https://doi.org/10.31891/csit-2025-4-8
UDC 004.93:004.415.2

MALIARENKO Vladyslav
National Technical University “Kharkiv Polytechnic Institute”

AI-BASED AUTOMATION OF DECISION LOGIC REPRESENTATION: BRIDGING GAP IN AUTOMATED DECISION MODELING VALIDATION

The study aims to resolve the "modeling bottleneck" in Business Process Management by developing an automated method for ensuring the correctness of Decision Model and Notation (DMN) tables generated by Large Language Models (LLMs). The primary goal is to determine whether shifting the AI's role from a pure generator to a validator within a closed-loop system can overcome the structural limitations inherent in stochastic models.

The research employs a comparative experimental design using a "Mutation Testing" approach. We analyze two distinct workflows: (1) Static Generation, where test cases are fixed, and (2) Dynamic Paired Generation, where the LLM regenerates both the decision logic (DMN XML) and the validation criteria (Test Cases JSON) simultaneously upon failure. The methodology integrates concepts from "Struc-Bench" for structural analysis and utilizes a deterministic DMN engine (Camunda) for execution-based verification.

The experiments demonstrate that "out-of-the-box" LLM generation fails in approximately 4.5% of cases due to semantic drift and structural hallucinations when validated against static benchmarks. However, the proposed "Dynamic Paired Generation" workflow achieved a 100% convergence rate across 200 cycles. The system successfully identified and corrected both syntactic errors (XML schema violations) and logical errors (Hit Policy violations) without human intervention.

The study introduces the concept of "Dynamic Paired Generation" for neuro-symbolic systems. Unlike traditional "Chain-of-Thought" prompting, this approach leverages the mutual consistency between two independent structural representations (Logic and Examples) to filter out hallucinations, proving that dynamic validation is superior to static prompting for structured data tasks.

The proposed framework provides a blueprint for "Self-Healing" decision management systems. It allows non-technical business analysts to convert natural language policies into executable, error-free DMN models, significantly reducing the time and cost of regulatory compliance and process automation.

Keywords: Large Language Models, DMN, Automated Decisioning, Neuro-Symbolic AI, Validation, Struc-Bench, Business Process Management.

МАЛЯРЕНКО Владислав
Національний технічний університет «Харківський політехнічний інститут»

ШІ-БАЗОВАНА АВТОМАТИЗАЦІЯ ПРЕДСТАВЛЕННЯ ЛОГІКИ РІШЕНЬ: ПОДОЛАННЯ РОЗРИВУ У ВАЛІДАЦІЇ АВТОМАТИЗОВАНОГО МОДЕЛЮВАННЯ РІШЕНЬ

Дослідження спрямоване на вирішення "вузького місця моделювання" в управлінні бізнес-процесами шляхом розробки автоматизованого методу забезпечення коректності таблиць Decision Model and Notation (DMN), згенерованих великими мовними моделями (LLM). Основна мета – визначити, чи може зміна ролі ШІ з чистого генератора на валідатор у системі із замкненим циклом подолати структурні обмеження, властиві стохастичним моделям.

У дослідженні використовується порівняльний експериментальний дизайн із застосуванням підходу "Мутаційне тестування". Ми аналізуємо два окремі робочі процеси: (1) Статична генерація, де тестові випадки є фіксованими, та (2) Динамічна парна генерація, де LLM одночасно регенерує як логіку рішення (DMN XML), так і критерії валідації (Test Cases JSON) у разі невдачі. Методологія інтегрує концепції "Struc-Bench" для структурного аналізу та використовує детермінований DMN-рушій (Camunda) для верифікації на основі виконання.

Експерименти демонструють, що "з коробки" генерація LLM є помилковою приблизно у 4,5% випадків через семантичний зсув та структурні галюцинації при валідації відносно статичних еталонів. Однак запропонований робочий процес "Динамічна парна генерація" досяг 100% рівня конвергенції протягом 200 циклів. Система успішно ідентифікувала та виправила як синтаксичні помилки (порушення схеми XML), так і логічні помилки (порушення політики Hit Policy) без втручання людини.

Дослідження впроваджує концепцію "Динамічної парної генерації" для нейро-символьних систем. На відміну від традиційного промптингу "Ланцюг думок", цей підхід використовує взаємну узгодженість між двома незалежними структурними представленнями (Логіка та Приклади) для фільтрації галюцинацій, доводячи, що динамічна валідація перевершує статичний промптинг для завдань зі структурованими даними.

Запропонована структура надає план для систем управління рішеннями, що "Самостійно Відновлюються". Вона дозволяє нетехнічним бізнес-аналітикам перетворювати політики природної мови на виконувані, безпомилкові моделі DMN, значно скорочуючи час та витрати на відповідність нормативним вимогам та автоматизацію процесів.

Ключові слова: Великі мовні моделі, DMN, Автоматизоване прийняття рішень, Нейро-символьний ШІ, Валідація, Struc-Bench, Управління бізнес-процесами.

Received / Стаття надійшла до редакції 11.10.2025
Accepted / Прийнята до друку 30.11.2025

Introduction

The contemporary landscape of Business Process Management and Information Systems requires the rapid deployment and modification of decision logic as a key competitive differentiator. Organizations, particularly in financial, insurance, and public sectors, face volatile regulatory environments. To manage this complexity, the

Object Management Group introduced the Decision Model and Notation standard. DMN uses a standardized XML-based syntax to allow business analysts to define rules in a format that is both human-readable and machine-executable. Despite DMN's clear representational standard, the process of knowledge acquisition, translating the unstructured prose of policy documents, legislative texts, and expert interviews into the structured rows and columns of a DMN Decision Table, remains a labor-intensive, manual endeavor. This "modeling bottleneck" is prone to translation errors, where the nuance of a natural language requirement is lost or misinterpreted during its conversion into the FEEL, the expression language used by DMN.

