ingegneria sismica
International Journal of Earthquake Engineering
Anno XLIII - Num. 3 - 2026

FlowMock: A gRPC-based Process-Aware Service Simulator

Chunyan Yang[1], *Jinran Du*[1], *Haonan Liu*[1,*], *Miaopeng Yu*[1], *Tao Dai*[1], *Chuanmao Xu*[1] *and Kequan Lin*[2]

[1] China Southern Power Grid Electric Power Research Institute, Guangzhou 510663, Guangdong, China
[2] China Southern Power Grid, Guangzhou 510663, Guangdong, China

SUMMARY: *The study examines important obstacles to external service simulation in microservice architectures and it is identified that existing simulators lack the ability to simulate complex business processes and exceptions. To address these limitations, a process-aware service simulator, FlowMock, is proposed. FlowMock is implemented on the open-source Remote Procedure Call (RPC) framework gRPC and supports tight integration between process mining algorithms and the large language model DeepSeek. On this basis, a fully automated pipeline is constructed, covering event-log–based process discovery, interface and response configuration, and test-client code generation. Experiments on multiple real-world and synthetic log datasets show that the simulator achieves high process fidelity and stable response behavior. These results suggest that FlowMock constitutes a new technical framework for automated testing and service verification in microservice-based systems.*

KEYWORDS: *process-aware; service simulator; gRPC; business process mining; AI-driven code generation; automated testing*

1 Introduction

Microservice architectures have become widely used in enterprise systems, and this has led to increasing workload and complexity in service construction and verification. In such architectures, applications are divided into multiple autonomous components, which improves flexibility in deployment and scaling. However, these components span more runtime boundaries, making dependency paths and interaction patterns more diverse and difficult to predict.

In practical testing, if the controllability and realism of external service interactions can be improved and real service behavior can be simulated at a low cost, the efficiency of testing and iterative development will also benefit.Traditional service simulation tools mainly rely on static configurations or simple rule-based responses and typically support mocking and testing only a single interface at a time, making it difficult to cover complete business processes involving multiple stages and complex logic. Similar limitations have also been observed in log-based business process simulation, where purely data-driven approaches struggle to reproduce realistic process behaviour and temporal dynamics in complex processes [1].

In real-world business scenarios in the power industry, edge collaboration for smart grids and multi-priority service queuing have become common patterns. In such scenarios, interface-level stubs alone cannot reproduce end-to-end process behavior that includes

*527484533@qq.com
https://doi.org/10.65102/is20261232
Published: 10 June, 2026

priority-based preemption and dynamic scheduling, which makes accurate evaluation of service-level agreements and resource allocation more difficult [2]. Therefore, a process-oriented service simulation framework is practically necessary to cover complex paths, temporal constraints, and priority-aware interactions.

In recent years, advances in business process mining and artificial intelligence (AI) have created new opportunities for the intelligent evolution of service simulation. According to the Process Mining Handbook, process mining can automatically reconstruct business process models from event logs and support a wide range of analysis tasks, including automated process discovery, conformance checking, and performance analysis [3]. On this basis, process mining results can be exploited as precise process foundations for simulation tools, while AI techniques can further assist in the automatic generation of interface definitions and process-related test data, thereby improving the realism and coverage of test scenarios.

However, most current research and tools still remain at the interface level and lack automated and intelligent simulation capabilities for entire business processes. In many real development environments, dependencies among microservices are complex and constantly changing, whereas traditional service simulators are usually based on static configurations or simple rules to emulate service responses. Static simulation often fails to represent operative business behaviours under exceptional situations or rapid change; as a result, both the accuracy and the coverage of simulated outputs decline. Most existing service simulators focus on interface inputs and outputs while paying little attention to how the underlying business process actually runs. Once these process constraints are ignored, the generated test data can hardly reflect key aspects such as what needs to happen first, what comes next, and under which conditions different paths are taken. Without this sense of process, simulated interactions with external services gradually drift away from the real system, which in turn limits our ability to improve the stability and development efficiency of microservice systems. Even if we run many tests, it's hard to cover all of the important business scenarios and edge cases.

To solve this problem, service simulators evolve to be smarter and more automated. It's not a library of hand-written stubs, it learns the business processes by looking at runtime data, and uses artificial intelligence to create them automatically. The study by Camargo et al. found that as the simulation model is created based on the process structures that are mined out of the event logs and is combined with advanced modeling techniques the simulation result would be closer to the behavior that can be seen in the real log. And the fact is, if its data-driven and is sensitive enough for structure, a simulation can be a better fidelity test environment. [4].

Building on this insight, this paper introduces FlowMock, a gRPC based service simulator that embeds process awareness directly into automated test generation. Its central idea is to let process mining algorithms reconstruct how the real business actually runs, and then have an AI driven code generation pipeline realize these processes as gRPC interface definitions and simulation logic. As a result, simulations are no longer designed by guesswork; they are produced automatically around the actual operational state of the business. The system consists of three closely coordinated stages.

