GTA: Automated Semantic NLP-Based Generation of UML Activity Diagrams from Acceptance Criteria

Samia Nasiri[1]*, Salim Bloundi[2], Mohammed Lahmer[3]
ISIC Research Team, High School of Technology, Moulay Ismail University, Meknes, Morocco[1, 3]
Department of Mathematics and Computer Science, ENSAM, Moulay Ismail University, Meknes, Morocco[2]

***Abstract*—In agile development, user stories express stakeholder needs, and the associated acceptance criteria (AC), written in the Given/When/Then (GWT) notation, specify the behaviour expected of the system under stated preconditions. Their manual translation into UML activity diagrams is laborious and remains sensitive to wording differences between authors. Existing automated approaches rely on lexical matching or on grammatical rules. Both fail when an identical precondition is expressed through different wording, a case that occurs frequently in collaborative backlogs. This study introduces GTA (GWT-to-Activity), a deterministic pipeline that converts GWT AC into UML activity diagrams. The clause structure is first recovered through dependency parsing. Paraphrased *Given* states are then merged based on sentence embedding similarity using Sentence-BERT (SBERT), and natural language inference (NLI) is used to separate semantic opposition from lexical variation, with thresholds calibrated on annotated data to preserve output stability under paraphrastic variation. Eleven behavioural patterns are supported, from simple sequences to nested compound conditions. The pipeline was evaluated on 107 test sets containing 230 AC drawn from four heterogeneous corpora of synthetic, open-source, and benchmark origin, and it reaches a node F1-score of 99.26%, an edge F1-score of 97.37%, and an exact topological match rate of 95.33%. All eleven patterns are recovered, and every generated file is a valid PlantUML that requires no post-processing. The output is structurally correct, deterministic from one run to another, and traceable to the thresholds set at each gate.**

Keywords*—Behaviour-driven development; GWT acceptance criteria; UML activity diagrams; sentence-BERT; NLI; natural language processing; requirements engineering

I. Introduction

In agile projects, teams write requirements as user stories in natural language, and each story reflects a stakeholder need at a particular level of granularity. Every story is broken down into AC written in the Given-When-Then (GWT) template. This template comes from Behaviour-Driven Development (BDD) and is used in tools like Cucumber, JBehave, and SpecFlow [1] [2]. A GWT AC has three clauses conventionally defined. The Given clause sets the initial context of the scenario, namely the state of the system, the available data, and any prior conditions. The When clause identifies the event that takes place, whether triggered by the user, raised by another system, or fired by a timer. The Then clause reports the externally visible outcome of the event. The fact that a GWT AC is both executable and human-readable allows a single artefact to serve as a business requirement, an implementation specification, and an automated acceptance test.

*Corresponding author.

Producing a UML activity diagram from a set of GWT ACs is not a trivial task. The analyst has to identify shared preconditions, separate conditional branches from parallel actions, and rebuild the story's sequential flow, even when its wording has drifted across successive sprints. In most projects, this translation remains a manual task, which makes it time-consuming and prone to inconsistency. When it is not carried out or updated, a traceability gap grows between requirements and design artefacts [3].

Existing approaches to activity diagram generation rely either on syntactic heuristic rules applied to informal text [4], [5], [6], which lack semantic understanding and do not accept GWT structured input, or on LLM-based generation of process models from free-form process descriptions [7], which does not target structured AC and does not define a behavioural pattern taxonomy comparable to the one adopted in this work. None of these methods addresses the paraphrastic variability that arises when the same precondition is expressed differently across AC, a situation that is common in collaborative backlogs. This gap motivates the approach presented in this study.

Previous work [8] proposed an initial automated approach using Stanford CoreNLP for natural language preprocessing and Prolog rules for pattern extraction, generating PlantUML activity diagrams from GWT AC. That work demonstrated the feasibility of the task and achieved Recall of 97% and Precision of 94% on two banking-domain case studies. However, the evaluation exposed three limitations: decision-node detection relied on WordNet antonyms, failing for implicit negations and indirect oppositions; Given-state grouping assumed identical phrasing across AC, with no mechanism for phrasing variability; and the pattern taxonomy covered only three structural forms: sequence, fork, and binary decision.

This study addresses these limitations through a redesigned pipeline whose main contributions are:

- A hybrid semantic layer that combines syntactic dependency parsing with sentence embedding and natural language inference, in place of the rule-based semantic reasoning of the previous approach.
- A Given-state grouping mechanism whose outcome is stable across variations in authorship and wording.
- A behavioural pattern taxonomy covering eleven patterns, including loops, multi-branch decisions, conjunctive Then-clauses, asymmetric branches, independent parallel flows, and nested compound

conditions, each associated with a PlantUML generation rule.

- A systematic evaluation on 107 test sets covering 230 AC from 9 application domains, complemented by a phrasing-invariance benchmark and by an external validation on Cucumber feature files from public GitHub repositories.

The pipeline is evaluated on 107 test sets covering 230 AC drawn from four corpora and achieves a graph-level Node F1-score of 99.26%, an Edge F1-score of 97.37%, and an exact topological match rate of 95.3%.

The remainder of this study is organised as follows. Section II reviews related work. Section III presents background on the GWT format and PlantUML notation. Section IV describes the proposed pipeline. Section V reports the evaluation. Section VI discusses the results. Section VII concludes and outlines future work.

II. Related Work

A. UML Generation from Natural Language

NLP techniques have been applied to convert textual specifications into UML artefacts. Class diagrams are generated from textual requirements through linguistic analysis in [9] and from user stories in [10]. Earlier work [11] extracted use case, class, and package diagrams from user stories, then refined them through an ontological model. Sequence diagrams are derived from user stories in [12] by combining generative AI with rule-based methods. A survey by the authors of [13] reports that most approaches in this line of work remain semi-automatic, rely on handcrafted rules or predefined templates, and concentrate on structural rather than behavioural diagrams.

B. Activity Diagram Generation from Natural Language

The extraction of activity diagrams is more difficult than that of class diagrams. The control flow that these diagrams represent is sensitive to the way each behaviour is phrased, particularly for branching and iteration constructs, and this sensitivity increases when several behaviours are composed. Few approaches address this problem. In [14], activity diagrams are generated from Arabic user requirements through the MADA-TOKAN parser and heuristic rules over verbal and conditional sentence grammars. The method is not implemented on real cases, is language-specific, and requires strict input formatting. In [15], the SBVR2ACTIVITY tool maps SBVR specifications to activity diagrams through a POS tagger and transformation rules. Its applicability is bounded by the requirement that the input be written in SBVR notation. In [16], activity and use case diagrams are generated from informal natural language through sixteen syntactic reconstruction rules followed by Stanford CoreNLP analysis. Decisions are detected only when explicit conditional particles appear in the text; fork and join constructs are not handled, and the analysis remains at the part-of-speech level.

To the best of current knowledge, previous work [8] is the only reported approach that generates UML activity diagrams from GWT AC. It belongs to the family of rule-based text-to-model methods that use lexical resources for semantic reasoning and treat the input as a symbolic surface. Its pattern taxonomy, limited to three structural forms, is characteristic of first-generation pipelines. The three limitations of that work, listed in the introduction, are addressed by the semantic pipeline proposed in this study.