The advent of Large Language Models and Transformer architectures have significantly influenced the approach to automated software engineering. Models such as Claude Sonnet 4.5, GPT-4.1, Gemini 2.0 Pro, Codestral, Llama 3.1 and others have shown strong proficiency in code generation, summarization, and natural language understanding. This progress has prompted the research community to explore their use for DMN. However, early experimental inquiries have tempered this enthusiasm with empirical reality. As documented by [1], initial attempts to utilize GPT-3 for generating DMN decision logic from user queries resulted in low success rates. The stochastic nature of Next-Token Prediction often conflicts with the strict deterministic requirements of the DMN XML schema. The model might "hallucinate" a closing tag, invent a non-existent FEEL function, or misalign columns in a decision table, rendering the output syntactically invalid. These findings align with broader research by [2]. on "Struc-Bench," which questions the inherent capability of LLMs to generate complex structured data without specific instruction tuning or external constraints.

While the generation of DMN models has proven difficult, a less explored but potentially more high-value application is the validation of existing models. The hypothesis driving the research is that LLMs, while poor architects of structure, may be excellent critics. Their vast pre-training on code and logic puzzles may allow them to identify subtle inconsistencies (overlapping rules, missing inputs, or semantic ambiguities) that traditional rule-based validators miss.

Purpose

The study aims to resolve the "modeling bottleneck" in Business Process Management by developing an automated workflow for ensuring the correctness of Decision Model and Notation tables generated by Large Language Models. The primary goal is to assess whether a closed-loop system, which shifts the AI's role from a sole generator to a validator, can overcome the structural and semantic limitations of stochastic text generation. Specifically, the research aims to fill the gap in automatically validating generated structured responses. By evaluating test strategies, the study intends to demonstrate that dynamic realignment of decision logic and validation criteria enables the system to self-correct and converge on executable DMN models.

Related Works

To ground the experimental design, this chapter examines the intersection of distinct domains. The DMN Standard, the specific challenges of structured data generation in Natural Language Processing, and the emerging field of Neuro-Symbolic AI. This chapter synthesizes recent literature to establish the necessity of a validation-driven approach. DMN is not merely a file format; it is a rigorous specification for decision logic. At its core is the Decision Table, which maps a set of inputs (Conditions) to a set of outputs (Conclusions) based on rows of rules. DMN defines "Hit Policies" (e.g., Unique, Any, First, Priority) that dictate how the engine should behave if multiple rules match the input. A "Unique" policy implies that for any set of inputs, only one rule can be true. A violation of this (i.e., rule overlap) is a logical error.

The expression language, FEEL, supports complex logic including ranges ([1..10]), list manipulation, and boolean logic. The complexity of DMN lies in the interaction between these elements. A decision modeler must ensure that:

1. Every possible combination of inputs has a defined output (Completeness).
2. No two rules produce conflicting outputs for the same input under specific hit policies (Consistency).
3. The logic matches the business intent derived from the source text (Correctness).

Traditional validators (like those in Camunda or Drools) can mathematically check for Completeness and Consistency. However, they cannot check for Correctness relative to the business domain. This is where the semantic capabilities of LLMs become relevant. The "modeling bottleneck" described by Vanthienen et al. arises because the translation from "business intent" to "FEEL syntax" requires a dual expertise for domain knowledge and technical specification.

The literature regarding LLMs and logic modeling reveals a trajectory from optimism to skepticism, and finally to nuanced hybridization. The study [1] serves as a primary baseline for our research. Their experiment involved prompting GPT-3 to transform natural language scenarios into DMN tables. The results were categorized as having "very low success." The authors noted that while the model captured the "gist" of the logic, it failed to produce executable XML. The disconnect between the probabilistic generation of text and the rigid syntax of XML resulted in "hallucinated syntax" – code that looks correct to a human but fails compilation. In [2] provided a theoretical explanation for these failures in their work on "Struc-Bench". They argue that LLMs are optimized for

natural language coherence, not structural rigidity. Complex data structures like SQL tables or DMN XML require a level of long-range dependency tracking and strict schema adherence that standard pre-training objectives do not sufficiently penalize. When an LLM generates a closing tag </inputEntry>, it is predicting a token based on probability, not parsing a syntax tree. Tokenization of Structure demonstrates that LLMs treat structural markers (tags, braces) as semantic tokens. This leads to common errors such as unclosed tags or mismatched attribute values [2]. The Struc-Bench study introduced specific metrics (P-Score and H-Score) to quantify this deficit, showing that LLMs perform significantly worse on structured tasks (HTML, LaTeX, Tables) than on free-text generation [2].

Therefore, "out-of-the-box" generation of DMN is fundamentally limited by the architecture of current LLMs. The model might understand the rule ("If Credit Score > 700, Approve"), but it struggles to serialize this rule into the verbose and strictly nested structure of DMN 1.3 XML. Recognizing the limits of pure generation, researchers have pivoted toward extraction and augmentation.

The paper [3] explored using BERT and Bi-LSTM-CRF for extracting decision models from text. Their findings were pivotal: BERT outperformed the recurrent Bi-LSTM architectures significantly. Crucially, they discovered that stop-word removal was critical for achieving high F1-scores. This implies that the "noise" of natural language interferes with the signal of decision logic. By stripping away non-essential linguistic tokens, the model could better focus on the decision variables and dependencies. This insight informs our experimental preprocessing strategy: rather than feeding raw policy text directly into a generator, preprocessing steps that isolate logic are beneficial. The work of [4] provides the architectural context for our study. They propose a symbiotic relationship: DMN models generate clean, structured training data for Machine Learning, while ML models provide analytics to refine the DMN logic. [5] extend this with Neuro-Symbolic AI for Compliance Checking. Their system uses Deep Learning for perception and Answer Set Programming, a logic programming paradigm similar to DMN, for verifying compliance constraints. This layered architecture, where neural networks handle the fuzziness of the real world and symbolic engines handle the rigidity of rules, is the most robust path forward for automated decisioning. If the model cannot generate perfect structure, can it be guided to do so? Chain-of-Thought prompting, introduced by [6], encourages the model to generate intermediate reasoning steps. In the context of DMN, this means asking the model to list the inputs and outputs before writing the XML. In [7] recently proposed "DMN-Guided Prompting" , a framework that integrates DMN concepts directly into the prompt structure. By forcing the LLM to structure its "thought process" using DMN elements (Input Data, Decisions, Knowledge Sources), they achieved better control over LLM behavior. This suggests that DMN is not just an output format but a useful cognitive scaffold for the LLM itself.