FlowMock applies the Split Miner algorithm to business event logs to extract an executable process model and then converts it into BPMN form. Through this step, the system recovers from the logs which steps can run in parallel, which must be executed in sequence, and under what conditions different branches are taken [5]. In other words, the simulations generated later are constrained from the outset by the real process structure.

Once the process model is available, an AI module based on the deepseek v3 language model takes over. It automatically generates gRPC service specifications (.proto files),

corresponding stub data, and client test code. Because these artifacts are jointly derived from the process model and the event logs rather than written manually one by one, they naturally cover more invocation paths that have actually occurred and more easily preserve semantic consistency between requests and responses. At the same time, the portion that developers need to maintain by hand is greatly reduced, and the reachable test space expands significantly.

After the process model and simulation configuration are in place, FlowMock synthesizes runtime simulation data according to this information so that its behavior closely follows the real call traces in the event logs. For different test scenarios, the system can adjust specific responses while remaining within the same set of process constraints. In this way it maintains high accuracy and still supports abnormal and boundary scenarios with good flexibility. The result is a simulation capability that runs like the real business while remaining easy to tune and extend.

Experiments conducted on both real and synthetic datasets show that FlowMock performs well in terms of process fitness and response behavior stability. It can faithfully reproduce complex business scenarios and thus provides a practical and effective solution for automated testing and service verification in microservice ecosystems.

The remainder of this paper is organized as follows. **Section 2** reviews related work on process mining and service simulation. **Section 3** describes the methods used by FlowMock to identify control-flow structures and generate BPMN models. **Section 4** presents the automated code-generation pipeline for producing gRPC interfaces, JSON-based stub configurations, and Go test clients. **Section 5** reports the experimental evaluation and discusses the results. **Section 6** concludes the paper and outlines promising directions for future work.

2 Related Work

Business process mining has emerged as a specialised area that investigates how operational knowledge can be extracted from event data. Early work by van der Aalst established a tight coupling between event data and business processes, clarifying both the theoretical underpinnings of process mining and its role within management information systems [6]. Dakic et al. define process mining as a systematic approach to using event logs created by information systems to discover, monitor, and improve the performance of business processes [7]. Park and Kang build on this view to show how process mining related techniques support process innovation and large scale-organiztional design. The process mining handbook continues to map out the development of the field with the description of main research directions such as automation of process discovery, conformance checking, social network analytics, and building of simulation models; and the positioning of process mining in-process optimization frameworks [8]. While these are good developments, there are still many unresolved challenges, such as how to correctly “cleanse” event-data and identify events, how to handle heterogeneous and noisy logs in concurrent environments and the creation of widely adopted benchmark datasets.

At the algorithmic level, current process discovery algorithms mainly include algorithms for direct-follows relations, heuristic inference-based algorithms, and various algorithms of computational intelligence. They usually want something that is pretty fit, but precise and not overly complicated with the model. But its performance is usually terrible if the underlying processes are highly variable or if the event logs are noisy. A lot of researchers have used process mining for simulating analysis and optimization. Martin et al. carried out a systematic review on methods that use process mining for business process simulation model building

[10]. Camargo et al. built simulation models directly from the execution data and suggested measurement approaches to evaluate the degree of simulation fidelity [11]. To reduce manual intervention in process improvements, Soliman et al. explored integrating process mining with reinforcement learning and showed that hybrid methods could work in complicated and changing environments by adjusting to process changes [12].

Process view from the process-management view. As Kokala states, the traditional BPM methods had been relying heavily on a practitioner's intuition and static view of describing the process and the previous assumptions about how to finish the work. To solve these limitations, this study proposes embedding process mining in day-to-day manager behavior and enabling governance decisions to be data-driven and trackable [13]. Building on this, vom Brocke et al. created a five-layered research framework covering technologies, individuals, teams, organizations, and ecosystems, that systematically classifies process mining applications in enterprise settings [14]. Mamudu et al. from an empirical point of view identified the most critical factors for process mining projects and looked at the interdependencies between the factors [15]. In the more elaborate review of Reinkemeyer et al., the authors stress that instead of discovery of the models, we move from them to supporting the real-time execution and operational decision-making. He emphasizes that process mining must be performed directly on the running information systems [16].

In software engineering, mocking is a class of testing techniques in which fake components are created for the purpose of making the dependent units of code testable in isolation. Freeman et al. proposed the “mock roles, not objects” principle, which offers a theoretical foundation for responsibility-centered mocking, emphasizing that tests should capture what they expect from behavioral collaborations, not the implementation details of concrete classes [17]. The JMock framework's small API size and runtime generation of mock objects line up well with this view and support an easy test-driven development for object-oriented projects[18]. Empirical studies done by Spadini et al. show that developers use mocks when they encounter hard-to-test dependencies like remote services or tightly coupled components [19]. They do more research on the Java ecosystem afterwards to study how mocks develop through a system's life cycle and which kinds of dependencies occur most often in big code bases [20]. After the analysis of over 600 Mock Classes, Pereira and Hora also found out that mocks are mostly used to emulate Domain Object, External Dependencies, and Web Services; most mocks also are part of Inheritance Hierarchies [21].

