LLM-Based Automated State Machine Generation and Assessment

Mohamed Abdelrahman Ibrahim*†, Tiffany Miller*†, Marc-Antoine Nadeau*†, Rehean Thillainathalingam*†, Boqi Chen‡, Gunter Mussbacher†

†*Electrical and Computer Engineering, McGill University, Montreal, Canada*

‡*School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada*

Email: {mohamed.abdelrahmanibrahim, tiffany.miller, marc-antoine.nadeau, rehean.thillainathalingam}@mail.mcgill.ca, boqi.chen@uottawa.ca, gunter.mussbacher@mcgill.ca

***Abstract*—UML state machine modeling is a critical activity for the validation/verification of requirements and the specification of dynamic system behavior. Traditionally, state machines are manually crafted and assessed by experienced engineers based on natural-language requirements — a time-consuming and error-prone procedure. While many structured natural-language-based automated generation approaches exist, automated assessments have received little attention. In this paper, we empirically investigate the capabilities of state-of-the-art Large Language Models (LLMs) to (i) fully automate UML state machine generation from non-structured natural-language requirements, and (ii) automatically assess generated outputs by comparing them to a ground-truth solution. We first use human assessment to evaluate the generation quality of `Claude Sonnet 4.5` using the single-stage baseline and a two-stage prompting approach. We then compare three LLMs — `Claude Sonnet 4.5`, `GPT-5.5`, and `Gemini 3.1 Pro Preview` — by evaluating their assessment performance (i) against human assessment and (ii) relative to each other for state machines without human assessment. Our experiments indicate that the two-stage prompting approach improves generation quality (F_1-score 0.898) over the single-stage baseline (F_1-score 0.775), driven by recall gains while precision remains very high across both configurations (> 0.91), confirming that the generation gap is one of coverage rather than correctness. Furthermore, our assessment comparison shows that all LLMs do not reach assessment levels required for full automation, exhibiting a systematic conservative bias by missing credit that human raters award rather than fabricating it. Based on these results, we suggest a hybrid assessment strategy. Our evaluation highlights both the potential and the limitations of current LLMs for automated state machine generation and assessment, providing a baseline for future research in this domain.**

***Index Terms*—state machine, large language models, prompt engineering, model generation, model assessment**

I. Introduction

Context: UML state machine modeling is an important activity for requirements validation/verification and the specification of dynamic behavior of systems through states, transitions, and hierarchical features such as parallel regions, composite states, and history states [1]. Traditionally, state machines are manually crafted and assessed in a labor-intensive and error-prone process by experienced engineers who interpret natural language (NL) system descriptions to produce and assess comprehensive models. Existing automated generation approaches typically require structured NL descriptions [2]–[5]. At the same time, automated assessment approaches have received little attention [6]. This gap motivates the need for novel approaches that can accurately capture system behavior from non-structured textual descriptions.

Problem Statement: In this paper, we first address the problem of *fully automated* state machine generation. Given non-structured NL system descriptions, our goal is to automatically derive a state machine that accurately represents the system's behavior, including states, events, transitions, guards, actions, parallel regions, composite states, and history states, without any human interaction. Second, we address the problem of *automated* assessment of a state machine given a ground truth.

While Large Language Models (LLMs) have not been applied to the assessment of state machines to the best of our knowledge, advancements in LLMs have shown they are capable of automating the modeling of many different types of software models, such as domain models [7], [8], goal models [9], and sequence diagrams [10]. Although these LLMs demonstrate excellent understanding of input system descriptions, they still struggle to produce high-quality models fully autonomously [7]. Multi-step prompting techniques such as Chain-of-Thought [11] and Tree-of-Thought [12] involve breaking down a complex problem into multiple sub-problems and invoking the LLM multiple times for each subtask. These strategies have been shown to improve LLM performance for automating modeling tasks [13].

Recently, frontier LLMs such as Anthropic's `Claude Sonnet 4.5` [14], OpenAI's `GPT-5.5` [15], and Google's `Gemini 3.1 Pro Preview` [16], [17] represent the current state of the art in general-purpose language understanding and generation. These LLMs differ in their architectural emphasis, training strategies, and context-handling capabilities. However, their effectiveness in software modeling tasks, especially for state machine generation and automated assessment, has not yet been systematically studied or compared.

Objectives: Our study aims to first evaluate the extent to which a state-of-the-art reasoning LLM can be harnessed to *fully automate* the state machine modeling process using non-structured NL system descriptions. We build our approach on

* All four authors contributed equally to this research.

prior work on LLM-based state machine generation, which shows that LLMs can reasonably well identify basic states and transitions but consistently fail to capture more complex model elements such as guards, composite states, and interactions between multiple regions [18]. We update this previous work to more recent LLMs and a more common output format based on Mermaid syntax [19].

Second, we investigate to what extent the assessment of a state machine can be automated given a ground-truth state machine, comparing the assessment performance of `Claude Sonnet 4.5`, `GPT-5.5`, and `Gemini 3.1 Pro Preview` against human assessment. Furthermore, we evaluate their assessment performance relative to each other for state machines without human assessment.

Contributions: Specifically, this paper makes the following contributions to the modeling community:

- We evaluate the generation performance of `Claude Sonnet 4.5` for the single-stage baseline and a two-stage prompting approach to fully automate state machine generation.
- We evaluate LLM performance based on quantitative metrics (precision, recall, and F_1-score) obtained from a dataset of non-structured NL system descriptions with corresponding ground-truth state machines using human assessment and weighted Cohen's kappa.
- We compare the automated state machine assessment performance of `Claude Sonnet 4.5`, `GPT-5.5`, and `Gemini 3.1 Pro Preview` against human assessment using weighted Cohen's kappa, and further provide a detailed analysis of where LLM-based assessment aligns with and diverges from human judgment.
- Our experiment results provide insights into the strengths and limitations of current frontier LLMs for state machine generation and assessment, establishing a valuable baseline for future research in automated model generation and model assessment.

Added Value: Basic and Vujasinovic [20] also investigate LLM-based UML state diagram generation, but evaluate only three examples using qualitative ordinal scales without breaking down performance by model element type. In contrast, our work evaluates nine benchmark problems with fine-grained metrics per element type and additionally investigates automated LLM-based assessment.

In this context, we explore the single-stage baseline and a two-stage generation approach with a state-of-the-art reasoning LLM. Furthermore, we present a hybrid assessment strategy based on a comparison of human assessment of state machines against three frontier LLMs given ground-truth state machines.

II. Background

State Machine Modeling. UML state machines [1] are used to model reactive systems, capturing how a system transitions between states and performs actions in response to events and considering guard conditions. UML state machines extend basic state machines with features like hierarchically nested states (composite states), orthogonal regions for parallelism, and history states to remember prior substates, making them particularly suitable for specifying reactive and event-driven behaviors of software and embedded systems. The significant expertise required for state machine modeling, which is also time-consuming and prone to human error, motivates automated state machine modeling: if non-structured NL requirements or scenarios could be translated into state machines automatically, it would save manual effort and reduce errors.