C. LLM-Based Approaches for Model Generation

The Transformer architecture [17], [18] has been applied to software engineering tasks that include UML generation, mostly on structural diagrams [12]. Few studies have applied it to activity diagram generation from GWT specifications. Sequence diagrams are generated from requirements through single-invocation prompting with GPT-3.5 and GPT-4 in [19], with an expert-based evaluation limited to completeness and correctness. A multi-step iterative pipeline for domain model extraction is proposed in [20]. The approach targets structural models and is evaluated by a single author. In [21], multiple candidate models produced by LLMs are aggregated into a probabilistic partial model and reduced to a single diagram through an optimisation solver; the evaluation is conducted on taxonomies and flowcharts rather than on activity diagrams derived from structured requirements. LADEX [22] targets activity diagram generation through a critique-and-refine loop that iteratively checks structural and alignment constraints. Fork and join are represented as ordinary action nodes, so parallel execution is not modelled. The best variant reaches 86% correctness and 92% completeness at an average cost of 4.91 LLM calls per generation. LADEX operates on free-form process descriptions rather than on structured AC and does not define an explicit pattern taxonomy. The evaluation used two datasets: 200 entries from the public PAGED dataset [23] and 20 industrial documents from Ciena, tested with three LLMs (GPT-4.1 Mini, O4 Mini, and DeepSeek-R1-Distill-Llama-70B).

D. UML Datasets and AI-Assisted Modelling

Datasets available for UML remain modest in size compared to those developed for NLP or computer vision [24]. An annotated image dataset for diagram recognition is released in [25], and ModelSet [26] provides a structural corpus of models intended for machine learning experiments in model-driven engineering. Neither dataset, however, contains behavioural diagrams. A different line of work is followed in [27], where thousands of UML diagrams from several diagram families are generated using LLMs and Vision-Language Models, then checked for syntactic and semantic correctness; the objective there is dataset synthesis, not requirement-driven generation. LLMs have also been used for other UML tasks, in particular the generation of UML diagrams from images [28] and the generation of method bodies from class diagrams [29].

E. Research Gap

Existing work concentrates on structural diagrams and leaves activity diagrams underrepresented. NLP-based methods for activity diagrams [14], [15], [16] rely on syntactic heuristics without a semantic component and are tied to specific input styles. LLM-based approaches [19], [22] handle semantic variability but operate on free-form text, define no explicit pattern taxonomy, and offer no reproducibility guarantee. No existing study addresses the automatic generation of activity

diagrams from Given/When/Then AC through a deterministic pipeline that combines syntactic and semantic analysis.

III. Background

A. The Given/When/Then Format in BDD

The GWT template splits an AC into three clauses: a Given clause that fixes a precondition on the system, a When clause that names the action triggering the behaviour, and a Then clause that gives the expected outcome. Cucumber [2] embeds this template in the Gherkin DSL, so that criteria written in Given-When-Then form become executable test scripts without further translation. A user story is usually specified by a set of such criteria produced collaboratively within the team.

B. Composition of AC Within a User Story

The AC of a single user story tends to follow a small number of composition patterns. Several criteria often share the same Given clause and differ only in their When/Then pairs, so each pair encodes one branch of a decision. Two criteria can also be linked in sequence, with the Then clause of the first providing the precondition of the second, or they can describe parallel flows within a single story, though this last case is rare in practice.

The formulation of the Given clause itself varies considerably. In production BDD files, the same underlying state is often written in different words from one iteration to the next. For example, “user is not authenticated”, “the user has not yet logged in”, and "no active session exists" all refer to the same precondition. The converse situation occurs just as often, since two clauses that look syntactically similar can in fact describe opposite states. The pipeline presented in this study is built to recognise equivalence in the first situation and to preserve the distinction in the second.

C. PlantUML Activity Diagram Constructs

PlantUML is a textual notation for describing UML diagrams, adopted in this work for the emission of activity diagrams. Each activity diagram construct corresponds to a specific PlantUML keyword or syntactic block. Table I lists the constructs used by the pipeline together with their PlantUML representation.

TABLE I. PlantUML Activity Diagram Constructs with Their Standard UML Notation

Construct	UML Notation	PlantUML Syntax	Description
Sequential Action	action	:action_label;	Rounded rectangle node
Conditional Branch	cond? [yes] [no]	if (cond) then (yes) ... else (no) ... endif	Diamond with true/false branches
Multi-branch Switch	switch	switch (subj) case (v1) ... endswitch	Diamond with 3 and guarded branches
Parallel Fork / Join		fork ... fork again ... end fork	Thick bars with concurrent flows
Loop	body	repeat :body; repeat while (cond?)	Backward-testing loop with exit condition
Flow Boundary	start stop	start/stop	Filled circle (start) and bull’s-eye (stop)

IV. Proposed Approach

A. Architecture Overview

This section presents GWT-to-Activity (GTA), a deterministic pipeline that converts GWT AC into syntactically valid UML activity diagrams expressed in the PlantUML notation. The overall architecture is organised in five sequential steps, shown in Fig. 1. Each step is described in the following subsections.

In the first step, action labels are extracted, and compound clauses are decomposed by dependency parsing. In the second step, criteria that share the same Given state are grouped by means of semantic similarity and natural language inference. In the third step, duplicate When clauses are identified within each group, and cross-group chains, in which the Then clause of one group corresponds to the Given clause of another, are detected. In the fourth step, each group is classified into one of eleven behavioural patterns. In the fifth step, the recognised pattern is translated into PlantUML syntax through dedicated template emitters.

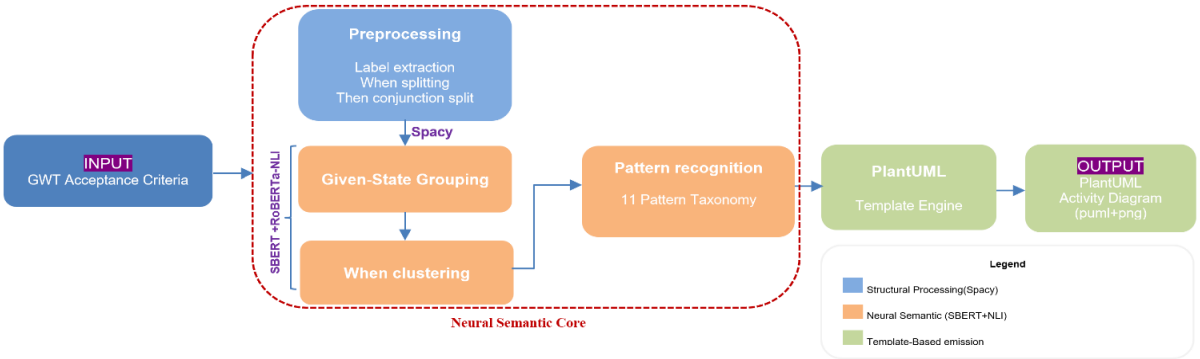


Fig. 1. Architecture of the proposed pipeline.