These empirical findings collectively suggest that mocks are an effective mechanism for isolating complex or hard-to-test dependencies, which is closely related to the design of process-aware service simulators such as FlowMock that must emulate external services while preserving realistic business behaviour.

3 Proposed Method

The process-aware service simulation framework FlowMock proposed in this paper is shown in Fig. 1. It primarily consists of three core components: process control-flow discovery and modeling, automatic generation of gRPC-based mock services, and dynamic response with end-to-end process-level testing.

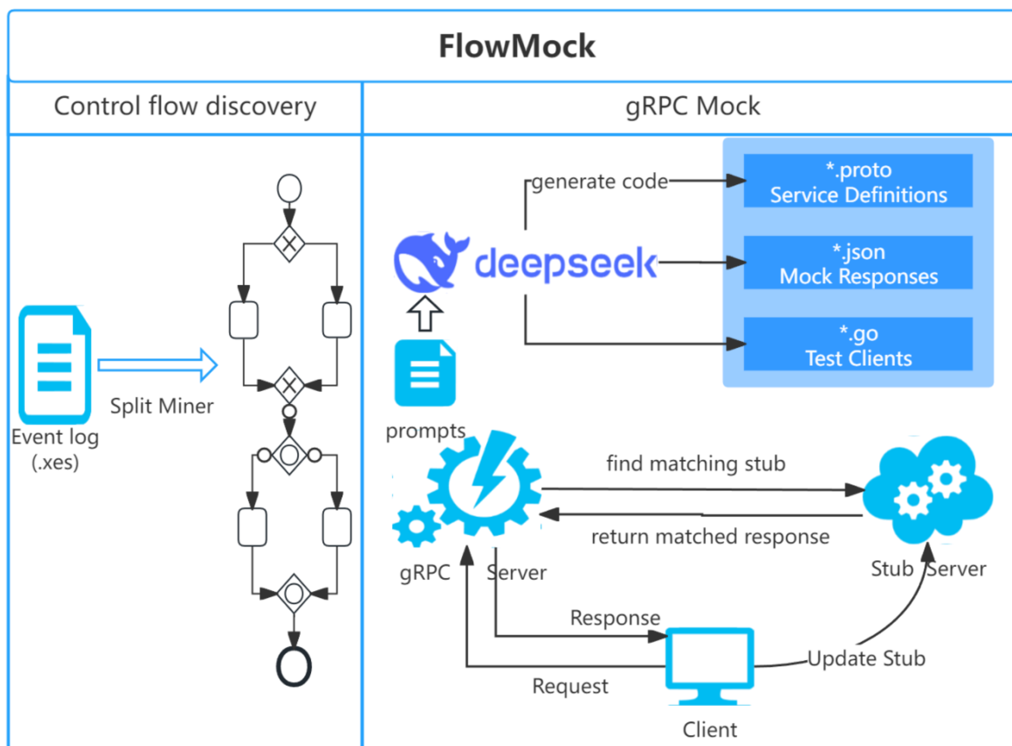

Figure 1: Framework of the FlowMock process-aware service simulation system

Table 1: Example of business process event log data

Case ID	Activity	User	Start timestamp	End timestamp
173712	W_Afhandelen leads	10912	2011-10-01 08:08:36	2011-10-01 08:10:25
173706	W_Afhandelen leads	10912	2011-10-01 08:15:43	2011-10-01 08:16:49
173718	W_Completeren aanvraag	NOT_SET	2011-10-01 08:38:48	2011-10-01 08:44:43
173718	W_Nabellen offertes	NOT_SET	2011-10-01 08:44:54	2011-10-01 08:44:58
173718	W_Nabellen offertes	NOT_SET	2011-10-01 08:45:27	2011-10-01 08:46:56
173718	W_Nabellen offertes	NOT_SET	2011-10-01 08:47:14	2011-10-01 08:50:20
...	...	...	...	...

3.1 Control-Flow Mining

Before conducting an in-depth analysis of business processes, it is first necessary to clarify the origin of business process data and its structural characteristics. Table 1 presents a portion of the BPI12-W log. XES is an open XML-based standard that is widely used for storing and exchanging event log data in the field of process mining. Files in this format record detailed attributes for each event, such as timestamps, executors, and resource usage.

FlowMock integrates a built-in process-awareness module that automatically discovers

control-flow structures from business logs based on the Split Miner algorithm, thereby enabling the service simulator to more accurately reproduce business process behavior. Specifically, this module comprises the following three core stages.

3.1.1 Construction of the Directly-Follows Graph and Short-Loop Detection

FlowMock first parses the incoming event logs (in either XES or CSV format), extracts event sequences, and constructs a Directly-Follows Graph (DFG) based on directly-follows relations. The directly-follows relation captures how frequently one event directly follows another in the log and is defined as follows:

$$|a \to b| = \left| \begin{Bmatrix} (e_i, e_j) \in E \times E \mid (e_i^l = a) \wedge (e_j^l = b) \\ \wedge \exists t \in L[\exists e_x \in t(e_x = e_i \wedge e_{x+1} = e_j)] \end{Bmatrix} \right| \tag{1}$$

Here, E denotes the set of all events in the log; e_i,$e_j$represent arbitrary event instances belonging to E; e_i^l,e_j^ldenote the event labels of the event instances; LLL denotes the event log, which consists of a collection of traces; ttt denotes a trace in L, where each trace is an ordered sequence of events; $e_x$denotes the event at position x in trace t; and e_{x+1}denotes the event that immediately follows e_x in the same trace.