Prompting Techniques. Fine-tuning an LLM involves updating some of its parameters to fit a specific task and dataset. This practice is more common with smaller-scale LLMs. However, with larger LLMs ($>$ 1B Parameters), this approach becomes harder and less common due to training costs and lack of sufficient training data. Therefore, we adopt two alternative prompting techniques to help guide the LLM through the state machine generation process. **Few-shot Prompting** [21] incorporates N labeled examples into the task context alongside the task description and input. However, solely using input/output pairs in these examples may not suffice for the LLM to learn complex tasks that require multiple steps to solve. **Chain-of-Thought Prompting** [11] addresses this issue by including a sequence of reasoning steps in the examples provided to the LLM. This approach enables the LLM to generate reasoning alongside the task output and has been shown to improve performance on various benchmarks compared to few-shot techniques. We use a combination of both techniques to enhance performance.

III. Approach

In this section, we describe our LLM-based framework for automatically generating UML state machine models from NL descriptions of reactive systems, followed by the automatic assessment framework that evaluates the generated state machine. We then present the employed LLMs.

A. Overview

Figure 1 provides an overview of the proposed framework. The framework takes a *system description* in natural language as input and produces a UML state machine model in an extended Mermaid syntax [19] as output. The generated Mermaid state machine is then parsed using a custom Mermaid parser extended for our use case, converted into an intermediate representation, and rendered into diagrams. In addition, the framework includes an automatic assessment component that evaluates generated Mermaid state machines against a pre-established ground truth assessment specification.

The framework comprises three modules. The first two are *generation modules*, each implementing a distinct prompting strategy for state machine production: a *single-stage generation module*, in which the LLM produces the complete UML state machine in a single prompt, and a *two-stage generation module*, in which an initial state machine is first generated and then passed to a second refinement prompt that reviews and corrects common structural and semantic issues. The third module is an *assessment module*, which takes a generated

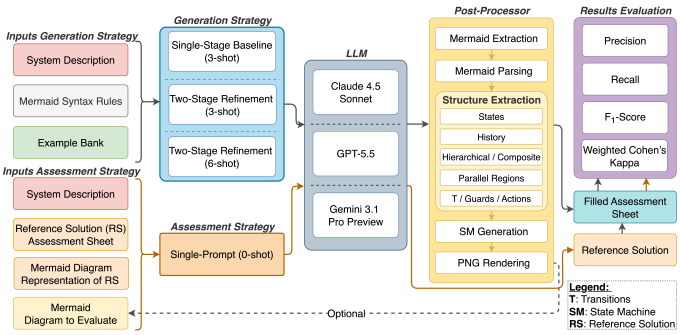


Fig. 1: Architecture for LLM-based automated state machine generation and assessment.

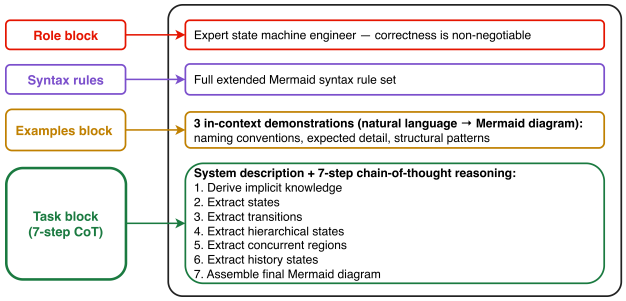


Fig. 2: Prompt structure of the single-stage baseline.

Mermaid state machine as input and produces a completed assessment sheet by filling in a pre-established rubric based on a reference solution state machine, enabling evaluation of state machines without requiring human intervention.

Single-Stage Baseline is a generation strategy with a 3-shot technique in which the full task is provided to the LLM in a single step. The prompt is structured into four distinct components which are illustrated in Figure 2.

First, a *role block* establishes the LLM's persona as an expert state machine engineer and frames correctness as non-negotiable. Second, a *syntax rules block* specifies the complete set of constraints imposed on the custom `stateDiagram-v2` Mermaid syntax parser [22]. The standard Mermaid syntax already supports core state machine constructs used in this work, including state declarations, initial states, transitions, and composite state. However, it does not natively support several UML state machine features required for this work. To address these limitations, we introduced a lightweight syntax extension that adds support for transitions containing explicit *event*, *guard*, and *action* elements using the notation `SourceState --> TargetState : event [guard] / action`, as well as state-level *entry*, *do*, and *exit actions* represented using Mermaid `note` blocks, *history states* represented using the reserved state name `H`, and parallel *regions* enclosed using a custom `region RegionName{...}` construct. Third, an *examples block* supplies three in-context demonstrations, each pairing a natural language system description with its corresponding valid Mermaid diagram. These examples establish the expected level of

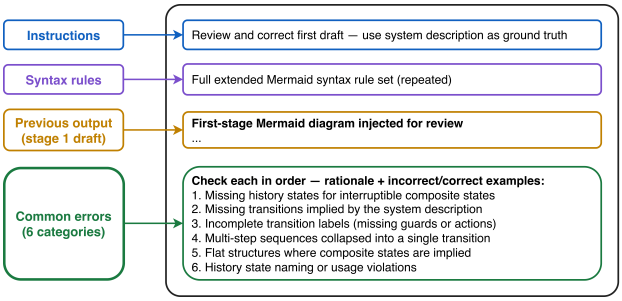


Fig. 3: Prompt structure of two-stage refinement.

detail, naming conventions, and structural patterns. Fourth, a *task block* provides the target system description and instructs the LLM to work through seven explicit reasoning steps: deriving implicit knowledge, extracting states, extracting transitions, identifying hierarchical structures, identifying concurrent regions, identifying history states, and finally assembling the complete Mermaid diagram.

Two-Stage Refinement is a generation strategy with a 3- or 6-shot configuration in which state machine production is decomposed into two successive LLM calls. The first stage is identical to the *Single-Stage Baseline*, as illustrated in Figure 2: the LLM receives the same four-block prompt—role, syntax rules, in-context examples, and task—and produces an initial Mermaid diagram using the same chain-of-thought reasoning steps. The 3-shot configuration of this generation strategy is identical to the *Single-Stage Baseline*, while the 6-shot configuration supplies three additional in-context examples in the examples block. The first-stage output is treated as a structured draft to be critically reviewed in the second stage. The second stage employs a refinement prompt designed to address recurring structural errors identified during preliminary experiments and evaluation of the Single-Stage Baseline. By targeting these issues, the refinement prompt aims to improve the quality, consistency, and completeness of the final output. This prompt is organized into four components, as illustrated in Figure 3.

