From Informal to Conformant Models: Benchmarking Vision–Language Models for UML Generation

Cecilia Eklund
Mälardalen University
Västerås, Sweden
ced14002@mdu.se

Tom Jonsson
Mälardalen University
Västerås, Sweden
tjn21007@mdu.se

Riccardo Rubei
Mälardalen University
Västerås, Sweden
riccardo.rubei@mdu.se

Alessio Bucaioni
Mälardalen University
Västerås, Sweden
alessio.bucaioni@mdu.se

Abstract

Software teams routinely sketch and draw informal diagrams to capture design intent, yet these artifacts rarely evolve into conformant models that automated tooling can process or that can support later stages of development. This tension motivates flexible modeling, which aims to bridge free-form sketching and canonical modeling by enabling progressive formalization. Recent advances in vision–language models suggest a new mechanism for operationalizing this bridge by translating informal visuals into canonical modeling artifacts, but evidence on current capabilities remains limited.

In this paper, we provide a systematic empirical benchmark of cross-diagram, cross-model performance of four vision–language models: Claude 3.7 Sonnet, GPT-4o, Llama 4 Maverick, and Gemini 2.5 Pro, for generating five UML diagram types: class, sequence, activity, use case, and state machine. We used a dataset of informal and schematic diagrams extracted from scientific publications and evaluated outputs using automated syntactic validation and structured manual semantic assessment of element and relationship correspondence to the source.

Our results showed that most models produced syntactically valid diagrams with high accuracy, particularly for sequence and use case diagrams, while semantic fidelity remained substantially lower and varied across models and diagram types. Activity diagrams were easiest to generate semantically, whereas use case diagrams were most challenging due to conceptual ambiguity, highlighting a stable syntax–semantics gap that currently limits fully automated design formalization.

Keywords

Software Modeling, Design Automation, Vision–Language Models, Benchmarking, Empirical Evaluation

1 Introduction

Software design models promise abstraction, analyzability, and automated manipulation, and are widely recognized as pivotal to the development and evolution of complex systems [15, 17]. Software engineering disciplines such as Component-Based Development [10], Low-code/No-code Development [3], and Model-Driven Engineering rely heavily on software models and enable transformations, analysis, simulation, and consistency checking. Yet, their widespread adoption remains limited [16]. A key reason is the perceived imbalance between the upfront effort required to create and maintain conformant, well-formed software models and the tangible benefits these models provide during development. In practice, teams often sacrifice strict conformance to modeling languages in favor of speed and flexibility [5].

At the same time, development teams routinely rely on sketches, whiteboard drawings, and informal diagrams to reason about system structure and behavior [13]. These artifacts are genuine abstractions: they capture shared syntax and semantics that remain meaningful within a team and often persist throughout the project lifecycle. Yet, because they typically do not conform to a modeling language, they cannot be directly processed, transformed, or validated by automated tooling or reused in subsequent development tasks. This gap is especially problematic in early-stage design, where domain experts often hold the key business logic but may not have the technical expertise required to produce syntactically valid models. Traditional workflows therefore force a choice between learning formal notation and modeling tools or delegating the task to software engineers, which creates a recurring risk of "lost in translation" between design intent and formal specification.

This tension between free-form artifacts and conformant software models has been explicitly addressed in research on *flexible modeling* [14], which advocates a continuum between informal sketching and strictly conformant software models. The goal is to preserve human-friendly editing freedom while progressively enabling automated actions such as transformation, analysis, and simulation. This perspective is especially relevant in generative Artificial Intelligence (AI)-aided software development, where software design models are both a convenient starting point for hybrid human-AI software development and a key instrument through which humans can retain control over the development process [4]. In this context, the value of AI is not necessarily to produce a final architectural blueprint in one step, but to provide a syntactically valid and discussion-ready starting point that developers and domain experts can refine together. Such an artifact can act as a conversational bridge between the problem space and the notation space: formal enough to support downstream tooling, yet provisional enough to invite correction, clarification, and iteration. Despite this vision, existing approaches to flexible modeling still

depend largely on tool-centric mechanisms and predefined constraints, and bridging informal visuals and machine-processable models remains challenging in practice.

Recent advances in vision–language models introduce a new mechanism for operationalizing this vision. Multimodal Large Language Models (LLMs) can interpret complex visual inputs and generate structured textual representations, suggesting that they may automate the transition from informal sketches to canonical models. Concretely, such models could (i) interpret informal diagrams, (ii) generate structured intermediate representations, and (iii) produce conformant artifacts suitable for downstream manipulation. Whether current models can reliably support this transition, however, remains an open empirical question.