Subsequently, the system detects and removes self-loops (a→a)and short loops (a↻b). These patterns may interfere with the identification of concurrency relations in later stages, and therefore need to be filtered out at the DFG construction stage.

3.1.2 Discovery of Concurrency Relations

On the basis of the constructed DFG, FlowMock further identifies concurrency relations. When there exist bidirectional directly-follows edges in the DFG (i.e., both a→band b→a), a concurrency balance threshold ε is used to determine whether a pair of nodes is in a concurrent relation, according to the following equation:

$$\frac{\left| |\mathrm{a} \to \mathrm{b}| - |b \to a| \right|}{|\mathrm{a} \to \mathrm{b}| + |b \to a|} < \varepsilon \tag{2}$$

Node pairs that satisfy the above condition and do not form short loops are regarded as concurrent. The system then removes the corresponding bidirectional edges and obtains a Pruned Directly-Follows Graph (PDFG). A smaller value of the concurrency balance threshold ε implies a stricter criterion for determining concurrency.

3.1.3 Path Filtering and BPMN Model Generation

Based on the PDFG, the system applies a path-filtering technique to generate the final BPMN process model. Specifically, it computes, for each path from the start node to the end node, the minimum edge frequency along that path and treats this value as the path capacity. By setting a percentile threshold η, the system selects and retains paths with relatively high capacity and removes edges whose frequencies fall below the threshold. The procedure attempts to balance the fitness and accuracy of the final process model.

To be more concrete, the fitness metric tests if the process model can recreate all behaviors actually observed in the event log, measuring the consistency or recall of the model with regard to the logged behavior. Higher fitness indicates that the model can more completely cover the actual behavior observed in the log.

The precision metric measures the consistency between the behavior generated by the process model and the behavior observed in the log, focusing on whether the model produces additional behavior that does not appear in the event log. Higher precision means that the model produces fewer redundant behaviors, thereby ensuring high accuracy.

Figure 2 shows the BPMN diagram generated from the BPI12-W log using the Split Miner algorithm.

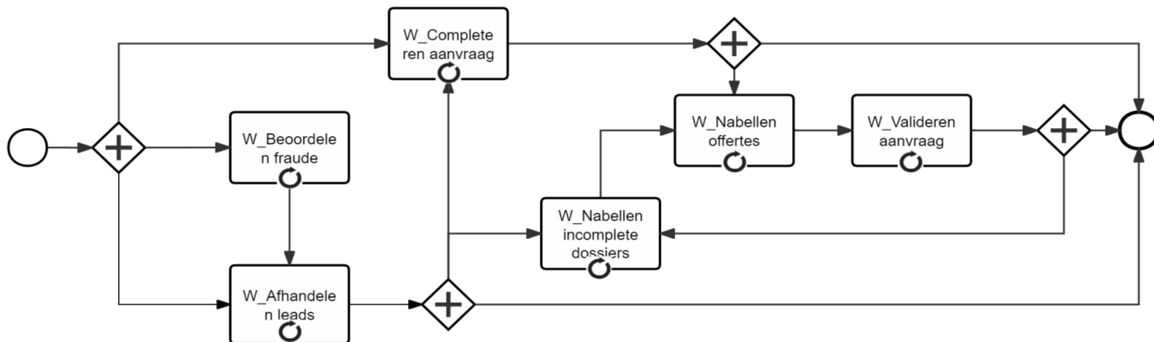

Figure 2: BPMN process model automatically discovered from the BPI12-W log using the Split Miner algorithm

3.2 Generation of gRPC-Based Mock Services

3.2.1 Deepseek-Based Processing

In FlowMock, the Deepseek model serves as the core "process-to-code" engine. The system first reads the BPMN file generated in the previous section and, via custom scripts, parses its nodes (activities, gateways, and events) and their connecting relations into a weighted directed graph. Under a predefined bound on the number of loop iterations, all possible execution paths are then enumerated.

Subsequently, Deepseek acts in the roles of both a "Protocol Buffers expert" and a "gRPC mock configuration expert," mapping natural-language business descriptions such as "create order" and "approval granted" to corresponding code skeletons. In this process, the model mainly generates three types of code: (1) a file defining the gRPC interfaces; (2) a JSON stub template for process-aware mocking; and (3) a Go client and test script skeleton.

the whole flow is called in the same Python call, Deepseek is present at each stage as some sort of "helper", the generated interfaces, the generated data, the generated codes are both in accordance with the tech specs but also make sense in terms of the logical flow of real-world businesses, BPMN -> Executable Mock Service.