A critical point for the automatic structural data generation is whether LLMs can correct their own mistakes. The [8] study evaluates self-correction strategies across reasoning and code generation tasks. Self-correction can improve accuracy, particularly when the model is provided with external feedback (e.g., an error message from a validator). However, "intrinsic" self-correction (asking the model "Are you sure?") is less effective than "extrinsic" correction (showing the model the compiler error). This supports the design of our Paired Generation experiment. We do not merely ask the model to "check its work"; we execute the generated DMN against generated test cases and feed the specific failures back into the generation loop. This aligns with the findings of [9], which uses formal verification to bootstrap code generation.

Finally, we must contextualize DMN generation within the broader scope of Large Process Models defined by [10]. They envision LLMs as orchestrators of business processes, capable of generating, analyzing, and optimizing process flows. However, they explicitly warn of the "audibility gap" – LLMs are black boxes, while business processes require transparency. [11] proposed the BPMN Assistant, an LLM-based tool for process modeling, noting that while generation is feasible, the verification of the generated model against business constraints is the primary challenge. Our research addresses this specific gap for DMN, proposing that validation is the key to unlocking generation.

Summary of Literature Review

The existing literature converges on several consistent findings regarding the generation of DMN artifacts using Large Language Models. First, direct text-to-DMN generation is unreliable: models frequently violate required structural constraints, leading to syntactically invalid or non-executable artifacts, as demonstrated in [1, 2]. Second, hybrid neuro-symbolic approaches are essential. Methods that combine neural interpretation with formal symbolic verification consistently achieve superior results, as reported in [4, 5]. Third, self-correction mechanisms are effective primarily when external, execution-based feedback is available; intrinsic self-reflection alone has limited impact on convergence, as shown in [8].

Despite these insights, the literature lacks a direct experimental comparison between static and dynamic validation strategies for DMN. Prior studies have focused predominantly on static logic extraction (e.g., BERT-based approaches) or one-shot generation pipelines, leaving the role of iterative, feedback-driven validation underexplored.

This study addresses this gap by evaluating a closed-loop system in which the Large Language Model serves both as the generator of decision logic and as the generator of validation criteria in the form of test cases. This

setup enables a systematic assessment of whether dynamic, real-time realignment between generated logic and validation artifacts facilitates reliable convergence toward correct and executable DMN models.

Proposed Technique

The core concept lies under the investigation we made, that a non-expert user can obtain a correct DMN decision table from natural-language business rules if the language model is embedded into a controlled generation-and-validation loop. The user provides a textual rule set and a schema descriptor that defines the structure of the decision input object and the allowed output values. The system then delegates all modelling work to an LLM, while a deterministic validator ensures that only DMN tables that behave correctly with respect to the rules are accepted. In this setting, the language model is not treated as a reliable source of ground truth, but as a generator of candidate artefacts whose correctness must always be verified by execution. The initial evaluation (Experiment 1) confirmed that structural correctness cannot be guaranteed solely through prompting. Typical failure modes included XML tables missing required output entries and tables assigning incorrect values to the product field. Such errors persisted across several regeneration attempts, indicating that static validation of structural patterns is insufficient for reliable DMN generation. Conceptually, the workflow is built around two independent calls to the LLM. The first call generates a collection of test cases. Each test case is a JSON object that conforms to the input schema and contains two parts: an object with attribute values (for example, LTV, DTI, credit score, fixed-rate preference, intended years in the property) and an expected field specifying the resulting mortgage product. The mortgage rule set defines three possible outcomes ("Fixed30", "ARM7/1", and "ManualReview") under a FIRST hit policy, and the prompt instructs the model to cover typical as well as edge scenarios. The second call generates a DMN 1.3 decision table in Camunda-compatible XML. This table must implement the same rules using FEEL expressions and the attribute paths defined in the schema descriptor. The prompts for both calls are strictly formatted and forbid any extraneous text, enabling direct machine processing. As depicted in Figure 1, the generation process involves a sequence of steps.

The architecture that realises this concept is modular. A request layer accepts the natural-language rules and triggers the generation loop. A schema injector transforms the JSON schema and DMN context descriptor into a form that can be embedded into prompts, ensuring that both test cases and DMN expressions use the same attribute structure. A prompt composer merges the rule set, schema information, and technical instructions into the two prompts described above. A generation module orchestrates the interaction with the LLM API and returns the generated test cases and DMN XML to the validation module. The validation module first performs syntactic checks (XML well-formedness and DMN schema compliance) and then executes the DMN table against all generated test cases, comparing the actual product value with the expected result contained in each test case. A dispatcher coordinates these components and decides whether the current artefacts are accepted or whether a new generation iteration is required.