The pipeline is implemented in Python on top of three pre-trained NLP components: spaCy for part-of-speech tagging and dependency parsing, Sentence-BERT for semantic similarity computation, and RoBERTa for natural language inference, producing entailment and contradiction probabilities that drive both the grouping and decision-detection stages. Steps 1 and 5

are structural, handling tokenisation and text generation without any semantic inference; steps 2, 3, and 4 perform all grouping, clustering, and classification, each relying on these Transformer models.

B. Preprocessing

Each AC is preprocessed in three operations. First, then clauses are cleaned by removing determiners, auxiliaries, and pronouns, and labels exceeding a length limit are reduced to a skeleton via dependency parsing. Second, compound When clauses are handled at the preprocessing stage. A statement like "enters username and clicks login" contains two actions performed one after the other, so the pipeline separates them into two individual action nodes rather than treating them as a single event. Third, coordinating conjunctions in the Then clause receive a different treatment. When the outcome takes the form "A, B, and C", the three items are read as parallel effects rather than as a sequence, and the pipeline emits a fork construct with one outgoing branch for each item.

C. Three-Gate Given-State Grouping

Given-state grouping identifies the ACs that share a precondition state and must be rendered within the same diagram block. Merging clauses that describe different states, or splitting clauses that describe the same state, produces structurally invalid diagrams and accounts for most of the errors reported in rule-based approaches [13]. In this pipeline, each pair of Given clauses (a, b) is examined through three successive gates.

- Gate 1: SBERT Pre-filter (*threshold $\theta_1 = 0.45$*). The cosine similarity of the sentence embeddings of a and b is computed. If the score falls below a pre-filter threshold θ_1, the pair is rejected without invoking the Natural Language Inference (NLI) model. This gate is deliberately permissive: it eliminates clearly unrelated sentences while allowing lexically distant paraphrases to reach Gate 3.
- Gate 2: Near-identical shortcut (threshold θ_2= 0.95). Pairs scoring above a near-identical threshold θ_2 are accepted immediately. This handles the common case where two criteria carry the same Given clause verbatim or with minor token differences.
- Gate 3: NLI decisive analysis (decision threshold δ^* = 0.00). For pairs in the similarity range [0.45, 0.95], RoBERTa is invoked bidirectionally to obtain entailment scores (e_{ab}, e_{ba}) and contradiction scores (c_{ab}, c_{ba}). The net margin *S* is computed as:

$$S = (e_{ab} + e_{ba})/2 - \max(c_{ab}, c_{ba}) \quad (1)$$

A pair takes the label SAME_STATE if *S* is greater than δ^*, meaning that its two clauses correspond to the same precondition state; otherwise it takes the label DIFFERENT_STATE, and the two preconditions are counted as semantically distinct. The two entailment scores from the two inference directions are combined by their arithmetic mean, not by their minimum. Paraphrastic relations are frequently asymmetric, as illustrated by the pair "user is logged out" and "user has not yet authenticated": the entailment score is 0.97 in one direction and 0.002 in the reverse direction. The mean of the two directions yields a value of 0.45 for S, which assigns the pair to SAME_STATE in accordance with the gold annotation, whereas the minimum would produce −0.04 and separate the two clauses into distinct classes. Contradiction scores follow the inverse aggregation rule and are combined through the maximum of the two directions. Under this rule, a single direction with a strong contradiction score is sufficient to reject the pair, and clauses expressing opposed states are excluded from any common group. The threshold δ^* was calibrated through grid search over the interval [0.00, 0.60] with a step of 0.05, using a balanced gold standard of 96 pairs (48 SAME_STATE and 48 DIFFERENT_STATE); the outcome of this calibration is reported in Table II. Gates 1 and 2 use literature-informed thresholds ($\theta_1 = 0.45$, $\theta_2 = 0.95$) [30] and were not recalibrated on this dataset.

A high entailment score alone does not justify grouping two Given clauses. Two sentences with the same surface pattern can refer to different domain entities, in which case NLI models still return a moderate entailment score. The contradiction term guards against these FPs: if either inference direction yields a strong contradiction, *S* falls below δ^*, and the two clauses stay apart. Once each pair has been decided, the Given clauses are grouped into the connected components of the undirected graph associated with the three-gate criterion. The grouping is transitive by construction: if A matches B and B matches C, then A, B, and C fall in the same group, even if the pair (A, C) fails the acceptance threshold. Transitivity is intentional, since acceptance criteria written collaboratively often go through successive rewordings that the algorithm needs to reconcile into a single state. The resulting partition is also independent of input order, because connected components are fully determined by the edge set of the graph, not by the traversal sequence. This invariance was verified empirically by running each test set through the pipeline 100 times with the AC shuffled at each run; the partitions produced were identical across all runs.

TABLE II. GATE 3 THRESHOLD CALIBRATION: SETUP AND OPTIMAL OUTCOME ON THE 96-PAIR GOLD STANDARD DATASET

Parameter	Value
Dataset size	96 pairs (48 SAME_STATE and 48 DIFFERENT_STATE)
Search range	$\delta^* \in [0.00, 0.60]$, step = 0.05
Optimal δ^*	0.00
F1 score	97.96%
FP	2

D. When-Cluster Analysis and Decision Scoring

Within each Given group, every pair (a, b) of *When* clauses is examined to decide whether the two clauses correspond to alternative branches of a decision or to independent concurrent actions. The former requires a decision diamond, the latter a fork/join construct. This decision is made by the decision detection procedure, which applies a score gate defined by the following scoring function:

$$\text{score}(a, b) = C(a, b) + \alpha \times \text{Sim}(a, b) \quad (2)$$

where, C(a, b) is the RoBERTa contradiction probability for the pair and Sim(a, b) is the SBERT cosine similarity. The two

terms are complementary: contradiction captures semantic opposition between mutually exclusive branch conditions, while similarity confirms that both clauses operate in the same action domain.

The weight α and the classification threshold τ were calibrated by grid search over α ∈ [0.0, 3.0] in steps of 0.1 and τ ∈ [0.00, 3.00] in steps of 0.05, on a manually annotated gold set of 88 When-clause pairs, of which 77 are labelled DECISION and 11 are labelled FORK. The calibration selects α = 1.1 and τ = 0.50. A pair (a, b) is then classified as DECISION when score(a, b) ≥ τ and at least one marker of Table III applies. Comparison threshold patterns such as "more than N" or "at least K" form an exception: they trigger the DECISION classification alone, without passing through the score gate. Table III lists the four linguistic markers that confirm a DECISION classification, and Table IV reports the calibration results measured on the 88 When-clause pairs.

TABLE III. LINGUISTIC MARKERS FOR DECISION CLASSIFICATION

Marker	Detection
NLI contradiction	contradiction score ≥ 0.90 with an identical grammatical subject
Negation	spaCy dependency parsing combined with NLI zero-shot prototype matching against positive and negative reference sentences
Variant specialization	one *When* clause refines the other
Shared finite verb	The same finite verb occurs in both clauses

TABLE IV. DECISION SCORING PERFORMANCE ON THE 88-PAIR WHEN-CLAUSE GOLD SET

Metric	Value
F1	95.06%
Precision	90.60%
Recall	100%
FN	0
FP	8
Dataset	88 pairs (77 DECISION / 11 FORK)

Across all 88 gold set pairs, the calibrated function achieves a recall of 100% and produces no FN; therefore, no DECISION branch is classified as a FORK. This zero-FN calibration yields 8 FP and an accuracy of 90.6%.