3.2.2 Automatic Generation of Service Definitions

The flowmock system has a part of process that generate.proto service definition automatically, which is an important element for enabling the automation and intelligence of service simulation. FlowMock can generate the fully proto3 syntax spec conforming service definitions very quickly and automatically. Figure 3 shows the structural schematic of the generated.proto service definition file.

```
syntax = "proto3";

package BPI12;

option go_package = "./proto";

// BPI12Service defines the service
service BPI12Service {
    rpc BeoordelenFraude (BeoordelenFraudeRequest)
        returns (BeoordelenFraudeResponse);
    ...
}

// Request for BeoordelenFraude
message BeoordelenFraudeRequest {
    ...
}

// Response for BeoordelenFraude
message BeoordelenFraudeResponse {
    ...
}

...
```

Figure 3: Schematic of .proto service definitions

Header of file, protocol syntax version (syntax = "proto3";), package name, and cross-language consistency declarations used during compilation. These structure statements hold the autowriting interfaces, matches and regular in different programming languages, thus when you work with the same source code next time, it becomes convenient to use those given interface definitions.

Each BPMN business process activity is mapped to an RPC method in the gRPC service. Examining what each node in the Activity is doing, and also how the data between them are being related, FlowMock can then know what request message structure and response message structure should be created for a every method. And these message structures contain not only the necessary input variables and output results included in the process, but also through the semantic understanding ability of AI models, they automatically generate detailed and clear comments for each field, greatly improving the readability and maintainability of the.proto definition file. After generating the.proto file, FlowMock also automates the compilation process, using the protoc command to compile the.proto file into Go language stubs and client codes.

3.2.3 Generation of JSON Mock Responses (Stubs)

The.proto service specs are defined, then the FlowMock framework auto-generates JSON-based mock-response stubs which mimic the real operational workflows' logic and progression. These auto-generated configurations, as summarized conceptually in Fig. 4, specify the service keys, RPC method IDs, and the rules that dictate how incoming requests are matched to their respective outputs.

The generation of stubs comes from the application, by both the business process model and Deepseek, of reasoning on the meanings of terms used in the description of the problem. For each RPC endpoint, the framework deduces fine-grained criteria for analyzing input parameters and associates them with predetermined response patterns that the system ought to give back when those conditions are met.

```
{
  "service":"<servicename>", // name of service defined in proto
  "method":"<methodname>", // name of method that we want to mock
  "input":{
  // input matching rule. put rule here
  },
  "output":{ // output json if input were matched
    "data":{
      // put result fields here
    },
    "headers": {
      // put result headers here
    },
    "error":"<error message>"
    // Optional. if you want to return error instead.
    "code":"<response code>"
    // Optional. Grpc response code.
  }
}
```

Figure 4: JSON Stub Setup Overview

The stubs here follow the JSON format of GripMock, rules of [22]. Every configuration file has the name of the service, which mocking method to use, the input matching rules, and what the output should be. various types of matching can be performed, from exact matching to exact match order with ignore, is checks and input matches against regular expressions. And then you'll be able to answer all kinds of different questions, even if the information is being shown to you in completely different ways.

All that is needed in the output is filled in according to your processing logic. The output can simply be the result or the response headers or an error message or any different status codes covering all the different types of responses that the real service would give.

These columns will be added to GripMock Service Container as with hot reload, that means you can change columns configs and the result can be immediately observed without having to stop or start anything. And it has an embedded REST web interface that you can use to add, change, check or remove stubs from the running mock service. It makes the test setup much more flexible; developers and testers can change their test flow by changing responses, and test different business paths without waiting for it to restart.

When it is used along with a larger CPS (Cyber-Physical System) direct attack simulation[23], the test framework can be used to simulate timeout problems, packet loss problems, and problems caused by intentional data obfuscation. Error codes can be directly related to those situations and testers can easily add in complex conditions to see how the rest of the system performs or quickly revert back to normality for everyday testing.

3.2.4 Generation of Go Test Code

When FlowMock finishes generating the.proto service definitions and JSON-based stub config files, it moves on to the next phase – the automatic creation of Go test clients. Step 2 is done by the Deepseek model that synthesizes the generated definitions as well as config data for each service to get complete gRPC client implementations ready to use.

The code of the generated Go client has routines for establishing the gRPC connection, adding timeout to context, and handing errors properly. For each RPC endpoint, FlowMock embeds a test-call logic that relies on the example payloads specified in the stub configuration,

resulting in realistic request–response flows for the test cases. Also it can inspect responses, log outputs and other diagnostics to help evaluate behavior both in local runs and in CI.

After tests have been compiled and run, FlowMock centralizes all test output and execution artifacts so you can easily do followup analysis and reporting and review coverage.

3.2.5 Server and Client Components

FlowMock, a mock-service framework within the flowmock package, has more than an automatic code generation workflow, it also contains dedicated Server and Client modules which keep track of the entire simulation process. These parts together make it possible to repeat the interaction with outside services, to verify the business logic and to carry out automatic testing routines.

Server Module is the endpoint of the mock system's response. It accepts gRPC calls sent by the client, then creates reply messages from the previously created stub configuration data. On the other hand, the Client module runs the show as it sends out requests to the Server and records the responses. The client code generated by FlowMock based on Go not only calls the mock function but also embeds structured error handling and timeout control, and also provides interpretation of the returned data. The client can simulate different businesses and call the mock server multiple times to receive the return response.