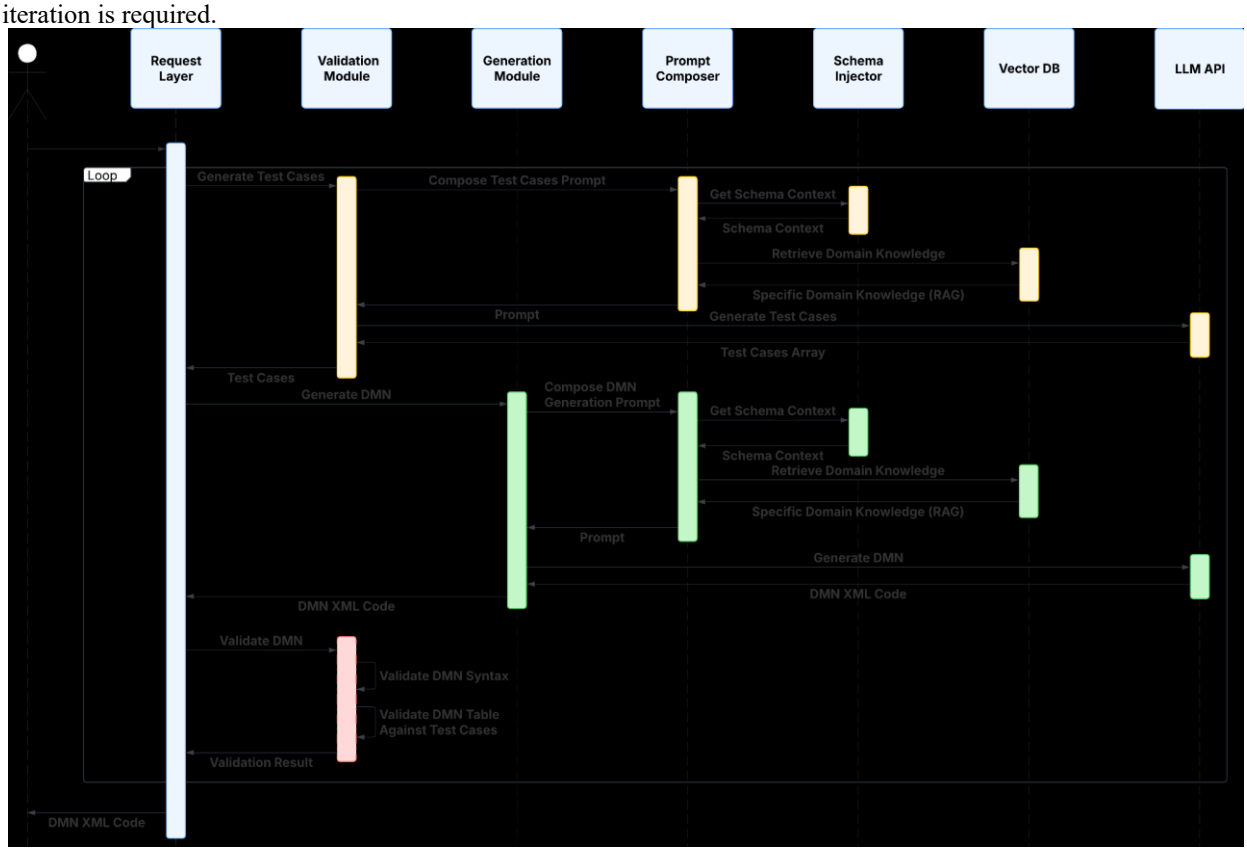

Fig. 1. Generation process sequence diagram

Within this framework, demonstrated at Figure 2, two configurations of the validation loop are evaluated. In the first configuration, test cases are generated once at the beginning of the experiment and reused for all subsequent attempts to generate a valid DMN table. When validation fails, only the DMN XML is regenerated, while the test collection remains fixed. This setting exposes the structural and logical robustness of the DMN generator when confronted with a static semantic reference. In the second configuration, both the test cases and the DMN table are generated together as a pair in every iteration. If validation fails, the entire pair is discarded and regenerated. Here the underlying hypothesis is that interpreting the same rule set twice in close succession – once as examples and once as a decision table – encourages the model to produce internally consistent artefacts, so that the validation loop can eventually converge to a fully correct DMN model.

The materials and methods presented in the following subsections are organised around this basic concept. The subsequent section on test results then analyses how these design choices affect success rates, failure modes, and execution time across 200 test cycles per configuration. The test collection in this study served as the primary mechanism for verifying whether the automatically generated DMN decision tables faithfully implemented the natural-language business rules. Since the workflow relies on semantic equivalence between two independently generated artefacts – test cases and a DMN table – the design of this collection directly determines the validity and robustness of the evaluation. All experiments in this section used the same rule set describing mortgage-product assignment, utilising a FIRST hit policy and specifying three possible outcomes: Fixed30, ARM7/1, and ManualReview. The rules were intentionally chosen for their mixture of discrete thresholds, conditional combinations, and overlapping applicability, providing a representative scenario for testing LLM-based decision modelling.

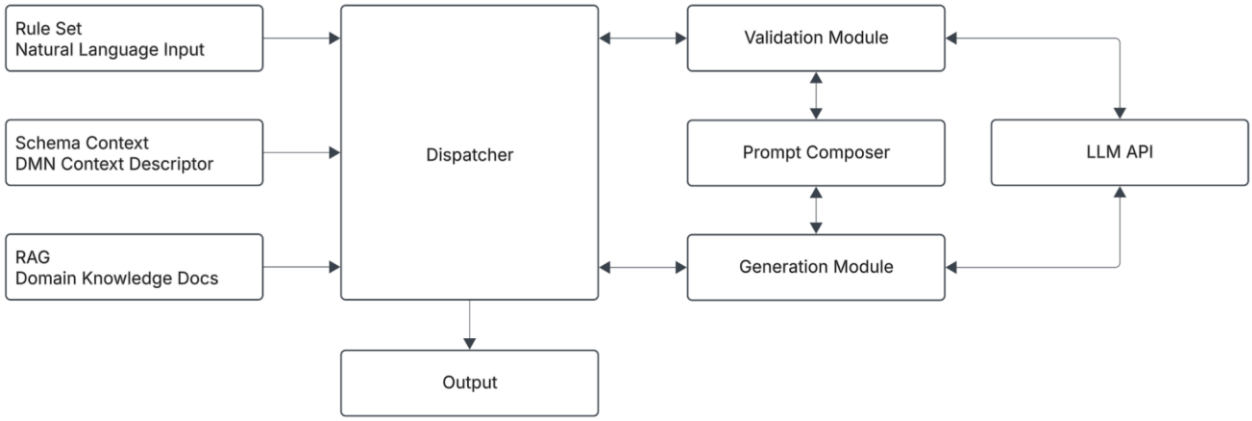

Fig. 2. Framework Components

Each test case comprised a structured JSON object representing a mortgage application and an expected output derived from the mortgage rules. The schema defined attributes such as credit score, DTI ratio, LTV ratio, income-stability indicators, fixed-rate preference flags, sensitivity to rate changes, and expected duration of property ownership (Figure 3). Embedding this schema into both the test-case and DMN-generation prompts ensured that all generated inputs were syntactically valid and executable without modification. An example of such a test-case structure is shown for illustration purposes.