E. Pattern Recognition and Structural Pattern Taxonomy

Through systematic analysis of the three evaluation corpora, eleven structural patterns (P1 to P11) were identified, which together cover the principal behavioural specifications. Table V summarises the patterns and their detection conditions.

The eleven patterns are organised into three behavioural families. The linear flow family groups the branchless patterns, that is P1, P2, and P8. Under P1, a single AC is expressed as a linear action sequence. P2 covers the case in which several ACs are chained together: the Then clause of one criterion is semantically equivalent to the Given clause of the next, and the pipeline exploits this equivalence to remove the intermediate state node from the resulting diagram. P8 is an asymmetric form of the chain, in which the binary decision contained inside the pattern has one branch that continues downstream while the other branch is a dead end and is closed by an explicit stop node.

TABLE V. STRUCTURAL PATTERN TAXONOMY WITH DETECTION SIGNALS

ID	Pattern	Structure	Detection Signal
P1	Simple Sequence	Linear chain	Single when clustered in a group
P2	Chain	Then-to-Given link	SBERT/NLI cross-group match
P3a	Decision:Opposition	if/else	NLI contradiction and negation markers
P3b	Decision: Variant	if/else	One When subsumes the other
P3c	Decision:Comparative	if/else	Comparison operators in When or Then
P4	Multi-branch Switch	switch/case	Three or more distinct When clauses sharing a common grammatical subject
P5	Fork / Join	Parallel independent actions	No semantic opposition or shared verb between When clauses
P6	Loop	repeat/while	SBERT(Then, Given) ≥ 0.85
P7	Conjunction in Then	fork from A, B, and C	Multiple verb clauses and parallelism keyword in Then
P8	Dead-end Branch	Asymmetric chain	Only one decision branch links to a subsequent Given
P9	Independent Flows	Separate start/stop	Disjoint Given groups with no chain link
P10	Arithmetic Operators	Numeric guard labels	Numeric comparison expression (>, ≥, <, ≤) in clause
P11	Nested Decision	Compound condition	AND conjunction between two stative conditions in When

The conditional branching family regroups the constructs built around a decision diamond, and it contains P3a, P3b, P3c, P4, and P11. Binary decisions occur in three forms, one per type of semantic relation between the two When clauses. P3a covers the case of pure opposition and yields if/else blocks with yes/no labels. P3b covers variant specialisation, in which one When clause refines the other, and it uses a descriptive else label rather than a generic one.P3c captures quantitative thresholds through a numeric comparison operator. P4 generalises the same idea to three or more mutually exclusive alternatives and emits a switch/case construct. P11 handles compound stative conditions, which are When clauses joining two independent sub-conditions, and renders them as nested if/endif blocks.

The remaining patterns cover concurrent execution, iteration, and structural isolation. P5 produces a fork/join construct when several When clauses within the same Given group correspond to semantically independent triggers running in parallel. P7 produces the same construct for a different reason: when a single Then clause lists several simultaneous outcomes. P6 captures loop semantics: the pattern is triggered when the Then clause re-establishes its own Given precondition, and the emitter produces a repeat/while structure. P9 applies when two or more Given groups have no semantic link between them; each group is then rendered as an independent flow. P10 is not a standalone pattern, but a label-level specialization applied to any

decision construct whose When clause contains a numeric comparison expression.

F. PlantUML Emission

Each recognised pattern is translated into syntactically valid PlantUML activity diagram notation by a template-based emitter. The emitter instantiates a fixed skeleton per pattern, substituting extracted labels into predefined slots.

Three emission rules apply to every branching pattern. First, branch ordering is deterministic: the affirmative (or non-negated) When clause always occupies the then (yes) branch, so diagram orientation does not depend on the order of the AC in the input. Second, the else label is chosen differently for P3a and P3b. If the minimum bilateral NLI contradiction score exceeds the decision threshold, the branch takes the generic (no) label; otherwise, the label reuses the descriptive text of the second When clause, since the two clauses describe variant specialisations of the same intent rather than opposite outcomes. Third, topological ordering is enforced between groups: when a group is the target of a chain link, we render it inline through a recursive depth-first traversal, so the sequential flow reads from top to bottom in the generated code. Every opened control structure is explicitly closed by the emitter.

V. Evaluation

A. Experimental Setup

The pipeline is implemented in Python 3.10. Dependency parsing uses spaCy with the en_core_web_sm model. Sentence similarity relies on Sentence-BERT (all-MiniLM-L6-v2, 80 MB). Natural language inference uses RoBERTa-large fine-tuned on MultiNLI (1.4 GB) [26]. All experiments were run on a standard laptop (Intel i7, 16 GB RAM, CPU only).

B. Dataset Construction

The proposed approach is evaluated on a dataset of 230 AC distributed across 107 AC sets drawn from four independent sources, summarised in Table VI. No existing public corpus covers the full range of target structural patterns: available Gherkin repositories on GitHub are dominated by simple sequences and basic decisions, with rare or no instances of patterns such as multi-branch switch (P4), Then-clause conjunction (P7), dead-end branches (P8), or nested decisions (P11). The dataset was therefore constructed incrementally from four complementary sources.

The Controlled Benchmark covers all eleven structural patterns P1 to P11 and includes seven sets drawn from the baseline [8], along with two adversarial robustness series: RB1, targeting paraphrase equivalence where semantically identical Given clauses are expressed with different surface forms, and RB2, targeting false similarity where lexically close Given clauses denote distinct states. The Real GitHub partition provides eight AC sets extracted verbatim from public open-source repositories [31], [32], [33], [34] with no modification, serving as an uncontrolled reference. After an initial evaluation on these two partitions, the need to strengthen the statistical reliability of per-pattern metrics for the most frequent patterns (P1, P2, P3a) motivated the addition of a third partition, the Synthetic DS, which increases the number of AC sets covering these common patterns. A fourth partition, the Paraphrase benchmark, was then added to specifically evaluate the pipeline robustness to paraphrased Given clauses and Then-to-Given transitions across consecutive AC, covering patterns P3a, P4, P5, P6, and P11 with semantically equivalent but lexically varied formulations.

TABLE VI. Benchmark Dataset Composition by Pattern

Dataset	Sub-group	Sets	AC	Patterns Covered
Controlled Benchmark	baseline [8]	7	14	P1, P3a, P3a+P6, P3b, P9
	Core patterns	36	85	P1–P11 (all eleven patterns)
	Robustness (RB1/RB2)	16	33	RB1, RB2 with P3a, P3c, P6, P7, P11
	Sub-total	**59**	**132**	
Real GitHub	—	8	20	P1, P3a, P4, P9
Synthetic DS	—	20	28	P1, P2, P3a
Paraphrase Benchmark	—	20	50	P3a, P4, P5, P6, P11
Total		**107**	**230**	

All sets in the Controlled Benchmark (except the baseline sets), in the Synthetic DS partition, and in the Paraphrase benchmark were initially drafted using a large language model, then manually reviewed, corrected, and validated by the authors. Each AC set was verified against two criteria: syntactic well-formedness in standard GWT notation and exact correspondence with the structural pattern assigned in the ground truth.