These two parts are working together, FlowMock creates a closed loop simulation and test environment. The mock Server and Client together accomplish the service simulation and every step of the business process must be executable and validated as required.

4 Experimental Analysis and Results

4.1 Experimental Datasets

In this paper, six .xes event logs are used to evaluate the proposed method [1]. These logs consist of a combination of real-world logs and synthetic logs generated from models that simulate real processes. An overview of the event logs used is given in Table 2 and summarized as follows:

Academic Credential Recognition (ACR): An event log exported from the Bizagi BPM system of a university in Colombia.

BPI12-W: The W subset of the loan application log from the BPI Challenge 2012 (BPI12), containing events generated by human tasks and associated with durations.

BPI17-W: An updated W subset derived from the winning solution of the BPI Challenge 2017 (BPI17).

CVS: A simulated log from a retail pharmacy, generated according to the model presented in Fundamentals of Business Process Management.

Manufacturing Production (MP): A public log describing operational steps from an Enterprise Resource Planning (ERP) system.

Purchase-to-Pay (P2P): A publicly available synthetic log.

A trace refers to the sequence of events corresponding to a single business case from start to completion. An event is an atomic record of an activity execution, including attributes such as timestamps and executors. An activity denotes the type/label of an event (e.g., “W_Completeren aanvraag” in Table 1). In this paper, “number of traces” = number of cases in the log; “number of events” = total number of event records; and “number of activities” = number of distinct activity labels after deduplication.

Table 2: Overview of event log datasets used in the experiments

Log	Data source	Number of traces	Number of events	Number of activities
ACR	Colombian university	954	4962	16
BPI12-W	BPI Challenge	8616	59302	6
BPI17-W	BPI Challenge	30270	240854	8
CVS	Colombian university	10000	103906	15
MP	ERP system	225	4503	24
P2P	Public simulation log	608	9119	21

4.2 Performance Evaluation of Process Mining Algorithms

To assess the quality of the BPMN models generated by the Split Miner algorithm used in the control-flow discovery phase of the FlowMock system, this study selects several classical process mining algorithms—Alpha Miner, Heuristic Miner, and Inductive Miner—as baselines. Experiments and comparative performance analyses are conducted on the six BPMN datasets.

Table 3: Performance comparison of different process mining algorithms

Log	Mining method	Fitness	Precision	F1
ACR	split	0.76	1.00	0.87
	alpha	-	-	-
	heuristic	0.92	0.55	0.69
	inductive	1.00	0.21	0.34
BPI12-W	split	0.80	1.00	0.89
	alpha	0.30	1.00	0.46
	heuristic	0.93	0.78	0.85
	inductive	1.00	0.51	0.68
BPI17-W	split	0.99	0.90	0.94
	alpha	0.16	1.00	0.28
	heuristic	1.00	0.61	0.75
	inductive	1.00	0.37	0.55
MP	split	0.19	1.00	0.32
	alpha	-	-	-
	heuristic	0.93	0.16	0.28
	inductive	1.00	0.16	0.28
P2P	split	1.00	1.00	1.00
	alpha	-	-	-
	heuristic	0.27	0.59	0.37
	inductive	1.00	0.47	0.64
CVS	split	0.93	0.65	0.77
	alpha	-	-	-
	heuristic	0.99	0.49	0.66
	inductive	1.00	0.46	0.63

To evaluate the performance of the mining algorithms, this study adopts the process quality metrics fitness, precision, and F1-score. Among them, fitness measures the extent to which the discovered model conforms to the original event log; precision focuses on the

exactness of the model's predictions, i.e., its ability to avoid overgeneralization; and F1-score, as the harmonic mean of fitness and precision, provides a comprehensive assessment of the overall performance of the algorithm. The specific formulas are as follows:

$$F_1 = \frac{2 \times fitness \times precison}{fitness + precison} \tag{3}$$

Table 3 reports the experimental results obtained when using the default parameters for each method. For each log, the best score for each metric is highlighted in bold. A dash ("–") indicates that, due to syntactic or semantic problems in the discovered model (e.g., disconnected or unreliable models), the corresponding accuracy or complexity metric could not be computed in a reliable manner.

In the experiments conducted in this study, the process mining module of the FlowMock system uniformly adopts a concurrency balance threshold ε=0.3and a frequency threshold percentile η=0.7. This parameter setting improves the concurrent structure discovery and the rational process path filtering. The result shows consistently good process fitness, precision and F1-score across the 6 datasets. For example, the three metrics are all 1.00 in the P2P dataset. However, in some typical datasets, like BPI17-W and BPI12-W, the F1-scores reach 0.94 and 0.89, respectively, fully showing the method's great modeling ability and generalization ability for the complex business process.