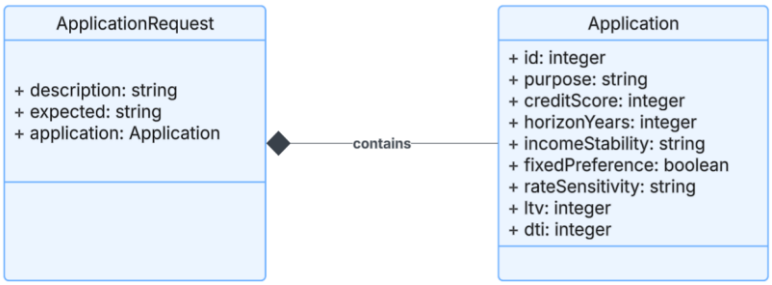

Fig. 3. Test-case class diagram

The test collection differed fundamentally between the two experiments. In Experiment 1, the collection was static: the LLM generated a single batch of test cases before the start of the 200-cycle evaluation. This batch served as a fixed semantic reference against which every generated DMN table was validated. If a DMN table failed validation, only the table was regenerated; the test cases remained unchanged. This configuration enabled an assessment of the model's structural reliability under a fixed interpretation of the rules, revealing how frequently the system produced a DMN table that correctly matched a pre-established ground truth. The distribution of outcomes showed that although most cycles converged immediately, persistent failures remained when the generated DMN logic diverged from the expected behaviour encoded in the static test cases.

In Experiment 2, the test collection adopted a dynamic approach. For each of the 200 cycles, the LLM independently generated a new batch of test cases and a new DMN table using the same rule set and schema. These artefacts were not synchronised through any shared memory or direct cross-referencing; instead, their alignment depended entirely on the LLM interpreting the rule set consistently in both prompts. Validation checked whether the decision produced by the DMN table for each test-case object matched the expected output encoded inside that object. If any mismatch occurred, both artefacts were discarded and regenerated together. This paired-generation mechanism significantly improved alignment because inconsistencies tended to be self-correcting: when both interpretations were regenerated as a pair, they converged rapidly, producing semantically consistent outputs in every test cycle.

A comparison of the aggregated cycle outcomes illustrates this distinction clearly (table 1). Experiment 1 produced 184 first-attempt successes, 7 second-attempt successes, and 9 total failures. Experiment 2 achieved 181 first-attempt successes, 17 second-attempt successes, 2 third-attempt successes, and no failures. This progression demonstrates that while a fixed semantic reference can expose structural fragility, dynamic pairwise generation mitigates these deviations and ensures complete convergence.

Table 1

Experiment comparison

Metric	Experiment 1 Static Test Collection	Experiment 2 Dynamic Paired Generation	Key Insight
Number of Cycles	200	200	Consistent volume for comparison.
First-Attempt Successes	184	181	Initial performance is similar.
Second-Attempt Successes	7	17	Dynamic strategy shows better convergence on retry.
Third-Attempt Successes	0	2	Paired regeneration enables deeper convergence.
Total Failures (Unresolved)	9	0	Dynamic strategy achieves 100% convergence.
Overall Success Rate	95.5% (191/200)	100% (200/200)	Paired regeneration guarantees correctness.
Expected Attempts per Success	~1.036	~1.105	Static setting is faster when successful, but fails entirely more often.
Failure Rate	4.5% (9/200)	0% (0/200)	Dynamic paired generation eliminates all dead-end failures

Beyond correctness, the second experiment also provided insight into runtime characteristics of the dynamic test-collection strategy. The full 200-cycle execution required 3 hours, 44 minutes, and 10 seconds. Individual cycles ranged from 42 seconds (minimum) to 498 seconds (maximum), with an average of 67 seconds. The longer cycles corresponded primarily to iterations requiring multiple regenerations. Despite this variance, the overall runtime demonstrates that the paired-generation approach is computationally feasible for both experimental and applied settings, particularly given its guarantee of convergence. Overall, the test collection demonstrates that correctness in LLM-generated DMN tables is not achieved through structural constraints alone, but through execution-based comparison against well-formed input–output examples. The static test collection highlights the difficulty of matching a fixed semantic reference, while the dynamic approach shows that regenerating test cases and DMN logic together enables stable convergence. This establishes the test collection as a central component of the workflow: not merely evaluative, but fundamental to enabling deterministic correctness in a probabilistic generation environment.