C. Evaluation Methodology

Two keyword lists in PlantUML syntax define the ground truth for each pattern. The first list holds the keywords that a correct diagram must contain; the second, the keywords that must not appear. Any match on the first list is scored as a True Positive (TP), and the absence of every keyword on that list is scored as a False Negative (FN). Symmetrically, any match on the second list is scored as a False Positive (FP), and the absence of every keyword on that list as a True Negative (TN). Detection is performed by substring matching on the PlantUML source, and Precision, Recall, F1, and Accuracy are then obtained from the four resulting counts. Aggregate values follow a micro-averaged scheme: the counts are pooled across all AC sets before the metrics are computed, so a set with more markers weighs proportionally more in the final scores. Table VII reports the overall classification results on the four datasets.

On the full 107-set benchmark, the pipeline reaches an F1 score of 98.57%, with 275 TPs, 2 FNs, 6 FPs, and a specificity of 97.6%. A True Negative means that a PlantUML construct that should not appear in the diagram is indeed absent from the generated output. In a simple sequence pattern (P1), for example, decision, fork, and loop constructs must all stay out of the output, and each construct correctly kept out adds one to the TN count. Both FNs come from the Baseline set, the only test case in the evaluation that combines several patterns within a single specification. In this case, the loop sub-structure is missing from the generated diagram because the sequential priority tree stops as soon as the decision construct is recognised,

before reaching the loop-detection step. Three FPs result from Given-state fork ambiguity, where asymmetric NLI entailment between consecutive preconditions causes the three-gate grouping to merge distinct states, leading the pattern recogniser to emit a fork construct. Two FPs arise from decision-boundary ambiguity, where opposed When clauses within a sequential pattern trigger the decision detection procedure. One FP is caused by loop-boundary ambiguity, where high SBERT similarity between a Then clause and its own Given state crosses the loop detection threshold. No FP was observed on the Real GitHub or Paraphrase DS partitions, indicating that these boundary cases are specific to controlled test formulations rather than real-world GWT specifications.

To complement this keyword-based evaluation, which cannot detect inverted branches or incorrectly connected transitions, each generated PlantUML diagram was parsed into a typed directed graph with node types START/END/ACTION/DECISION/SWITCH/FORK/JOIN/LOOP_START and edges carrying yes/no/case-value guards. An edge is counted as correct only when both of its endpoints and its condition match those of the reference graph. Across the 107 benchmark sets, this graph-level comparison yields a node F1 of 99.26% and an edge F1 of 97.37%, and 102 of the 107 generated diagrams (95.33%) are topologically exact matches to their references. Precision, Recall, F1, and Accuracy, together with the corresponding TP, FP, and FN values, are given per structural element in Table VIII.

The point estimates for Precision, Recall, F1, and Accuracy do not convey their sampling variability. To quantify this uncertainty, 95% confidence intervals (CIs) were computed using a non-parametric bootstrap [35] with B = 10,000 resamples. This procedure was performed at the level of test sets, not at the level of individual ACs, because ACs within a single set are not independent. The results are presented together in a single diagram. Resampling at the criterion level would therefore break this dependence and produce artificially narrow intervals. Table IX reports the point estimates of the structural metrics along with their 95% bootstrap intervals.

To illustrate the pipeline output, Table X presents representative generated activity diagrams alongside the corresponding AC sets and the detected pattern types. The selection covers five patterns spanning the four behavioural families of the taxonomy: P3a (binary decision), P4 (multi-branch switch), P6 (loop), P8 (dead-end branch), and P11 (nested decision).

TABLE VII. KEYWORD-LEVEL CLASSIFICATION PERFORMANCE ACROSS EVALUATION DATASETS

Dataset	Sets	AC	TP	FN	FP	TN	Precision	Recall	F1	Accuracy
Controlled Benchmark	59	132	146	2	2	108	98.65%	98.65%	98.65%	98.45%
Real GitHub	8	20	18	0	0	15	100%	100%	100%	100%
Synthetic DS	20	28	46	0	4	73	92%	100%	95.83%	96.75%
Paraphrase DS	20	50	65	0	0	49	100%	100%	100%	100%

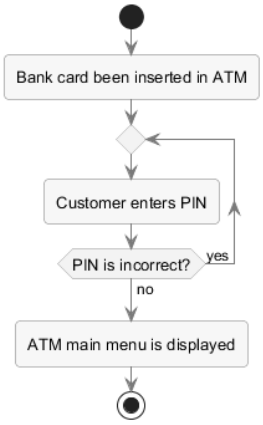
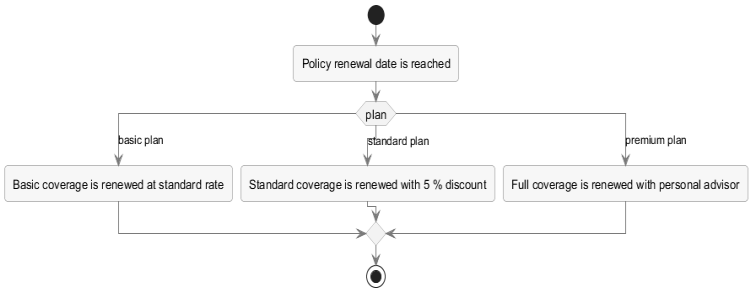
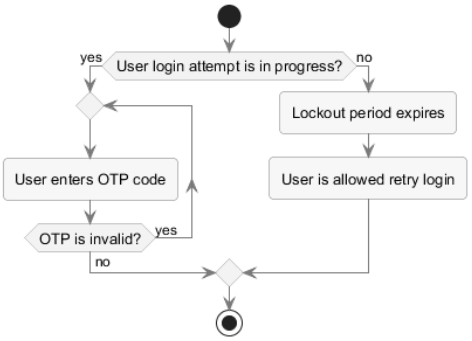
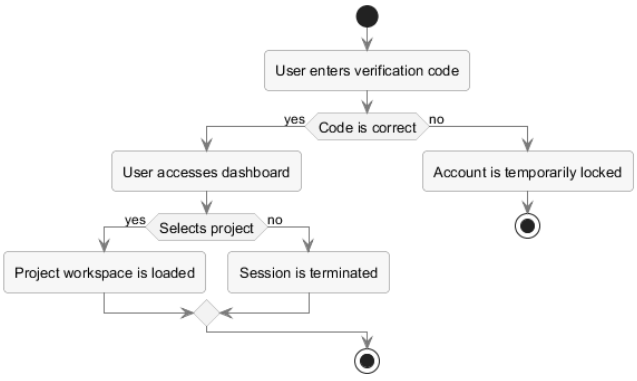
TABLE VIII. STRUCTURAL MATCHING PERFORMANCE