Prior studies have explored adjacent tasks, including generating class diagrams from natural language requirements and converting user stories into sequence diagrams [9, 11]. Other work has investigated translating handwritten or image-based diagrams into formal Unified Modeling Language (UML) diagrams [7, 18]. These efforts have advanced the field, but they often focus on a single diagram type or assume structured inputs, such as class diagrams created with drawing tools. Systematic evidence across multiple UML diagram types, using *informal* image-based inputs and comparable evaluation criteria, remains missing.

In this paper, we provide a systematic empirical benchmark of cross-diagram, cross-model performance of vision–language models for UML generation from informal visual inputs under a unified validation protocol. We benchmark four contemporary multimodal models: Claude 3.7 Sonnet, ChatGPT-4o, Llama 4 Maverick, and Gemini 2.5 Pro, across five UML diagram types: class, sequence, activity, use case, and state machine. We constructed a dataset of informal and schematic diagrams extracted from scientific publications, which we considered a reasonable approximation of software design models. This choice allowed us to build a sufficiently large dataset while reducing potential threats to validity related to replicability, transparency, and handpicking. We evaluated each generated output along two complementary dimensions: syntactic validity (via automated validation) and semantic fidelity (via structured manual assessment). We released the dataset to support independent verification, validation, and replication of our study [1].

Our results demonstrate a consistent pattern: while multimodal models reliably produce syntactically valid UML representations, semantic fidelity remains substantially lower and varies across diagram types. This stable syntax–semantics gap quantifies the current limits of AI-assisted model formalization and grounds the flexible-modeling vision in empirical evidence, clarifying both the feasibility and the boundaries of using multimodal models to transform informal design artifacts into manipulable canonical models.

The remainder of this paper is organized as follows: Section 2 summarizes related work; Section 3 describes our methodology and threats to validity; Section 4 presents the results; Section 5 discusses implications; and Section 6 concludes.

2 Related Work

There is growing interest in automating software design representation from natural language and visual inputs. We summarize the work most closely related to our study and synthesize its main patterns to clarify our contribution.

De Bari *et al.* investigated how a conversational language model generated class diagrams from natural-language software requirements [8]. They compared model-produced diagrams with those created by a human participant and evaluated them along syntactic correctness, semantic accuracy, pragmatic quality, and similarity to a reference solution. Their results showed that model-generated diagrams were comparable to human-produced ones in syntactic and pragmatic dimensions, but exhibited a higher rate of semantic errors.

Jahan *et al.* compared a rule-based approach and a conversational model for deriving sequence diagrams from user stories [12]. Using a dataset of one hundred user stories from public repositories, experts assessed the outputs for accuracy and relevance. They found that the rule-based approach performed better on simpler user stories, whereas the model-based approach adapted more effectively to varied inputs.

Razinskas *et al.* explored the automatic generation of use case diagrams from sketches and videos, focusing on visual recognition and translation of hand-drawn diagrams [18]. They reported an accuracy of 95%. However, their evaluation targeted only use case diagrams and did not compare performance across different UML diagram types.

Conrardy [7] examined how language models translated handwritten class diagrams into formalized class diagrams *et al.*. Although this work addressed visual-to-formal transformation, it concentrated on a single diagram type and did not benchmark multiple models or diagram categories.

Overall, prior studies evaluated isolated diagram types or specific input modalities, and they rarely assessed how models generalize across UML categories under comparable evaluation criteria. In contrast, we systematically benchmark multiple vision–language models across five UML diagram types using informal, image-based inputs and a unified evaluation protocol that assesses both syntactic validity and semantic fidelity. This broader and standardized design enables a direct comparison of cross-diagram generalization, providing empirical evidence on where current models succeed and where they struggle when formalizing informal software design artifacts.

3 Research Methodology

The overall research methodology is structured as a pipeline depicted in Figure 1 and consisting of dataset creation, image extraction, model inference, and syntactic and semantic evaluation.

3.1 Dataset creation

To obtain a set of informal and schematic diagrams, we opted for a lightweight review inspired by the work of Cartaxo et al. [6]. The main motivation was that a more formal and comprehensive literature review would have produced an unsustainable number of candidate papers to analyze.