First, an *instructions block* reframes the LLM's objective: rather than generating from scratch, the LLM is asked to review and correct its previous output using the original system description, which is provided again in the instructions block as the sole ground truth. Second, a *syntax rules block* repeats the full extended Mermaid syntax rule set. Third, a *previous output block* injects the first-stage diagram directly into the prompt for refinement. Fourth, a *common errors block* enumerates six specific error categories that the LLM must check against in order: (1) missing history states for composite states that can be interrupted and resumed; (2) missing transitions implied by the system description; (3) incomplete transition labels, specifically absent guards or actions; (4) multi-step behavioral sequences collapsed into a single transition; (5) flat structures where composite states are implied by the description; and (6) history state naming or usage violations, such as suffixed names like `H_Mode` that cause parse failures. For

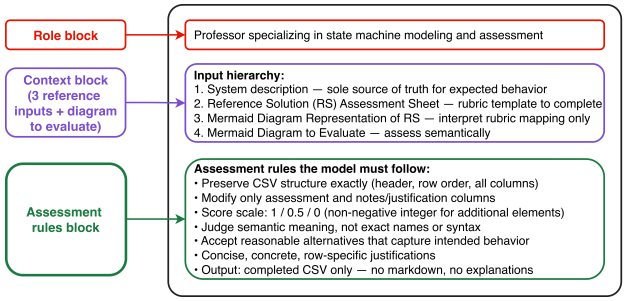


Fig. 4: Prompt Structure of the assessment module.

each error category, the block provides a rationale explaining why the error is harmful and a pair of incorrect and correct Mermaid examples illustrating the expected fix.

Single-Prompt Assessment is an assessment module in which an LLM evaluates a submitted state machine, represented in the extended Mermaid syntax, with a zero-shot configuration using three reference inputs, as illustrated in Figure 4: (1) the system description provided during the generation phase, (2) an assessment sheet in CSV format that decomposes an expert-created reference state machine solution derived from the natural language system description into atomic assessment components, and (3) the corresponding Mermaid diagram representation of the reference state machine solution. This module may be executed automatically following a generation phase, where the Mermaid representation of the state machine produced by the post-processor in Figure 1 is used as the input Mermaid diagram to evaluate. The prompt is structured into three distinct components.

First, a *role block* establishes the LLM's persona as a professor specializing in state machine modeling and assessment, framing the task as high-precision rubric completion. Second, a *context block* defines the hierarchy and interpretation of the provided inputs: the system description is treated as the sole source of truth for expected behavior, the reference solution assessment sheet serves only as a rubric template defining the expected assessment coverage, the Mermaid diagram representation of the reference solution is used solely to interpret how the rubric maps to the system description rather than to force literal similarity, and the Mermaid diagram to evaluate is assessed semantically against the intended behavior. Third, an *assessment rules block* specifies the constraints and evaluation rules the LLM must follow, including exact CSV structure preservation, modifications limited to assessment and justification columns, semantic rather than syntactic evaluation, acceptance of behaviorally equivalent modeling alternatives, score scales for standard and additional-element rows, concise row-specific justifications, and validation requirements.

B. Large Language Models

In our framework, we use three LLMs for state machine generation and assessment: `Claude Sonnet 4.5`, developed by Anthropic; `GPT-5.5`, developed by OpenAI; and `Gemini 3.1 Pro Preview`, developed by Google, from here-on referred to as `Claude`, `GPT`, and `Gemini`. However, the framework is not limited to these LLMs and can be extended to support other LLMs.

`Claude` has demonstrated state-of-the-art performance in coding, agentic task execution, and sustained multi-step reasoning [14], making it a strong candidate for automated state machine generation, which we evaluate with human assessment. We selected Claude based on preliminary experiments and prior work [18]. We further evaluate the assessment performance of three frontier LLMs on the state machines generated by `Claude` by comparing it against the human assessment: (i) `Claude`, which brings the same reasoning strengths to the assessment task, (ii) `GPT`, which has demonstrated strong capabilities in long-context reasoning, planning and iterative refinement [15], and (iii) `Gemini`, which introduces strong core reasoning capabilities for complex problem-solving [16], [17]. We further evaluate the assessment of state machines generated with `GPT` and `Gemini` not to compare them against human assessment, but rather to investigate assessment trends for these three LLMs.

The temperature parameter [23] controls the determinism of LLM outputs and was set differently across modules: 0.01 for first-stage and single-stage generation to encourage syntactic consistency, 0.3 for second-stage refinement to allow greater structural flexibility, and 0 for automatic assessment to minimize potential output variation.

IV. Manual State Machine Evaluation

We discuss the evaluation procedure, scheme, and criteria, as well as inter-rater agreement and confusion matrices.

A. Evaluation Procedure

To evaluate the state machine generation strategies proposed in this paper, we adopt an ideal scenario where the system description is clear and concise. This evaluation is based on nine modeling problem descriptions in English (see Table I) from an undergraduate university course, designed to assess students' proficiency in state machine modeling. Each problem comes with a reference state machine in diagrammatic format, created by modeling experts, which serves as the ground truth for comparison. Our evaluation involves comparing the

TABLE I: Summary of state machine model elements in ground-truth solutions.

Model Element	P	TA	SM	D	C	B	TM	W	S	Total
Action	3	12	0	7	8	10	6	36	21	103
Composite State	2	2	3	2	3	3	1	5	1	22
Guard	6	14	4	4	4	4	7	3	11	57
History State	1	1	1	1	1	1	1	1	1	9
Parallel Region	0	4	5	2	2	0	0	2	0	15
States	8	17	16	12	13	12	11	24	9	122
Transition	17	20	15	19	14	18	17	37	23	180
Total	**37**	**70**	**44**	**47**	**45**	**48**	**43**	**108**	**66**	**508**

P: Printer, TA: Train Automation, SM: Spa Manager,
D: Dishwasher, C: Chess Clock, B: Bread Maker,
TM: Thermomix TM6, W: W-UMPLE, S: SSC7.

generated models to these reference solutions to assess the effectiveness of our strategies.

Due to the subjective nature of software modeling, multiple state machines may correctly model the same behavior for a given problem description. For example, state names, composite state names, or parallel region names may differ while preserving equivalent transitions and overall system behavior. Furthermore, to the best of our knowledge, no established automated evaluators currently exist for UML state machines, motivating the need for a structured manual evaluation process.

To support reproducible evaluation, we created a dedicated rubric in the form of an assessment spreadsheet template for each benchmark problem. Generated outputs are evaluated against the corresponding ground-truth solutions using the predefined assessment templates.

For each generated state machine, two independent raters evaluate the outputs without consulting one another in order to reduce evaluation bias. After evaluation, inter-rater agreement is computed using weighted Cohen's kappa [24]. When the agreement score exceeds 0.8, the evaluations are considered sufficiently consistent; otherwise, the evaluators jointly reconcile disagreements to produce a final evaluation.