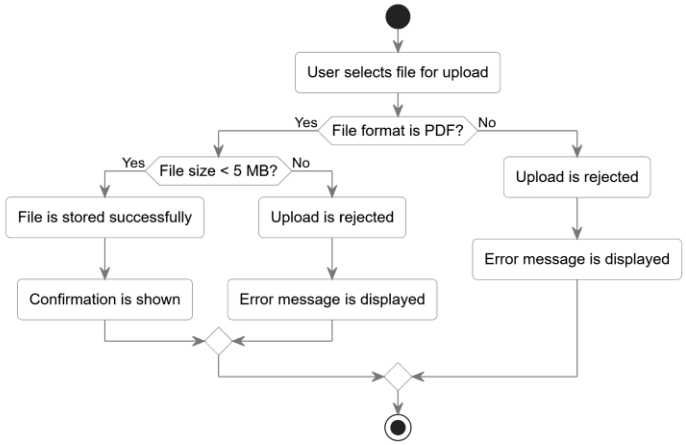
Structural element	TP	FP	FN	Precision	Recall	F1	Accuracy
START	107	1	0	99.07%	100%	99.53%	99.07%
END	109	1	0	99.09%	100%	99.54%	99.09%
ACTION	457	1	0	99.78%	100%	99.89%	99.78%
DECISION	89	1	1	98.89%	98.89%	98.89%	97.80%
SWITCH	10	0	0	100%	100%	100%	100%
FORK	15	3	0	83.33%	100%	90.91%	83.33%
JOIN	15	3	0	83.33%	100%	90.91%	83.33%
LOOP_START	7	1	0	87.50%	100%	93.33%	87.50%
TOTAL (Node)	**809**	**11**	**1**	**98.66%**	**99.88%**	**99.26%**	**98.54%**
TOTAL (Edge)	**815**	**28**	**16**	**96.68%**	**98.07%**	**97.37%**	**94.88%**

TABLE IX. EXACT TOPOLOGICAL MATCH RATE WITH 95% BOOTSTRAP CONFIDENCE INTERVAL

Metric	Point estimate	95% CI (bootstrap, n=107, B=10,000)
Node F1	99.26%	[98.56%, 99.82%]
Edge F1	97.37%	[94.85%, 99.46%]
Exact-match rate	95.33%	[90.65%, 99.07%]

TABLE X. GENERATED ACTIVITY DIAGRAMS WITH THE CORRESPONDING ACCEPTANCE CRITERIA SETS AND DETECTED PATTERN TYPES

Pattern	Set AC	Generated Activity Diagram
P3a	**AC 1:** **Given**: A bank card has been inserted in the ATM. **When**: Customer enters the PIN, and the PIN is correct **Then**: The ATM main menu is displayed **AC 2:** **Given**: A bank card is present in the ATM card reader **When**: Customer enters the PIN, and the PIN is incorrect **Then**: Customer enters the PIN	
P4	**AC 1:** **Given:** Policy renewal date is reached **When:** Customer selects basic plan **Then:** Basic coverage is renewed at the standard rate **AC 2:** **Given:** Policy renewal date is reached **When:** Customer selects standard plan **Then:** Standard coverage is renewed with a 5% discount **AC 3:** **Given:** Policy renewal date is reached **When:** Customer selects premium plan **Then:** Full coverage is renewed with a personal advisor	
P6	**AC 1:** **Given:** User login attempt is in progress **When:** user enters the OTP code, and the OTP is invalid **Then: the** user is prompted to enter the OTP code **AC 2:** **Given:** User account is locked due to failed attempts **When:** lockout period expires **Then:** user is allowed to retry login	
P8	**AC 1:** **Given:** User enters the verification code **When:** Code is correct **Then:** User accesses the dashboard **AC 2:** **Given:** User enters the verification code **When:** Code is incorrect **Then:** Account is temporarily locked **AC 3:** **Given:** User accesses the dashboard **When:** User selects a project **Then:** Project workspace is loaded **AC 4:** **Given:** User accesses the dashboard **When:** User logs out **Then:** Session is terminated	

P11	**AC 1:** **Given**: User has selected a file for upload **When**: The file format is PDF, and the file size is under 5 MB **Then**: the file is stored successfully, and confirmation is shown **AC 2:** **Given**: A file has been chosen by the user for upload **When**: file format is not PDF or file size exceeds 5 MB **Then**: upload is rejected, and an error message is displayed	

VI. Discussion

A. Overall Performance

The proposed GTA pipeline achieves a global F1-score of 98.57% across all 107 test sets covering 230 AC. The recall reached by the pipeline (99.28%) is particularly important because a missed pattern feature (FN) produces a structurally incorrect diagram, whereas a spurious feature (FP) is generally less harmful. This result is supported by the calibrated decision scoring function, which achieves zero FN on the 88-pair When gold set, ensuring high recall at the pattern classification level.

Performance is consistent across the four dataset partitions: the Controlled Benchmark achieves an F1 score of 98.65%, the Real GitHub partition reaches 100%, the Synthetic DS 95.83%, and the SBERT Paraphrase benchmark 100%. The absence of a significant gap between the controlled and real-world partitions indicates that the approach generalizes beyond controlled benchmark settings. The Real GitHub partition exhibits inconsistent formatting, abbreviated vocabulary, and multi-author linguistic variation, conditions known to challenge rule-based and embedding-based systems, yet all eight sets are classified without error.

The graph-level structural evaluation applies a more demanding test than the keyword-level analysis on branch-assignment correctness. On the 107-set benchmark, DECISION nodes, which correspond most directly to branch assignment, are recovered with an F1 of 98.89%. The residual structural errors are all located in one confusion, between fork and sequential-chain constructs (FORK/JOIN F1 of 90.91% on 3 of the 107 sets), and are not uniformly distributed across the benchmark. This pattern suggests a localised limitation and not a systemic one. A dedicated gate to separate sequential chains from parallel forks is left for future work. In addition, 95.33% of the generated diagrams are topologically identical to their references, which establishes that the pipeline output is correct at the structural level, and not merely at the keyword level. The bootstrap confidence intervals for each partition are given in Table X. Their width depends on the number of test sets in the partition and should be read together with that number. The Real GitHub partition contains only 8 sets, which makes it the smallest of the four, and its intervals are consequently the widest. The maximum scores observed on this partition should therefore be interpreted with caution.

B. Comparison with the Baseline Reference Diagrams

In previous work [8], the authors report 47 TP, 1 FN, and 2 FP on a 7-set corpus, for Precision, Recall, and F1 of 95.92%, 97.92%, and 96.91%, respectively. A direct comparison with the GTA figures would be misleading, since the two evaluations operate at different granularities. The earlier framework counts each node and each edge of the diagram individually, while GTA checks whether specific PlantUML constructs are present in the output, namely if/else, fork/join, repeat/while, and switch/case. Despite this difference, a qualitative comparison on the shared baseline sets highlights three limitations of the prior approach that motivated the redesign.