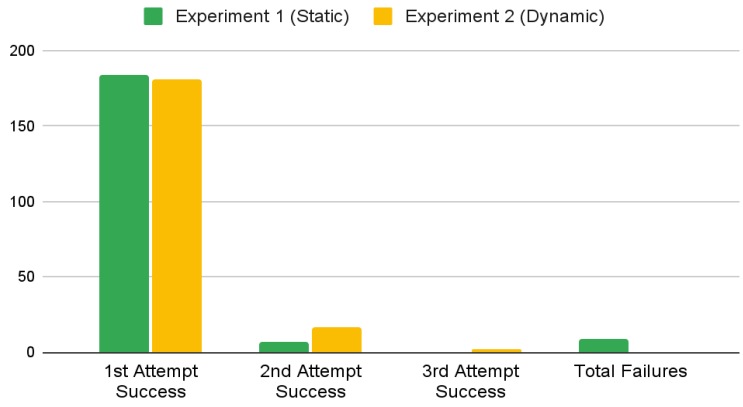

Fig. 4. Experiment Comparison

The generation workflow relies on two independent prompts: one dedicated to creating test cases and another to constructing the DMN decision table. Although both prompts operate over the same rule set and schema, they are intentionally separated to avoid direct cross-alignment. This ensures that any semantic agreement detected

during validation results from consistent reasoning by the model rather than from prompt-induced dependencies. The test-case prompt instructs the model to generate a JSON array of domain objects that follow the injected schema and encode the expected outcomes derived from the natural-language mortgage rules. The output is restricted to JSON only, with no commentary or explanatory text, ensuring deterministic parsing and compatibility with the validation engine. A second system-level instruction ensures that each generated case contains not only the input attributes but also a precise expected result, enabling runtime comparison during validation.

Test-Case System Prompt

You are generator of JSON array containing objects that fulfill the structure from <objectSchema> and the values from <rules>. Return JSON in the following format:

```
[{
   "name": "<testCaseName>",
   "description": "<testCaseDescription>",
   "expected": [ { ... } ],
   "object": { ... }
}]
```

Return only JSON code with no additional text.
RETURN NOTHING EXCEPT JSON CODE. TRY TO RETURN JSON IN A SINGLE LINE.
Listing 1 presents the exact test-case prompt used in the experiments.

```
Generate N test cases for DMN decision table validation.
Each test case should be a JSON object with fields matching the input schema.
Provide as many test cases as possible, covering edge cases and typical scenarios.
Ensure the test cases are valid according to the provided JSON schema.
Respond only with a JSON array of test case objects, without any additional text.
IT IS VERY IMPORTANT!
<objectSchema>$schema</objectSchema>
<rules>$input</rules>
```

The DMN-generation prompt performs a different task: transforming the same rule set into a complete DMN 1.3 decision table encoded as valid XML compatible with the Camunda engine. The prompt enforces strict syntactic rules, including the use of FEEL-compatible operators, correct attribute paths derived from the schema, and proper table orientation. The model is forbidden from returning anything other than XML, ensuring that the result can be fed directly into the validator.

DMN System Prompt

You are CAMUNDA DMN 1.3 decision table XML generator.
Return only DMN XML code compatible with Camunda engine.
Do not use <= or >= or > or < in <expression>. Use <= , >= , > , < instead.
If a range has two numeric boundaries, use (value1..value2) or [value1..value2].
Leave <text></text> empty if the entry has no condition.
RETURN ONLY XML AND NOTHING ELSE.
Listing 2 presents the exact prompt used for generating DMN tables.

```
RETURN ONLY AND ONLY CAMUNDA XML CODE IT'S VERY IMPORTANT!
Generate Camunda DMN table that can be applied to following schema and rules.
Make sure the table can be read horizontally.
USE <schema> to find the correct <variable> path for <expression> from the <rules>.
If <variable> is not present in the <schema> then try to calculate it from existing schema attributes. The
column headers contain the inputs and the output. Specify hitPolicy from the rules or set it to "RULE ORDER" if
none is specified. The <decision> id must be "decision".
If <outputEntry> <text> is a string type, wrap values in double quotes.
<schema>$schema</schema>
<rules>$input</rules>
```

Together, these prompts form the operational core of the generation workflow. Their separation guarantees that the model independently interprets the same rule set along two complementary dimensions: example-driven reasoning for test-case construction and rule formalisation for DMN generation. Validation occurs only after both artefacts are produced, allowing Experiments 1 and 2 to evaluate structural reliability and semantic convergence respectively.

Results

The experimental evaluation consisted of two independent test campaigns, each comprising 200 execution cycles. Both experiments used the same mortgage-rule set, the same schema descriptor, and the same validation

engine, but differed in the strategy by which test cases and DMN tables were generated. The first experiment employed a static test collection that remained unchanged across all regeneration attempts, while the second experiment used a dynamicpaired-generation strategy in which both artefacts were regenerated together whenever validation failed. The results presented in this section analyse the performance of each configuration in terms of correctness, convergence behaviour, stability across regeneration attempts, and practical computational cost.

In the first configuration, the test-case set was generated once at the beginning of the experiment and then reused for all attempts to generate a valid DMN table. Validation failures triggered regeneration of the DMN table only, while the test cases remained constant. This setup emphasises structural and logical robustness: because test cases do not change, the system cannot rely on reinterpreting the rule set to resolve inconsistencies. Out of 200 test cycles, 184 DMN tables were valid on the first attempt, seven became valid on the second attempt, and nine remained invalid after all five attempts. The overall success rate of the experiment can be formalised as:

$$SuccessRate = \frac{N_{Valid}}{N_{Total}} \tag{1}$$

where N_{Valid} - number of validated (successful) results, N_{Total} - total number of attempts or requests.

For Experiment 1 this yields:

$$SuccessRate_{Exp1} = \frac{191}{200} = 0.955 \tag{2}$$

which corresponds to 95.5% correctness across all cycles.

To characterise stability, the expected number of attempts per successful generation is given by:

$$E[T] = \frac{1 * 184 + 2 * 7}{191} = 1.036 \tag{3}$$

where $E[T]$ - expected number of attempts per successful generation.

This value confirms that most valid outputs required exactly one attempt and that regeneration almost never corrected deeper misinterpretations. The nine unresolved failures illustrate this limitation: errors such as malformed XML fragments or persistent semantic mismatches (e.g., expected "ManualReview" but produced "Fixed30") were reproduced across attempts, showing that static validation exposes systematic LLM inconsistencies that cannot be overcome by retries alone.