In addition to manual evaluation, the automated assessment module is provided with the same assessment template scheme in CSV format used by the human evaluators and tasked with assessing the generated state machines while preserving the original rubric structure.

B. Evaluation Scheme

When evaluating a state machine generated by the generation strategies, we considered the seven types of state machine model elements that we deemed to be the most characteristic or representative of the state machine modeling decisions. These seven types of model elements are state, transition, guard, action, composite state, parallel region, and history state. The distribution of these types of model elements across the ground-truth state machines is shown in Table I.

For each benchmark problem, the ground-truth UML state machine is decomposed into these atomic types of model elements. Each expected type of model elements is represented as an individual assessment entry and evaluated independently with scores of 1 (correct), 0.5 (partially correct), or 0 (incorrect or missing). Additionally, a separate section is included for each type of model elements to record additional generated elements that are not present in the reference solution and are inconsistent with the system description. These additional elements are counted independently and treated as false positives during metric computation.

The evaluation emphasizes semantic correctness rather than exact textual similarity. Generated elements may still be considered correct even if they differ from the reference solution, provided that they represent equivalent behavior or structure.

C. Evaluation Criteria

To evaluate the quality of the generated model, we consider partial results ($S = \sum_i s_i TP_i$, where $s_i = (0.5, 1)$ for each TP) and compute the precision ($P = \frac{S}{TP+FP}$), recall ($R = \frac{S}{TP+FN}$), and F_1-scores ($F_1 = \frac{2 \times P \times R}{P+R}$) for each of the seven types of state machine model elements detailed in Section IV-B for each generated state machine, in addition to the overall result (computed by aggregating the true positives (TP), false positives (FP), and false negatives (FN) of all types of model elements). The overall F_1-score is calculated to estimate the overall quality of the output of a generation strategy. The individual metrics for the seven types of state machine model elements provide more detailed and interpretable results, as well as hints as to the next aspects to optimize with subsequent generation strategies. Precision gives an estimate of the accuracy of predictions made by the generation strategy. Recall measures the ability of the generation strategies to find all relevant instances of the concerned model elements in the ground-truth model. The F_1-score combines the precision and recall measures, i.e., representing their harmonic mean.

D. Inter-Rater Agreement and Confusion Matrices

To measure assessment consistency between raters, we use *weighted Cohen's kappa* ($\kappa_w = 1 - \frac{\sum_{i,j} w_{ij} o_{ij}}{\sum_{i,j} w_{ij} e_{ij}}$), where o_{ij} are the observed proportions and e_{ij} the expected proportions under chance agreement for category pair (i, j), and the disagreement weights $w_{ij} = \frac{|s_i - s_j|}{\max(s) - min(s)} \equiv |s_i - s_j|$ for scores $s \in \{0, 0.5, 1\}$ (see Section IV-B) penalize disagreements proportionally to their distance [24]. Inter-rater agreement is computed both globally and per model element type category.

In addition, confusion matrices are used to analyze the distribution of assessment disagreements, providing a detailed breakdown of score pairings and enabling comparison of assessment tendencies across categories and score assignments.

V. Experiments

This section presents the experimental setup and results (see repository [25]) for the three research questions, followed by a discussion including threats of validity. The three research questions are: **RQ1:** How well does a state-of-the-art reasoning LLM generate state machines using a single-stage and two-stage approach? **RQ2:** To what extent can the assessment of state machines be automated with state-of-the-art LLMs? **RQ3:** To what extent does the assessment trend observed in RQ2 hold for state machines generated by other LLMs?

A. Dataset and Setup

All experiments use the nine problems described in Table I. For RQ1, `Claude` generates state machines under three configurations: the single-stage baseline (3-shot), and two two-stage variants (3-shot and 6-shot). Few-shot examples use the same problems, except SSC7 and W-UMPLE are withheld to evaluate generalization. The evaluated problem is never included in the prompt. For 3-shot experiments, the examples are Printer, Spa Manager, and Dishwasher, with Chess Clock substituted whenever the evaluated problem is one of these. For 6-shot experiments, examples are drawn from the remaining few-shot examples, and when evaluating SSC7 or W-UMPLE, Train is also excluded. The 3-shot configurations

are evaluated on the seven non-blind problems, while the 6-shot configuration is additionally evaluated on the two blind problems, SSC7 and W-UMPLE.

For each generated state machine, two human raters independently evaluate the output using the scheme from Section IV. Inter-rater agreement is computed using weighted Cohen's kappa as a consistency measure, and all assessments are subsequently reconciled through discussion to produce a single agreed-upon final evaluation.

For RQ2, the two-stage 6-shot configuration is used as the basis for assessment comparison, as it produces the strongest generation results in RQ1. `Claude`, `GPT`, and `Gemini` each independently assess the same `Claude`-generated state machines against the reconciled human assessment.

For RQ3, `GPT` and `Gemini` each generate state machines with the two-stage 6-shot configuration, which are subsequently assessed by all three LLMs. No human assessment exists for these generated state machines. The goal is not to evaluate the quality of `GPT`- and `Gemini`-generated state machines, but to determine whether the assessment patterns observed in RQ2 remain consistent when the LLMs are applied to different state machines.

B. RQ1: Generation Quality of Claude Sonnet 4.5

The single-stage baseline reveals a consistent asymmetry: precision remains consistently very high (above 0.883, overall 0.911), while recall is considerably lower and more variable (see Table II). When `Claude` includes a model element, it is always at least partially correct relative to the ground truth (FP of 0 across all examples). `Claude` omits required elements rather than generating incorrect ones. History states show the lowest single-stage recall, though this reflects a small sample (see Table I) rather than a systematic failure. The more consequential gaps are in actions and guards, which appear frequently across problems and are most likely to be expressed implicitly or conditionally in a natural language description.

These observations shape the design of the two-stage refinement. Rather than rewriting the generation prompt, the second stage is built to recover what the single-stage consistently missed. The Two-Stage Refinement section of Section III-A describes these recurring shortfalls in detail. Actions, guards, and transitions, three element types with consistently weak single-stage recall and high practical importance, each improve by roughly 14 to 18 percentage points for F_1 under refinement.

TABLE II: Micro-average by element type and overall for single-stage and two-stage (6-shot) prompting. P, R, and F_1 computed across all examples. ΔF_1 shows improvement of two-stage over single-stage. Human assessment only.