The prior approach [8] groups Given clauses through exact token matching. Two developers writing the same precondition in slightly different words result in two distinct states, and the resulting diagram is split into unrelated fragments. In the proposed pipeline, SBERT similarity followed by RoBERTa NLI handles this case, and different wordings of the same precondition are grouped together without any lexical overlap. Decision detection in the earlier work [8] exhibits a comparable limitation. The reliance on WordNet antonymy and explicit negation covers only the oppositions expressed through common antonym pairs. In the present study, the pipeline integrates a decision scoring function based on contradiction and similarity signals with a set of complementary linguistic markers, and extends coverage to variant decisions (P3b), comparative decisions (P3c), and nested compound conditions (P11) beyond the binary opposition cases handled by lexical antonymy. Chain detection presents the same type of limitation. In the older algorithm, two ACs are linked only if the Then clause of the first is textually identical to the Given clause of the second. The pipeline replaces this strict textual match with an SBERT similarity computed between groups, and it turns to NLI entailment as a second stage whenever the similarity score alone is not sufficient to reach a decision.

The Cash Withdrawal set from [8] illustrates the practical impact. Its two Given clauses “account is in credit” and “account is overdrawn” refer to opposite states, but no entry in WordNet connects the words “credit” and “overdrawn”. The older

approach misses the opposition entirely and generates a fork-based structure in which the Then actions from both preconditions are executed on concurrent paths, rather than being placed in the two alternative branches of a decision. The NLI-based score used in the proposed pipeline recognizes the contradiction and outputs the decision diamond required by the AC, as shown in Fig. 2.

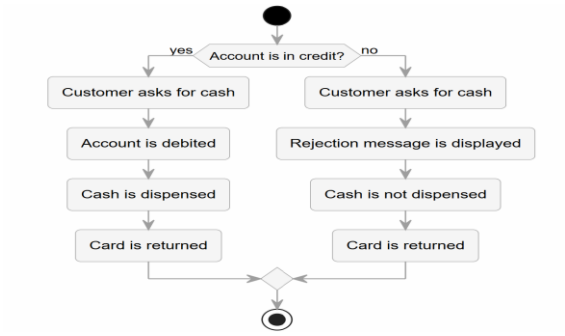


Fig. 2. Activity diagrams generated for the cash withdrawal specification.

On the seven shared baseline sets [8], the current pipeline achieves a precision of 100%, Recall of 90.48%, and F1 of 95%. The 2 FNs are confined to the ATM Transactions set, the only combined pattern (decision embedded in a loop), which the prior approach handles through direct syntactic rules but which is not addressed by the current pipeline. This trade-off is accepted for two reasons. First, the current pipeline covers eleven structural patterns compared to five in the prior approach [8]. Second, it generalises to paraphrased and linguistically varied inputs that the rule-based system cannot resolve.

C. NLI Selection

RoBERTa-large-MNLI was selected as the NLI component in Gate 3 after a systematic comparison with three alternative models on the full 107-set benchmark. All models were run under the same experimental configuration, with δ^* set to 0.00 and α set to 1.1, and no per-model threshold recalibration was applied. Table XI shows the results of the keyword-matching evaluation.

TABLE XI. KEYWORD-LEVEL NLI MODEL COMPARISON ON THE 107-SET BENCHMARK

Model	Param	Precision	Recall	F1	Time (s)
RoBERTa-large-MNLI	**355M**	**97.86%**	**99.28%**	**98.57%**	**712**
DeBERTa-v3-large	435M	97.52%	98.92%	98.21%	2005
DistilBERT-MNLI	66M	97.41%	94.95%	96.16%	145
BERT-base-MNLI	110M	90.59%	55.60%	68.90%	186

RoBERTa-large-MNLI achieves the best overall performance, with an F1-score of 98.57% and a Recall of 99.28%, ahead of DeBERTa-v3-large by 0.36 F1 points while running 2.8 times faster. BERT-base-MNLI cannot be retained for this task: its Recall drops to 55.60%, meaning that roughly 44% of paraphrased Given states go undetected. DistilBERT-MNLI, the fastest of the four at 145 seconds, reaches a Recall of only 94.95%; the resulting 14 additional FNs affect 12 AC, whose generated diagrams are left structurally incomplete. Because a single missing construct invalidates the diagram, this Recall loss cannot be traded against the computational saving.

The four backbones were then re-evaluated under the graph-level structural comparison introduced earlier in this study, which imposes a stricter criterion on branch assignment and control-flow correctness. The scores obtained by the four backbones on node F1, edge F1, and the exact-match rate are gathered in Table XII. RoBERTa-large-MNLI achieves the highest scores on the three metrics, which motivated its selection as the backbone of the pipeline. DeBERTa-v3-large trails it throughout, despite its larger size, and also runs substantially slower (1809s versus roughly 1249s), making it a dominated choice. DistilBERT-MNLI, with roughly a fifth of the parameter count, stays close behind (Node F1 = 96.15%) at a fraction of the compute cost, and offers a reasonable lightweight alternative when the compute budget is tight. BERT-base-MNLI trails all three other models, especially on edges (Edge F1 = 53.63%); its contradiction and entailment signals appear too noisy to support the branch-detection and Given-state grouping decisions on which the pipeline relies. RoBERTa-large-MNLI is also the default zero-shot classification model in the Transformers library [36], which keeps versioning and deployment reproducible.

TABLE XII. STRUCTURAL NLI MODEL COMPARISON ON THE 107-SET BENCHMARK

Model	Param	Node F1	Edge F1	Exact-match	Time (s)
RoBERTa-large-MNLI	**355M**	**99.26%**	**97.37%**	**95.33%**	**1249**
DeBERTa-v3-large	435M	97.30%	92.26%	89.72%	1809
DistilBERT-MNLI	66M	96.15%	89.75%	85.05%	172
BERT-base-MNLI	110M	78.29%	53.63%	41.12%	213

VII. CONCLUSION

This study has described an automated pipeline that turns GWT acceptance criteria into UML activity diagrams. The pipeline combines syntactic parsing with Transformer-based semantic similarity and natural language inference, and it is organised around a three-gate Given-state grouping mechanism together with a multi-signal decision scoring function. Eleven structural patterns are supported, ranging from simple sequences and binary decisions to multi-branch switches, fork/join constructs, loops, nested compound conditions, and independent flows.

On a benchmark of 230 AC distributed over 107 test sets drawn from four independent sources, the pipeline reaches a node F1 of 99.26% and an edge F1 of 97.37%; the latter measures the correctness of control-flow connections and branch guards. 95.33% of the generated diagrams are topologically identical to their reference, and this figure remains stable across the four partitions of the benchmark. The pipeline also covers configurations on which the earlier rule-based baseline fails, in particular paraphrased preconditions, variant decisions, and nested compound conditions that fall outside what string matching and lexical antonymy can capture.

Future work includes three directions. First, a dedicated gate that separates sequential chains from parallel forks would resolve the residual FORK/JOIN confusion identified in the structural evaluation. Second, an evaluation on larger industrial BDD repositories would establish how the pipeline performs at that scale. Third, a hybrid pipeline in which the structural output of the current approach is used as a reference for LLM-based PlantUML generation would allow LLM-generated diagrams to be validated and hallucinated constructs to be corrected.

REFERENCES

[1] A. Solis, "ULRR A study of the characteristics of behaviour driven development Item Type Meetings and Proceedings," 2011. [Online]. Available: https://hdl.handle.net/10344/1256

[2] A. Hellesoy, S. Tooke, and M. Wynne, "The cucumber book: behaviour-driven development for testers and developers," 2017.