4.3 Evaluation of AI-Based Code Generation Capability

The paper also does a comparison of the quality of AI generated code and code that has been manually written code. Among the six BPMN datasets, select the execution paths used for mock testing from those with at most two loop iterations and the maximum number of activities, and record the test success rate. On this basis, a further evaluation of code quality. Code quality metrics includes 2 dimensions, code complexity and structural complexity. Code complexity comprises things like the quantity of functions, the quantity of imports, and the comment ratio, structural complexity involves things like the quantity of fields, nesting depth, and the quantity of messages. Togeth, these metrics give an overall picture of how well and practically useful AI -based code generation is.

Table 4: Comparison of complexity and similarity between manually written code and AI-generated code

Log	RSR	.proto					.json			.go		
		Type	CR	FC	MC	TS	Depth	FC	TS	CR	FuncC	TS
ACR	1	manual	0.092	23	26	0.381	6	77	0.153	0.095	1	0.15
		ai	0.121	41	33		7	138		0.096	17	
BPI12-W	1	manual	0.137	23	15	0.0564	7	103	0.137	0.050	1	0.303
		ai	0.151	34	12		6	66		0.176	7	
BPI17-W	0.86	manual	0.194	13	12	0.0724	5	38	0.316	0.089	1	0.141
		ai	0.149	41	14		5	57		0.137	8	
CVS	0.82	manual	0.097	8	24	0.217	5	63	0.172	0.104	1	0.192
		ai	0.112	59	30		5	126		0.052	16	
MP	1	manual	0.097	8	24	0.291	5	64	0.113	0.104	1	0.239
		ai	0.122	57	32		6	145		0.097	13	
P2P	0.93	manual	0.094	10	42	0.441	5	111	0.0728	0.109	1	0.119
		ai	0.064	64	49		9	183		0.072	1	

In the experimental results reported in Table 4, a multidimensional comparison between manually written code and AI-generated code is presented. The specific meanings of each metric are as follows:

RSR (Response Success Rate): Measures the stability and effectiveness of mock service responses under different scenarios.

CR (Comment Ratio): The proportion of comment lines in the code file, reflecting the readability and maintainability of the code.

FC (Field Count): The number of fields. For .proto files, this corresponds to the total number of message fields defined in the service; for .json files, it is the number of all field paths after flattening.

MC (Message Count): The number of messages, mainly used for .proto files, indicating the total number of defined messages.

TS (Text Similarity): Measures the overall similarity between two files at the content level. It is computed based on the idea of the longest common subsequence by comparing the match degree between two complete texts in terms of character or token sequences. The similarity value ranges from 0 to 1, where values approaching 1 denote a higher degree of structural and content similarity between two texts, while lower values reflect greater divergence.

Depth refers to the nesting level of fields and is used to quantify structural complexity.

FuncC (Function Count) denotes the number of functions and is employed to evaluate the degree of modularization and invocation complexity in the Go client test code.

Type indicates the code-generation type, either manually written (manual) or AI generated (ai).

.proto/.json/.go correspond to interface definition files, mock data configuration files, and test client codes, respectively.

The experimental results indicate that FlowMock can keep a consistently high RSR for all the business-process datasets, which implies that the proposed process-aware service-simulation method may tend to maintain the response stability and service effectiveness under various conditions.

In terms of the size of the complexity and degree of structural change, we can clearly see from the.proto file and the.json file generated by the AI that the number of fields has more, so the AI model is very likely to integrate more complete business semantic information when it generates the corresponding interface and data model. This results in the generated mock services covering a wider variety of situations and being more general, but at the cost of increased nesting depth and structure, leading to higher maintenance costs.

The TS results reveal the differences between AI-written and man-written code. It is due to the fact that the AI is more likely to add extra function modules and invocation logic in the.go client for a greater number of test cases and more extensive scenarios coverage. Though such designs increase the automation of the test code, this introduces redundancy and added complexity. In comparison, manually written code is more concise in fields and functions, but usually has a higher Comment Ratio (CR), making it more readable and maintainable.

5 Conclusion and Future Work

FlowMock, a gRPC-based and process-aware service simulator which is combined with business process mining and AI-automatic code generation technology to simulate real-world business processes and generate the corresponding simulated responses. FlowMock uses Split Miner algorithm to extract the BPMN process model and it does an accurate job at reproducing concurrency, conditions and loop structures. Through the semantic analysis and

code generation capabilities of the deepseek model, the system will automatically generate a complete gRPC interface definition file, a process-aware JSON stub configuration, and a Go test client, thus completing the transformation from process models to executable mock service and providing dynamic response according to the actual business path.

The experimental result shows that on a range of real and synthetic event logs, FlowMock gets higher process fitness and precision over existing approaches. The created interfaces and codes have produced a huge success of responding successfully and a comprehensive coverage over all different kinds of cases has been achieved, which can be called as a huge success to prove the feasibility and feasibility of the above approach, to carry out the simulation of business processes and automatic testing. This study presents not only another viewpoint of the service-simulation technique, but also a reusable framework for process-awareness and automated testing in microservices architecture.

5.1 Future Work

Extending support for more communication protocols and service ecosystems.

FlowMock now focuses on gRPC, and future works would extend support for other mainstream distributed communications like REST, GraphQL and message queue (MQ) to fulfill the testing needs of diverse systems. Future work will also look into how to integrate with microservice-governance systems such as kubernetes and service mesh to add to simulation and validation of large scale distributive environments.