	Single-stage			Two-stage			
Model Element	P	R	F_1	P	R	F_1	ΔF_1
Action	0.923	0.522	0.667	0.967	0.704	0.815	+0.148
Composite State	0.933	0.875	0.903	0.976	0.932	0.953	+0.050
Guard	0.904	0.547	0.681	0.929	0.798	0.858	+0.177
History State	1.000	0.429	0.600	0.722	0.722	0.722	+0.122
Region	0.950	0.731	0.826	1.000	0.867	0.929	+0.103
State	0.929	0.803	0.861	0.961	0.898	0.928	+0.067
Transition	0.883	0.662	0.757	0.962	0.903	0.931	+0.174
Overall	**0.911**	**0.674**	**0.775**	**0.956**	**0.846**	**0.898**	**+0.123**

P: Precision, R: Recall, F_1: F_1-Score.
Single-stage: $n = 7$ examples; Two-stage: $n = 9$ examples.

Table II summarizes the two-stage results broken down by model element type. Composite states, transitions, regions, and states perform strongest. Overall, two-stage F_1 across all model elements reaches 0.898 for the 6-shot configuration with FP of only 2 (for history states) across all examples, compared to 0.775 for the single-stage baseline. The two-stage 3-shot configuration achieves an overall F_1 of 0.884, confirming that the second refinement stage drives most of the improvement and that additional in-context examples provide a smaller but consistent further gain.

History states show the lowest single-stage F_1 at 0.600, and improve to 0.722 under two-stage prompting. However, this reflects a very small sample size, so the result must be interpreted with caution. Actions and guards remain the lowest performing types with large sample size despite large absolute improvements. Recall sits at 0.704 and 0.798, respectively, even after refinement, i.e., roughly one in four expected actions and guards are still missed. Both are frequently left implicit in natural language descriptions and must be inferred from context rather than extracted directly, and the second stage does not fully close that gap. This suggests a capability limitation beyond what prompt engineering alone can address.

Both blind test cases, SSC7 and W-UMPLE, produce an aggregated overall F_1 of 0.902, compared to 0.896 for the seven other examples, giving no indication that the reported results are inflated by repeated exposure to the examples set.

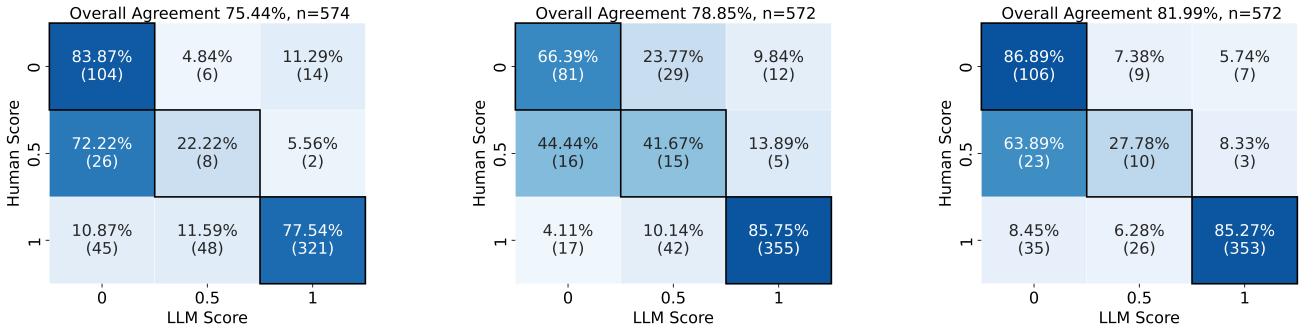
Answer to RQ1

The two-stage approach improves generation quality over the single-stage baseline for all seven model element types, driven by recall gains on the elements the refinement stage specifically targeted. Precision is consistently very high across both configurations (> 0.91) with FP of 2 across all examples, confirming that the generation gap is one of coverage rather than correctness. Actions and guards remain the hardest element types even after refinement, suggesting the LLM struggles to infer implicit behavior from non-structured descriptions regardless of prompting strategy.

C. RQ2: Automated Assessment Quality

Assessment is the primary scalability bottleneck in our experimentation. Each generated state machine requires two independent human raters and a reconciliation step, and that effort multiplies quickly as the number of examples and configurations grows. RQ2 examines whether LLM-based assessment can close this gap, and more specifically, where it agrees with human judgment and where it consistently strays.

Table III reports linear weighted Cohen's κ_w between each LLM and the reconciled human assessment, using the same



(a) Claude 4.5 Sonnet (b) GPT-5.5 (c) Gemini 3.1 Pro Preview

Fig. 5: Row-normalized confusion matrices comparing LLM and human assessment decisions across the three score options (0, 0.5, 1). Rows represent the human score; columns represent the LLM score.

TABLE III: Linear weighted Cohen's κ_w between each LLM and human assessment, overall and by model element type, evaluated on `Claude`-generated state machines using the two-stage 6-shot configuration.

Model Element	Claude 4.5 Sonnet	GPT-5.5	Gemini 3.1 Pro P.
Action	0.507	0.602	0.548
Composite State	0.647	0.893	0.826
Guard	0.658	0.615	0.720
History State	0.553	0.614	0.612
Region	0.834	0.915	0.834
State	0.451	0.648	0.693
Transition	0.551	0.529	0.655
Overall	**0.590**	**0.655**	**0.690**

weighting scheme applied for human inter-rater agreement in Section IV. `Gemini`, the best performing LLM, reaches only 0.690, i.e., medium moderate (κ_w = 0.60–0.79) agreement [26] with the human assessment. It is worth noting that human inter-rater kappa only exceeds 0.80 on these problems in 43.5% of all evaluated RQ1 problems, typically only when assessments are near-perfect, reflecting the inherent subjectivity of state machine evaluation rather than a ceiling that is easily reached.

At the element level, region and composite state exhibit strong agreement (κ_w = 0.80–0.90) [26], with region's κ_w values at 0.834, 0.915, and 0.834 for `Claude`, `GPT`, and `Gemini`, respectively. Composite state also shows strong agreement for `GPT` (0.893) and `Gemini` (0.826), though `Claude` lags at 0.647. Guard, transition, and history state reach moderate agreement, while action and state are the weakest element types, with the lowest result for state being 0.451 for `Claude` and action being the weakest overall across all three LLMs (weak level of agreement if κ_w = 0.40–0.59) [26].

The confusion matrices in Fig. 5 compare the consolidated human assessment and LLM assessment decisions directly. Each row represents the human score and each column represents the LLM score, so a cell at row 0.5 and column 0 means the human assessment is partial credit but the LLM's is zero. All three LLMs handle Score 1 well, with `GPT` and `Gemini` correctly identifying elements the human rated as correct at rates above 85%, and `Claude` at 77.5%. Identifying incorrect elements (Score 0) is also reasonable for `Claude` and `Gemini` at 83.9% and 86.9%, respectively, while `GPT-5.5` is weaker here at 66.4%, spreading disagreements into the 0.5 column instead. The consistent problem across all three LLMs is the Score 0.5 row. When a human assigns partial credit, all three LLMs tend to collapse that decision to Score 0. `Claude` does this most aggressively at 72.2%, `Gemini` less so at 63.9%, and `GPT` is the least biased at 44.4% though still unreliable. This pattern holds across three architecturally different LLMs from three different organisations, suggesting it is not a peculiarity of any one LLM. Without explicit examples of what a 0.5 assessment decision looks like in the prompt, the LLMs appear to default to the more certain binary options. Critically, all three LLMs miss credit that human raters award rather than fabricating credit that humans do not. The bias is conservative: LLM assessment systematically underestimates rather than overestimates how correct the generated state machines are.