[3] O. C. Z. Gotel and C. W. Finkelstein, "An analysis of the requirements traceability problem," in Proceedings of IEEE international conference on requirements engineering, IEEE, 1994, pp. 94–101.

[4] J. Fischbach, A. Vogelsang, D. Spies, A. Wehrle, M. Junker, and D. Freudenstein, "SPECMATE: Automated Creation of Test Cases from AC," in Proceedings - 2020 IEEE 13th International Conference on Software Testing, Verification and Validation, ICST 2020, Institute of Electrical and Electronics Engineers Inc., Oct. 2020, pp. 321–331. doi: 10.1109/ICST46399.2020.00040.

[5] P. Pandit and S. Tahiliani, "AgileUAT: A Framework for User Acceptance Testing based on User Stories and AC," 2015.

[6] Y. Wautelet, S. Heng, M. Kolp, and I. Mirbel, "Unifying and extending user story models," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 8484 LNCS, pp. 211–225, 2014, doi: 10.1007/978-3-319-07881-6_15.

[7] N. Klievtsova, J.-V. Benzin, T. Kampik, J. Mangler, and S. Rinderle-Ma, "Conversational process modeling: Can generative ai empower domain experts in creating and redesigning process models?," arXiv preprint arXiv:2304.11065, 2023.

[8] S. Nasiri, A. Adadi, and M. Lahmer, "Automatic generation of business process models from user stories," International Journal of Electrical and Computer Engineering, vol. 13, no. 1, pp. 809–822, Feb. 2023, doi: 10.11591/ijece.v13i1.pp809-822.

[9] E. A. Abdelnabi, A. M. Maatuk, T. M. Abdelaziz, and S. M. Elakeili, "Generating UML Class Diagram using NLP Techniques and Heuristic Rules," in Proceedings - STA 2020: 2020 20th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering, Institute of Electrical and Electronics Engineers Inc., Dec. 2020, pp. 277–282. doi: 10.1109/STA50679.2020.9329301.

[10] S. Nasiri, Y. Rhazali, M. Lahmer, and N. Chenfour, "Towards a Generation of Class Diagram from User Stories in Agile Methods," Procedia Comput. Sci., vol. 170, pp. 831–837, 2020, doi: 10.1016/j.procs.2020.03.148.

[11] S. Nasiri, Y. Rhazali, M. Lahmer, and A. Adadi, "From User Stories to UML Diagrams Driven by Ontological and Production Model." [Online]. Available: www.ijacsa.thesai.org

[12] 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 Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, 2024, pp. 138–148.

[13] S. Ahmed, A. Ahmed, and N. U. Eisty, "Automatic transformation of natural to unified modeling language: A systematic review," in 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA), IEEE, 2022, pp. 112–119.

[14] I. N. Nassar and F. T. Khamayseh, "Constructing activity diagrams from Arabic user requirements using Natural Language Processing tool," in 2015 6th International Conference on Information and Communication Systems (ICICS), IEEE, 2015, pp. 50–54.

[15] U. Iqbal and I. S. Bajwa, "Generating UML activity diagram from SBVR rules," in 2016 Sixth International Conference on Innovative Computing Technology (INTECH), IEEE, 2016, pp. 216–219.

[16] A. M. Maatuk and E. A. Abdelnabi, "Generating UML use case and activity diagrams using NLP techniques and heuristics rules," in ACM International Conference Proceeding Series, Association for Computing Machinery, Apr. 2021, pp. 271–277. doi: 10.1145/3460620.3460768.

[17] A. Vaswani et al., "Attention is all you need," Adv. Neural Inf. Process. Syst., vol. 30, 2017.

[18] A. Vogelsang and J. Fischbach, "Using large language models for natural language processing tasks in requirements engineering: A systematic guideline," in Handbook on Natural Language Processing for Requirements Engineering, Springer, 2025, pp. 435–456.

[19] A. Ferrari, S. Abualhaija, and C. Arora, "Model generation with LLMs: From requirements to UML sequence diagrams," in 2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW), IEEE, 2024, pp. 291–300.

[20] Y. Yang, B. Chen, K. Chen, G. Mussbacher, and D. Varró, "Multi-step iterative automated domain modeling with large language models," in Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, 2024, pp. 587–595.

[21] B. Chen, "Consistent Graph Model Generation with Large Language Models," in 2025 IEEE/ACM 47th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), IEEE, 2025, pp. 218–219.

[22] P. Khamsepour et al., "The Impact of Critique on LLM-Based Model Generation from Natural Language: The Case of Activity Diagrams," arXiv preprint arXiv:2509.03463, 2025.

[23] W. Du, W. Liao, H. Liang, and W. Lei, "PAGED: A benchmark for procedural graphs extraction from documents," in Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long papers), 2024, pp. 10829–10846.

[24] J. Whittle, J. Hutchinson, and M. Rouncefield, "The state of practice in model-driven engineering," IEEE Softw., vol. 31, no. 3, pp. 79–85, 2013.

[25] J. Torcal, V. Moreno, J. Llorens, and A. Granados, "Creating and validating a ground truth dataset of unified modeling language diagrams using deep learning techniques," Applied Sciences, vol. 14, no. 23, p. 10873, 2024.

[26] J. A. H. López, J. L. Canovas Izquierdo, and J. S. Cuadrado, "ModelSet: a dataset for machine learning in model-driven engineering," Softw. Syst. Model., vol. 21, no. 3, pp. 967–986, 2022.

[27] N. Van-Viet, H.-K. Nguyen, N. Kim-Son, T. M.-H. Luong, D.-Q. Vu, and T.-N. Phung, "A novel unified framework for automated generation and multimodal validation of UML diagrams," Computer Modeling in Engineering & Sciences, vol. 146, no. 1, 2026.

[28] A. Conrardy and J. Cabot, "From image to UML: First results of image based UML diagram generation using LLMs," arXiv preprint arXiv:2404.11376, 2024.

[29] D. Rouabhia and I. Hadjadj, "Behavioral Augmentation of UML Class Diagrams: An Empirical Study of Large Language Models for Method Generation," arXiv preprint arXiv:2506.00788, 2025.

[30] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," Aug. 2019, [Online]. Available: http://arxiv.org/abs/1908.10084

[31] Mamidisrisaiteja, "AIPythonPlaywrightE2E27July25_02," GitHub, 2025.[Online].Available: https://github.com/mamidisrisaiteja/AIPythonPlaywrightE2E27July25_02

[32] Vitorebatista, "Keepfy," GitHub. [Online]. Available: https://github.com/vitorebatista/Keepfy

[33] Ivanwidyan, "serenity-cucumber4-sample-project," GitHub. [Online]. Available: https://github.com/ivanwidyan/serenity-cucumber4-sample-project

[34] Serenity BDD, "serenity-cucumber-starter," GitHub. [Online]. Available: https://github.com/serenity-bdd/serenity-cucumber-starter

[35] B. Effron and R. J. Tibshirani, "An introduction to the bootstrap," Monographs on statistics and applied probability, vol. 57, p. 436, 1993.

[36] T. Wolf et al., "Transformers: State-of-the-art natural language processing," in Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations, 2020, pp. 38–45