5.2 Make the AI-generated code easier to maintain and understand.

Improving the maintainability and interpretability of AI-generated code.
Experiments show that the automatically generated code performs well in terms of field coverage and scenario coverage, but there remains room for improvement regarding comment ratio and structural complexity. Future work will introduce software-quality-metric-driven generation control mechanisms and automatic documentation techniques, with the aim of further enhancing the readability, maintainability, and interpretability of the generated code, making it better suited to enterprise-level development scenarios.

5.3 Realizing online learning and dynamic model updating.

As business processes evolve dynamically, purely offline modeling suffers from inherent latency. We therefore plan to integrate online log collection with real-time process-mining techniques so that the system can perform continuous learning and automatically update its models. During execution, stub configurations and testing logic will be adaptively revised to maintain a high level of consistency between the mock services and the evolving business processes.

5.4 Introducing cross-domain validation and industry-specific case studies.

For my future work, I'll apply FlowMock on some domains (finance, health care, manufactrueing) to do the cross domain application. To better accommodate domain specific process patterns and regulatory demands, the study will go on to better the generalizability, scalability, and cross-industry adaptability of the system.

References

[1] Camargo M, Báron D, Dumas M, et al. Learning business process simulation models: A hybrid process mining and deep learning approach[J]. Information Systems, 2023, 117: 102248.

[2] Wang S, Tang Y X, Guo W H. Service Scheduling and Resource Allocation Scheme of Smart Grid Considering Priority[J]. Southern Power System Technology, 2024, 18(04): 59-70.

[3] Van Der Aalst W M P, Carmona J. Process mining handbook[M]. Springer Nature, 2022.

[4] Augusto A, Conforti R, Dumas M, et al. Split miner: automated discovery of accurate and simple business process models from event logs[J]. Knowledge and Information Systems, 2019, 59(2): 251-284.

[5] Dakic D, Stefanovic D, Cosic I, et al. BUSINESS PROCESS MINING APPLICATION: A LITERATURE REVIEW[J]. Annals of DAAAM & Proceedings, 2018, 29.

[6] Van Der Aalst W. Process mining: Overview and opportunities[J]. ACM Transactions on Management Information Systems (TMIS), 2012, 3(2): 1-17.

[7] Park S, Kang Y S. A study of process mining-based business process innovation[J]. Procedia Computer Science, 2016, 91: 734-743.

[8] Van der Aalst W M P. Process mining: a 360 degree overview[M]//Process mining handbook. Cham: Springer International Publishing, 2022: 3-34.

[9] R'bigui H, Cho C. The state-of-the-art of business process mining challenges[J]. International Journal of Business Process Integration and Management, 2017, 8(4): 285-303.

[10] Martin N, Depaire B, Caris A. The use of process mining in business process simulation model construction: structuring the field[J]. Business & Information Systems Engineering, 2016, 58(1): 73-87.

[11] Camargo M, Dumas M, González-Rojas O. Automated discovery of business process simulation models from event logs[J]. Decision Support Systems, 2020, 134: 113284.

[12] Soliman G, Mostafa K, Younis O. Reinforcement learning for process mining: Business process optimization[C]//World conference on information systems and technologies. Cham: Springer Nature Switzerland, 2024: 108-125.

[13] Kokala A. Leveraging process mining to enhance business process management: Concepts, framework, and applications[J]. International Research Journal of Modernization in Engineering Technology and Science, 2024, 6(12).

[14] Vom Brocke J, Jans M, Mendling J, et al. A five-level framework for research on process mining[J]. Business & Information Systems Engineering, 2021, 63(5): 483-490.

[15] Mamudu A, Bandara W, Wynn M T, et al. Process mining success factors and their interrelationships[J]. Business & Information Systems Engineering, 2024: 1-20.

[16] Reinkemeyer L. Status and future of process mining: from process discovery to process execution[M]//Process Mining Handbook. Cham: Springer International Publishing, 2022: 405-415.

[17] Freeman S, Mackinnon T, Pryce N, et al. Mock roles, not objects[C]//Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications. 2004: 236-246.

[18] Freeman S, Mackinnon T, Pryce N, et al. jMock: supporting responsibility-based design with mock objects[C]//Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications. 2004: 4-5.

[19] Spadini D, Aniche M, Bruntink M, et al. To mock or not to mock? an empirical study on mocking practices[C]//2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). IEEE, 2017: 402-412.

[20] Spadini D, Aniche M, Bruntink M, et al. Mock objects for testing java systems: Why and how developers use them, and how they evolve[J]. Empirical Software Engineering, 2019, 24(3): 1461-1498.

[21] Pereira G, Hora A. Assessing mock classes: An empirical study[C]//2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2020: 453-46

[22] Tokopedia. gripmock: gRPC Mock Server. GitHub repository. Available: https://github.com/tokopedia/gripmock.

[23] Lin F, Mei Y, Zhu Y. Overview of the entire process influence of cyber attack on typical scenarios of power systems[J]. Southern Power System Technology, 2023, 17(11): 61-75.