In the second experiment, both the test cases and the DMN table were regenerated in every iteration when validation failed. This strategy enables the model to reinterpret the rule set in two parallel forms – examples and executable logic – within the same iteration, thereby strengthening semantic alignment. The results demonstrate a significant improvement. Out of 200 cycles, 181 were valid on the first attempt, 17 became valid on the second, and two were valid on the third, with zero failures after all regeneration attempts. The success rate for this configuration is therefore:

$$SuccessRate_{Exp2} = \frac{200}{200} = 1 \tag{4}$$

In addition, the expected number of attempts per cycle is:

$$E[T] = \frac{1 * 181 + 2 * 17 + 3 * 2}{200} = 1.105 \tag{5}$$

While slightly higher than in Experiment 1, this value reflects the paired-regeneration mechanism: some semantic inconsistencies required reevaluation, but unlike the static configuration, the system always converged. Execution time further illustrates these dynamics. The total runtime of 3 hours, 44 minutes, 10 seconds for all 200 cycles corresponds to an average duration of:

$$\underline{t} = \frac{t_{total}}{N} = \frac{13450}{200} = 67.25\ sec \tag{6}$$

where t_{total} - total execution time of all test cycles combined, N - the total number of test cycles, $\underline{t}$ - execution time for a single cycle.

The fastest cycle lasted 42 seconds, while the slowest required 498 seconds, despite succeeding on the first attempt. This variance is primarily attributable to LLM latency rather than repeated unsuccessful attempts. The absence of dead-end failures strongly supports the hypothesis underlying the dynamic configuration: paired regeneration expands the interpretation space of the LLM and allows semantic misalignments to be corrected through full-context reinterpretation rather than incremental patching.

A comparison between the two experiments reveals several notable differences in reliability and convergence. Under static test-case constraints, regeneration succeeded in the majority of cases but failed to repair deeply rooted structural or logical misinterpretations, leading to 9 unresolved failures. Conversely, dynamic paired generation achieved full convergence by allowing the model to reframe both artefacts simultaneously. This behavior suggests that semantic misalignment between test cases and DMN logic is more easily corrected when both are re-generated together, as the model has an opportunity to reinterpret the rules holistically rather than attempting to align a new DMN table to a fixed set of examples.

Convergence dynamics also differed. In Experiment 1, repeated attempts often reproduced similar structural errors or semantic deviations, indicating that the model was stuck in a narrow interpretation space. In

Experiment 2, mismatches were typically resolved within one or two attempts, confirming that paired generation expands the model's sampling space and reduces the likelihood of repeated errors.

From a practical perspective, the dynamic configuration provides significantly higher reliability at the cost of slightly longer individual cycles. However, given that the longest failing sequences were entirely absent in Experiment 2, its overall runtime remains favourable. The difference between a 95.5% reliability ceiling (Experiment 1) and 100% reliability (Experiment 2) is crucial in automated DMN synthesis, where even rare failures undermine trust in the system's ability to generate decision logic without human supervision.

The experimental results demonstrate that deterministic validation is essential for ensuring the correctness of LLM-generated DMN tables. Under static test-case constraints, validation exposes structural and semantic errors that persist across regeneration attempts, limiting overall reliability. Under dynamic paired generation, independent interpretations of the rule set converge reliably, enabling complete automation of DMN-table synthesis. The 100% success rate of Experiment 2 validates the conceptual design of the framework and confirms that the combination of schema injection, structured prompting, and execution-based validation can support non-expert users in producing correct decision models from natural language.

Conclusions

The experiments provide insight into how reliably a large language model can translate natural-language business rules into executable DMN decision tables when supported by a validation-driven workflow. The comparison between the two configurations highlights that correctness depends not merely on the generative capacity of the model but on the structure of the validation loop surrounding it. In the static setting, where test cases remain fixed, the model performs well initially yet occasionally produces structural or semantic inconsistencies that cannot be resolved through repeated regeneration. These observations suggest that once the model forms an incorrect internal interpretation of the rules, additional attempts offer little benefit. This behaviour aligns with earlier findings that LLMs can repeat certain types of errors across attempts when exposed to identical prompts.

The dynamic configuration demonstrates a different pattern. When both test cases and the DMN table are regenerated together, the system consistently converges to correct outputs. The paired-generation strategy encourages the model to reinterpret the rule set holistically on each attempt, which appears to reduce the likelihood of persistent misalignment. The complete absence of irrecoverable failures indicates that semantic agreement between independently generated artefacts is a stronger determinant of correctness than structural adherence alone. This result reinforces the idea that semantic coherence, rather than prompt rigidity, is the primary driver of reliability in LLM-based decision-logic generation. The findings also illustrate that deterministic validation plays a central role in achieving stable results. Rather than functioning as a simple quality-assurance step, validation actively shapes the behaviour of the generation process. In the static setting, validation exposes systematic errors; in the dynamic setting, it guides the system toward convergence by enforcing consistency between artefacts. The distinction between these configurations shows that the design of the validation loop is as important as the design of the prompts themselves.

The broader implication of these observations is that fully automatic DMN-table generation is feasible when generation and validation are tightly integrated. The dynamic configuration demonstrates that a non-expert user can obtain a correct decision table without manual adjustments, provided that the system evaluates the model's interpretations against each other rather than against a fixed reference. This suggests a practical path toward lowering the entry barrier for decision-modelling technologies, making the construction of executable decision logic more accessible and less error-prone. Overall, the discussion highlights that model reliability emerges not from eliminating stochastic behaviour but from structuring it. By allowing the model to reinterpret the rule set and by verifying consistency across independent outputs, the pipeline transforms probabilistic generation into predictable, verifiable decision logic. These results contribute to a more general understanding of how LLMs can be embedded into formal modelling workflows where correctness is essential.