Answer to RQ2

None of the three LLMs reach overall strong agreement (Cohen's $\kappa_w \geq 0.80$) with the consolidated human assessment. `Gemini` and `GPT` come closest at 0.690 and 0.655, respectively, with `Claude` only at 0.590. Region and composite state exhibit strong agreement for most LLMs, while action and state are the weakest element types across all LLMs. All three LLMs consistently miss credit that human raters award rather than fabricating credit that humans do not, meaning the bias is conservative.

D. RQ3: Assessment Trends on Further State Machines

For RQ3, all three LLMs assess the state machines generated by `GPT` and `Gemini`. We then compare the resulting assessment patterns against those observed for RQ2. Without human assessment for these state machines, the focus is on

TABLE IV: Score 1 rates (%) per model element type and overall for Claude-generated state machines (RQ2 baseline, human-validated) and the combined GPT- and Gemini-generated state machines (no human assessment). Results are pooled across both generators for the combined column. Ranks are shown in parentheses: element type ranks are within each LLM column; overall ranks are across LLM columns.

	C4.5S		GPT-5.5		G3.1PP	
Model Element	**C-gen**	**G+G-gen**	**C-gen**	**G+G-gen**	**C-gen**	**G+G-gen**
n	519	1141	510	1142	508	1142
Action	35.1 (6)	64.0 (4)	52.3 (6)	77.5 (3)	47.7 (5)	73.0 (3)
Composite State	58.1 (3)	62.9 (5)	58.1 (4)	67.7 (5)	58.1 (3)	69.4 (4)
Guard	45.5 (5)	68.2 (3)	60.6 (3)	69.7 (4)	54.5 (4)	68.2 (5)
History State	22.2 (7)	30.6 (7)	27.8 (7)	36.1 (7)	27.8 (7)	36.1 (7)
Region	45.8 (4)	51.1 (6)	54.2 (5)	54.2 (6)	45.8 (6)	50.0 (6)
State	73.6 (1)	83.2 (1)	76.3 (1)	85.1 (2)	75.6 (1)	84.7 (2)
Transition	71.6 (2)	81.6 (2)	72.6 (2)	85.3 (1)	74.2 (2)	86.8 (1)
Overall	58.5 (3)	73.1 (3)	65.1 (1)	78.1 (1)	63.6 (2)	77.4 (2)

C4.5S: Claude 4.5 Sonnet; G3.1PP: Gemini 3.1 Pro Preview.
C-gen: Claude-generated (human-validated); G+G-gen: combined GPT+Gemini-generated (no human assessment). All values are % Score 1.

assessment consistency rather than generation quality: do the same tendencies identified for RQ2, specifically the element-level ranking and the partial credit collapse, hold for further state machines? Table IV compares Score 1 rates (% of assessments with 1 as a score) on Claude-generated state machines against the combined GPT+Gemini-generated state machines for each LLM.

The Score 1 ranking from RQ2 (GPT most generous, Gemini middle, Claude strictest) holds for RQ3 as well. The gap between LLMs narrows slightly on the GPT+Gemini- vs. Claude-generated state machines (78.1%, 77.4%, 73.1% compared to 65.1%, 63.6%, 58.5%), but Claude remains the strictest in both conditions.

At the element level, the ranking of element types is broadly preserved across generator conditions. History states remain the lowest scoring type in both conditions for all three LLMs, never exceeding 36.1%. States and transitions remain the two highest, always at least above 71.6%. The most notable shift is for action, where all three LLMs assign substantially more Score 1 on the combined GPT+Gemini-generated state machines than on Claude-generated ones (> 25 percentage points higher). The magnitude is consistent across LLMs, which suggests this could reflect genuinely better action generation by GPT and Gemini. However, this could also be a surface-level alignment between their output style and the rubric, which cannot be determined without human assessment. Nevertheless, the shift is large enough to warrant further investigation, which we leave for future work. Beside action, guard also changes its rank once when assessed by Claude, but all other relative rankings stay the same, indicating that the RQ2 assessment trends hold for RQ3.

All three LLMs assign more Score 1 to the combined GPT+Gemini-generated state machines than to Claude-generated ones, with overall rates rising from 58.5–65.1% to 73.1–78.1%. Because this shift is consistent across all three LLMs, it is more likely a property of the generated state machines than of any individual LLM assessing them. However, these results again have to be reconfirmed through human assessment of the GPT+Gemini-generated state machines. Partial credit rates also drop across the board on the combined state machines (4.0–6.7%) compared to Claude-generated ones (7.9–15.1%), reinforcing the RQ2 finding that all three LLMs default to binary decisions. That tendency is even more pronounced for RQ3.

Answer to RQ3

The assessment patterns from RQ2 broadly hold when all three LLMs are applied to GPT+Gemini-generated state machines. The LLM ranking (GPT most generous, Claude strictest) is preserved, states and transitions remain the easiest element types, history states remain consistently the hardest element type, and the partial credit collapse persists. Score 1 rates are higher overall on the GPT+Gemini-generated state machines, but since no human assessment exists for these state machines, this cannot be interpreted as a quality comparison. The element-level ranking is largely stable across generator conditions, supporting the assessment patterns identified in RQ2.

E. Discussion

Across all three experiments, the same element types are consistently the hardest to generate and the least reliably assessed. Action and history state show the weakest F_1 values in RQ1, and they also show the lowest κ_w values in RQ2. Guards, transitions, and states fall well below strong agreement for RQ2, despite above 0.85 F_1-scores under two-stage refinement. Region and composite state are reliable on both sides, performing strongly in generation and reaching strong agreement for assessment. Region and composite state share a common property: they are structurally defined and directly extractable from a system description. Action and guard, by contrast, require inferring behavior that is often left implicit in natural language, which is equally hard whether the LLM is generating or evaluating.

For the Claude-generated state machines assessed in RQ2, all three LLMs miss credit rather than fabricate it, meaning LLM assessment does not inflate perceived generation quality on that set. Where an LLM assigns Score 1, that judgment is generally trustworthy. Where it assigns Score 0 on a semantically complex element, a human rater cannot be confident that judgment is correct. This asymmetry is what makes a hybrid strategy viable rather than speculative. Claude is consistently the strictest grader regardless of which LLM generated the state machine, while GPT is the most generous. This relative ranking is stable across both RQ2 and RQ3, holding regardless of which LLM generated the state machines, which suggests the choice of which LLM assesses state machines has a consistent effect on assessment outcomes.

