z. Prototypical verbalizer for prompt-based few-shot tuning. *arXiv* **2022**, arXiv:2203.09770.
35. Ge, S.; Parikh, D. Visual conceptual blending with large-scale language and vision models. *arXiv* **2021**, arXiv:2106.14127. [CrossRef]
36. Cantino, A. Prompt Engineering Tips and Tricks with gpt-3. 2016. Available online: <https://blog.andrewcantino.com/blog/2021/04/21/prompt-engineering-tips-and-tricks/> (accessed on 22 July 2025).
37. Forward, A.; Badreddin, O.; Lethbridge, T.C.; Solano, J. Model-driven rapid prototyping with Umple. *Softw. Pract. Exp.* **2012**, *42*, 781–797. [CrossRef]
38. Liu, P.; Yuan, W.; Fu, J.; Jiang, Z.; Hayashi, H.; Neubig, G. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.* **2023**, *55*, 1–35. [CrossRef]
39. Wang, S.; Liu, Y.; Xu, Y.; Zhu, C.; Zeng, M. Want to reduce labeling cost? GPT-3 can help. *arXiv* **2021**, arXiv:2108.13487. [CrossRef]
40. Ding, B.; Qin, C.; Liu, L.; Chia, Y.K.; Joty, S.; Li, B.; Bing, L. Is gpt-3 a good data annotator? *arXiv* **2022**, arXiv:2212.10450.
41. Dai, H.; Liu, Z.; Liao, W.; Huang, X.; Wu, Z.; Zhao, L.; Liu, W.; Liu, N.; Li, S.; Zhu, D.; et al. Chataug: Leveraging chatgpt for text data augmentation. *arXiv* **2023**, arXiv:2302.13007.
42. Kumar, V.; Choudhary, A.; Cho, E. Data augmentation using pre-trained transformer models. *arXiv* **2020**, arXiv:2003.02245.
43. Chung, J.J.Y.; Kamar, E.; Amershi, S. Increasing Diversity While Maintaining Accuracy: Text Data Generation with Large Language Models and Human Interventions. *arXiv* **2023**, arXiv:2306.04140. [CrossRef]
44. Sahu, G.; Rodriguez, P.; Laradji, I.H.; Atighehchian, P.; Vazquez, D.; Bahdanau, D. Data augmentation for intent classification with off-the-shelf large language models. *arXiv* **2022**, arXiv:2204.01959. [CrossRef]
45. Ye, J.; Gao, J.; Li, Q.; Xu, H.; Feng, J.; Wu, Z.; Yu, T.; Kong, L. Zeroshot: Efficient zero-shot learning via dataset generation. *arXiv* **2022**, arXiv:2202.07922.
46. Bao, T.; Yang, J.; Yang, Y.; Yin, Y. RM2Doc: A tool for automatic generation of requirements documents from requirements models. In Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings, Pittsburgh, PA, USA, 21–29 May 2022; pp. 188–192.
47. Pan, H.; Yang, Y. ValidGen: A Tool for Automatic Generation of Validation Scripts to Support Rapid Requirements Validation. In Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, Lisbon Portugal, 14–20 April 2024; pp. 124–128.
48. Jordan, P.W.; Thomas, B.; McClelland, I.L.; Weerdmeester, B. *Usability Evaluation in Industry*; CRC Press: Boca Raton, FL, USA, 1996.
49. Li, Z.; Zhu, H.; Lu, Z.; Yin, M. Synthetic Data Generation with Large Language Models for Text Classification: Potential and Limitations. *arXiv* **2023**, arXiv:2310.07849. [CrossRef]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.