The study examined the feasibility of generating executable DMN decision tables from natural-language rules by integrating a large language model into a validation-driven workflow. The results show that correctness cannot be guaranteed through prompting alone, even when the required structure is explicitly defined. Structural validity and semantic accuracy emerge only when LLM outputs are treated as candidates rather than final artefacts and are systematically verified through execution. Across both experiments, the central insight is that the design of the validation loop fundamentally determines the reliability of the pipeline. The static configuration reveals that fixed test cases expose persistent misinterpretations that the model cannot correct through regeneration. In contrast, the dynamic configuration demonstrates that reinterpreting the rule set in parallel – by regenerating both test cases and the DMN table – creates conditions under which semantic consistency is achieved reliably. This behaviour suggests that convergence depends on the interaction between independently generated artefacts rather than on stricter prompting or increased regeneration limits.

The findings indicate that a fully automated DMN-generation pipeline is achievable when schema injection, controlled prompting, and execution-based validation are combined. The approach reduces the need for expert intervention and offers a practical method for constructing correct decision logic from textual rule descriptions. More broadly, the results contribute to ongoing efforts to integrate LLMs into formal modelling tasks, showing that

reliability arises from workflow design rather than from improvements in the model alone. Future work may extend this methodology to larger rule sets, multi-table decision models, and alternative validation strategies, as well as investigate how retrieval-augmented context influences semantic convergence. However, the present evaluation already demonstrates that dynamic artefact generation coupled with deterministic validation provides a viable foundation for dependable, accessible DMN modelling.

Author Contributions

Conceptualization, V.M.; methodology, V.M.; software, V.M.; validation, V.M.; formal analysis, V.M.; investigation, V.M.; resources, V.M.; data curation, V.M.; writing – original draft preparation, V.M.; writing – review and editing, V.M.; visualization, V.M.; supervision, V.M.; project administration, V.M.; funding acquisition, V.M. The author has read and agreed to the published version of the manuscript.

Declaration on the Use of Generative Artificial Intelligence Tools

In preparing this work, the author used Gemini (Google) solely for language-related assistance, including improving sentence clarity and English language usage, without generating new content or sections of the manuscript. Additionally, Grammarly was used for grammar, spelling, and punctuation checks. All scientific content, structure, and interpretation were produced by the author. After using these tools, the author reviewed and edited the text and takes full responsibility for the content of this publication.

References

1. Goossens, A., Vandevelde, S., Vanthienen, J., & Vennekens, J. GPT-3 for Decision Logic Modeling. RuleML+RR. Arco García L., Nápoles G., Vanhoenshoven F., Lara A. L., Casas G., Vanhoof K. Natural Language Techniques Supporting Decision Modelers. Data Mining and Knowledge Discovery. 2023. Vol. 35, Art. 5. DOI: https://doi.org/10.1007/s10618-020-00718-4
2. Tang X., Zhang Y., Bai J., Zhang H., Yu D. Struc-Bench: Are Large Language Models Really Good at Structured Understanding? arXiv preprint arXiv:2403.10660. 2024. URL: https://arxiv.org/abs/2403.10660
3. Goossens L., de Weerdt J., Vanthienen J. Extracting Decision Model and Notation Models from Text Using Deep Learning Techniques. Decision Support Systems. 2022. Vol. 211. Art. 118667. DOI: https://doi.org/10.1016/j.eswa.2022.118667
4. Bork D., van der Aa H., Brown A. I., van der Aalst W. M. P., Vanthienen J. AI-Enhanced Hybrid Decision Management: Combining Decision Models and Machine Learning. Enterprise, Business-Process and Information Systems Modeling (EMMSAD 2024). Lecture Notes in Business Information Processing. 2024. Vol. 507. DOI: https://doi.org/10.1007/s12599-023-00790-2
5. Barbara N., Guarascio M., Leone N., Veltri P., Zangari N. Neuro-Symbolic AI for Compliance Checking of Electrical Control Panels. DOI: https://doi.org/10.48550/arXiv.2305.10113
6. Wei J., Wang X., Schuurmans D. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Advances in Neural Information Processing Systems (NeurIPS). 2022. Vol. 35. DOI: https://doi.org/10.48550/arXiv.2201.11903.
7. Abedi, S., & Jalali, A. DMN-Guided Prompting: A Low-Code Framework for Controlling LLM Behavior. 2025. arXiv preprint arXiv:2505.11701. DOI: https://doi.org/10.48550/arXiv.2510.16062
8. Zhang G. L., Qiu R., Drechsler R., Schlichtmann U., Li B. Can LLMs Correct Themselves? A Benchmark of Self-Correction in LLMs. Preprint. 2025. URL: https://www.researchgate.net/publication/396714832_Can_LLMs_Correct_Themselves_A_Benchmark_of_Self-Correction_in_LLMs
9. Qiu R., Zhang G. L., Drechsler R., Schlichtmann U., Li B. CorrectBench: Automatic Testbench Generation with Functional Self-Correction Using LLMs for HDL Design. arXiv preprint arXiv:2411.08510. 2024. URL: https://arxiv.org/abs/2411.08510
10. Kampik, T., Warmuth, C., Rebmann, A. et al. Large Process Models: A Vision for Business Process Management in the Age of Generative AI. 2025. Künstl Intell 39, 81–95. DOI: https://doi.org/10.1007/s13218-024-00863-8
11. Rupnik, D., & Avsec, S.. Toward a Coherent AI Literacy Pathway in Technology Education: Bibliometric Synthesis and Cross-Sectional Assessment. Education Sciences, 2025. 15(11), 1455. DOI: https://doi.org/10.3390/educsci15111455

Vladyslav Maliarenko **Владислав Маляренко**	PhD student, National Technical University "Kharkiv Polytechnic Institute" e-mail: Vladyslav.Maliarenko@cs.khpi.edu.ua https://orcid.org/0009-0009-6064-061X Scopus Author ID: 60193671700	аспірант, Національний технічний університет «Харківський політехнічний інститут»