While the ranking of element types is broadly preserved from RQ2 to RQ3, the large increase in Score 1 rates for action from RQ2 to RQ3, observed consistently across all three LLMs, is the one finding that does not fit cleanly into this picture. Whether this reflects genuinely better action generation or a surface-level alignment between the output style of the generator and the rubric cannot be resolved without human assessment. It requires human validation of `GPT+Gemini`-generated state machines specifically before drawing stronger conclusions about their generation quality.

These findings motivate a hybrid assessment strategy. Since generation precision is consistently very high and FP is either zero or near zero, elements the LLM includes are almost always at least partially correct. A human rater can therefore focus entirely on elements that were missed or received partial credit. Score 1 assignments can be reasonably assumed correct without further verification. Human review should concentrate on actions, guards, transitions, and states, where LLM reliability is lower and the partial credit collapse has the most impact, while regions can be left to automated assessment with confidence across all three LLMs. Composite states are reliable for GPT and Gemini but warrant spot-checking when using Claude. Validating this strategy against a human-assessed baseline is the natural next step for future work.

F. Threats to Validity

Internal validity. The non-zero generation temperatures introduce non-determinism, mitigated by low chosen values and averaging across nine problems. Manual evaluation bias is compensated by shared guidelines, two independent raters, weighted Cohen's kappa, and discussion-based reconciliation. **External validity.** All experiments use nine problems from a single undergraduate course, and the three LLMs evaluated constitute a limited sample, meaning results may not be generalized to other domains, description styles, languages, or LLMs. This is further constrained by only two blind test cases and the absence of human assessment for `GPT+Gemini`-generated state machines. **Construct validity.** Precision, recall, F_1-score [7], and weighted Cohen's kappa [26] are standard metrics, but the seven-element decomposition may not capture all quality dimensions. Moreover, the 0/0.5/1 scoring scale may introduce subjectivity in partial-credit decisions.

VI. Related Work

Automated state machine generation. Early studies primarily rely on statistical methods to extract state machines from structured software logs and execution traces [2]–[4]. More recent approaches incorporate various machine learning techniques, such as active learning, to improve extraction quality and efficiency [27], [28]. With the rise of LLMs, Wei et al. [29] propose ProtocolGPT which analyzes protocol source code to identify and reconstruct underlying state machines. Biase et al. [5] propose a semi-automatic approach to complete SysML state machines from partial models and Given-When-Then requirements. Abdulkarim et al. [18] investigate LLM-based state machine generation from unstructured NL descriptions, finding that LLMs handle basic states and transitions reasonably but struggle with guards, composite states, and parallel regions. Basic and Vujasinovic [20] evaluate GPT-4.1, Grok 3, and Qwen3.5-Plus for UML state diagram generation across three examples of varying complexity, reporting strong results for simpler systems but limitations in hierarchical modeling. Building on these efforts, we explore single-stage and two-stage generation strategies using more recent frontier LLMs and Mermaid syntax [19] as the output format.

LLMs for Model Generation. A frequent application of LLMs in MDE is the automatic generation of models from NL text [30]. Early studies typically use various prompting techniques, relying exclusively on a single LLM call to generate complete models, including domain models [7], [31], goal models [9], sequence diagrams [10], and use case models [32]. More recently, many multi-step model-generation methods based on iterative LLM interactions have been proposed, e.g., to integrate human best-practice modeling processes with iterative LLM-based domain model generation [13]. Additionally, the Chain-of-Layers approach [33] utilizes a top-down iterative approach, combining LLMs with masked language models to progressively construct taxonomies. This paper is an application of LLMs for model generation, focusing on state machines with single- and two-stage generation strategies.

LLMs for MDE. Besides complete model generation, LLMs have been integrated into various other aspects of MDE. A common use case is model completion, where LLMs assist human modelers by providing completions or suggestions during the modeling process [8], [34], targeting more efficient and accurate model construction. In this paper, we investigate the use of LLMs for state machine generation and assessment, focusing on single- and two-stage generation strategies. To our knowledge, this represents the first systematic study of LLM-based automated assessment of UML state machines.

Automated model assessment. Automated assessment of models has been explored primarily in software engineering education, covering DFA grading via edit distance and entropy [35], FSM problem generation and heuristic-based grading [6], autograding of DFA and NFA constructions with personalized feedback [36]. For UML models, automated grading of class diagrams [37] and embedding-based assessment of domain models against expert reference solutions [38] have been proposed. Automated assessment of UML state machines specifically has received little attention. In this paper, we investigate LLM-based automated assessment of UML state machines, comparing three frontier LLMs against human assessment and against each other.

VII. Conclusion

We present an LLM-based framework for the automated generation and assessment of UML state machines from non-structured NL descriptions, evaluated on nine benchmark problems with expert-created ground-truth solutions. We introduce and evaluate a single-stage baseline and a two-stage refinement approach using `Claude Sonnet 4.5`, and compare `Claude Sonnet 4.5`, `GPT-5.5`, and `Gemini 3.1`

`Pro Preview` as automated raters against human raters and relative to each other.

The two-stage approach substantially improves generation, raising the overall F_1-score from 0.775 to 0.898 by recovering omitted elements—particularly guards ($+0.177$), transitions ($+0.174$), and actions ($+0.148$) —while maintaining very high precision and very low FP. Actions and guards remain the weakest element types, with recall of only 0.704 and 0.798.

For automated assessment, no LLM reaches the threshold for full automation ($\kappa_w \geq 0.80$). `Gemini` performs best (0.690), ahead of `GPT` (0.655) and `Claude` (0.590). All LLMs exhibit a systematic conservative bias, collapsing partial-credit scores to zero in 44.4–72.2% of cases. Actions remain the weakest element type across all LLMs. Assessment patterns remain stable across generation-assessment configurations, with `GPT` the most generous and `Claude` the strictest, though the absence of human assessment prevents interpreting score differences across configurations as direct quality improvements.

Overall, iterative prompting enables high-quality generation, leveraging the very high precision and low FP results. However, current LLMs remain insufficiently reliable for fully automated assessment—motivating a hybrid human–LLM strategy using LLMs across the full rubric and targeted human review of semantically complex model elements.

References

[1] Object Management Group, "Unified Modeling Language Specification Version 2.5.1," December 2017, https://www.omg.org/spec/UML/2.5.1/.

[2] J. E. Cook and A. L. Wolf, "Discovering models of software processes from event-based data," *TOSEM*, vol. 7, no. 3, pp. 215–249, 1998.

[3] G. Ammons, R. Bodík, and J. R. Larus, "Mining specifications," *ACM Sigplan Notices*, vol. 37, no. 1, pp. 4–16, 2002.

[4] D. Lorenzoli, L. Mariani, and M. Pezzè, "Automatic generation of software behavioral models," in *30th International Conference on Software Engineering*, 2008, pp. 501–510.

[5] M. S. de Biase et al., "Completion of sysml state machines from given–when–then requirements," *Software and Systems Modeling*, vol. 23, no. 6, pp. 1455–1491, 2024.

[6] D. Kashyap, M. Bhat, and N. Kumar, "Automated generation and grading for finite state machine assignments," in *Proceedings of the 15th Annual ACM India Compute Conference*, ser. COMPUTE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 8–12. [Online]. Available: https://doi.org/10.1145/3561833.3561839

[7] K. Chen et al., "Automated domain modeling with large language models: A comparative study," in *ACM/IEEE 26th Intl. Conf. on Model Driven Eng. Languages and Systems (MODELS)*, 2023, pp. 162–172.

[8] J. Cámara et al., "On the assessment of generative AI in modeling tasks: An experience report with ChatGPT and UML," *Software and Systems Modeling*, vol. 22, no. 3, pp. 781–793, 2023.

[9] B. Chen et al., "On the use of GPT-4 for creating goal models: An exploratory study," in *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. IEEE, 2023, pp. 262–271.

[10] M. Jahan et al., "Automated derivation of UML sequence diagrams from user stories: Unleashing the power of generative AI vs. a rule-based approach," in *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, 2024, pp. 138–148.

[11] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in *NeurIPS*, vol. 35, 2022, pp. 24 824–24 837.

[12] S. Yao et al., "Tree of thoughts: Deliberate problem solving with large language models," in *NeurIPS*, vol. 36, 2023, pp. 11 809–11 822.

[13] Y. Yang et al., "Multi-step Iterative Automated Domain Modeling with Large Language Models," in *ACM/IEEE MODELS Companion 2024)*, 2024, p. 587–595.

[14] Anthropic, "Introducing Claude Sonnet 4.5," September 2025, accessed: May 12, 2026. [Online]. Available: https://www.anthropic.com/news/claude-sonnet-4-5

[15] OpenAI, "Introducing GPT-5.5," April 2026, accessed: May 12, 2026. [Online]. Available: https://openai.com/index/introducing-gpt-5-5/

[16] Google, "Gemini 3.1 Pro Preview," Febuary 2026, accessed: May 12, 2026. [Online]. Available: https://ai.google.dev/gemini-api/docs/models/gemini-3.1-pro-preview

[17] The Gemini Team, "Gemini 3.1 Pro: A Smarter Model for Your Most Complex Tasks," February 2026, accessed: May 13, 2026. [Online]. Available: https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/

[18] S. Abdulkarim, E. Boyd, K. Bridi, A. Tufenkjian, B. Chen, and G. Mussbacher, "Structure- and event-driven frameworks for state machine modeling with large language models," *arXiv preprint*, 2026. [Online]. Available: https://arxiv.org/abs/2604.00275

[19] Mermaid, "About mermaid," accessed: May 10, 2026. [Online]. Available: https://mermaid.js.org/intro/

[20] M. Basic and M. Vujasinovic, "A comparison of LLMs for UML state diagrams generation," *International Journal of Innovative Science and Research Technology*, vol. 11, no. 2, pp. 3134–3158, 2026.

[21] T. B. Brown et al., "Language models are few-shot learners," *arXiv preprint*, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[22] B. Chen, "mermaid-parser-py," 2026, accessed: May 15, 2026. [Online]. Available: https://github.com/reheant/mermaid-parser-py

[23] M. Renze, "The effect of sampling temperature on problem solving in large language models," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024, pp. 7346–7356.

[24] J. Cohen, "Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit." *Psychological Bulletin*, vol. 70, no. 4, pp. 213–220, 1968.

[25] M. Abdelrahman Ibrahim, M.-A. Nadeau, T. Miller, and R. Thillainathalingam, "Experiment Material," 2026, accessed: May 15, 2026. [Online]. Available: https://github.com/ma-nadeau/llm-automated-state-machine-generation-assessment

[26] M. McHugh, "Interrater reliability: the kappa statistic," *Biochemia Medica*, pp. 276–282, 01 2012.

[27] M. Isberner, F. Howar, and B. Steffen, "The open-source learnLib: a framework for active automata learning," in *Computer Aided Verification: 27th Intl. Conf., Part I 27*. Springer, 2015, pp. 487–495.

[28] N. Yang et al., "Improving model inference in industry by combining active and passive learning," in *2019 IEEE 26th Intl. Conf. on Software Analysis, Evolution and Reeng. (SANER)*. IEEE, 2019, pp. 253–263.

[29] H. Wei et al., "Unleashing the power of llm to infer state machine from the protocol implementation," in *2025 IEEE/ACM 33rd International Symposium on Quality of Service (IWQoS)*, 2025, pp. 1–10.

[30] J. Di Rocco et al., "On the use of large language models in model-driven engineering," *Software Systems Modeling*, vol. 24, pp. 923–948, 2025.

[31] S. Arulmohan, M.-J. Meurs, and S. Mosser, "Extracting domain models from textual requirements in the era of large language models," in *2023 ACM/IEEE Intl. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 2023, pp. 580–587.

[32] M. R. Tabassum et al., "Using LLMs for use case modelling of IoT systems: An experience report," in *ACM/IEEE 27th Intl. Conf. on Model Driven Eng. Languages and Systems – Companion*, 2024, pp. 611–619.

[33] Q. Zeng et al., "Chain-of-layer: Iteratively prompting large language models for taxonomy induction from limited examples," in *33rd ACM Intl. Conf. on Information and Knowledge Mgmt.*, 2024, pp. 3093–3102.

[34] M. B. Chaaben, L. Burgueño, and H. Sahraoui, "Towards using few-shot prompt learning for automating model completion," in *ICSE-NIER*. IEEE, 2023, pp. 7–12.

[35] R. Alur, L. D'Antoni, S. Gulwani, D. Kini, and M. Viswanathan, "Automated grading of DFA constructions," in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2013, pp. 1460–1466.

[36] E. W. Robson, S. Ruggerio, and J. Erickson, "FSM Builder: A tool for writing autograded finite automata questions," in *Proceedings of the 2024 Innovation and Technology in Computer Science Education (ITiCSE)*, 2024, pp. 269–275.

[37] W. Bian, O. Alam, and J. Kienzle, "Is automated grading of models effective? Assessing automated grading of class diagrams," in *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2020, pp. 365–376.

[38] K. Chen et al., "Embedding-based automated assessment of domain models," in *MODELS Companion '24: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, 2024, pp. 87–94.