We therefore retrieved scientific publications from open-access digital libraries and selected arXiv.org as the primary data source

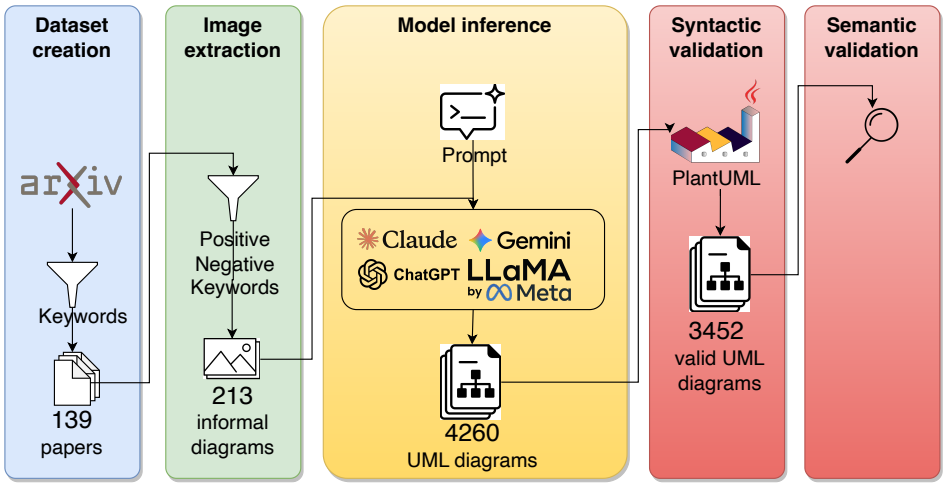

Figure 1: Research Methodology

due to licensing constraints. We used its official API[1] to programmatically query, filter, and download relevant papers. The filtering process relied on a predefined set of software-model-related keywords listed in the following section. Although not exhaustive, this set captured a representative subset of publications likely to contain architectural or design-related visual artifacts. We retrieved, analyzed and ultimately retained 139 papers, which formed the basis for the subsequent image-extraction phase.

Table 1 reports the set of keywords appearing in the different papers together with the frequency of their appearance.

Keyword	Number of papers
software model	179
software architecture	682
software system design	23
high level architecture	57
high level model	89
data pipeline	245
application pipeline	18
data architecture	86
system pipeline	30
block diagram	57
flowchart	94

Table 1: Keywords for Fetching Papers.

We focused on five widely used UML diagram types that capture complementary aspects of software design [19]: *use case*, *class*, *activity*, *sequence*, and *state machine*. These diagram types cover structural, behavioral, and interaction perspectives.

While prior work investigated generating class diagrams from handwritten sketches [7], systematic evaluation across multiple UML categories remains limited. We therefore assessed whether multimodal language models could derive structured representations from informal diagrams across diverse UML types.

[1] info.arxiv.org/help/api/index.html

3.2 Image Extraction

After identifying candidate papers, we extracted images using an external tool designed for retrieving figures from scientific publications[2]. To isolate relevant design images, we applied a filter on the different images' captions. We searched relevant keywords appearing in the different captions, and ultimately, we manually performed a double check to remove irrelevant images. We used two complementary keyword sets:

- Positive keywords: software model, software architecture, system design, workflow, pipeline, overview, framework, informal diagram, block diagram, flow chart, flowchart, architecture diagram, design diagram, component diagram, dataflow diagram.
- Negative keywords: uml, table, histogram, case diagram.

We finally obtained a total of 213 informal diagrams.

3.3 Model Inference

In this phase, we defined a unified prompting strategy and executed model inference. We evaluated four multimodal models: Anthropic's Claude 3.7 Sonnet, Google's Gemini 2.5 Pro, OpenAI's ChatGPT-4o, and Meta's Llama 4 Maverick. We selected these models because they support image-text multimodal input and represent diverse, recent families of large-scale models.

To ensure fairness and reproducibility, we used the same prompt structure across all models without manual adjustments. Each prompt contained *(i)* a Base64-encoded image of the informal diagram and *(ii)* a textual instruction requesting the generation of a corresponding UML diagram in PlantUML syntax. Listing 1 reports an example prompt.

We interacted with the LLMs through APIs using default parameters for temperature, quantization, and related settings. This choice helped us maintain a consistent evaluation setting across models. We intentionally avoided diagram-type-specific prompt tuning to isolate baseline model capabilities, ensure comparability

[2] https://github.com/allenai/pdffigures2

across models, and limit the manual effort required for semantic evaluation. We ultimately generated 4260 UML diagrams, considering the combination of 213 informal diagrams, 4 LLMs, and 5 UML diagrams type.

Listing 1: Prompt employed for LLMs Inference

```
You are a UML expert and PlantUML code generator.
I will give you an **image** of an informal
    diagram.
Your job is to:
1. Analyze the attached image and identify all
    classes, actors, activities, states, and
    interactions.
2. Produce FIVE separate diagrams-in valid
    PlantUML syntax-each under its own heading:
@startuml class_diagram - **Only** output the five
     fenced code blocks. **no extra prose or
    comments outside the code fences**. - ...
    @enduml @startuml activity_diagram ... @enduml
     @startuml usecase_diagram ... @enduml
    @startuml statemachine_diagram ... @enduml
    @startuml sequence_diagram ... @enduml
    Requirements:
If any detail is missing or ambiguous (
    multiplicity, initial node, state entry, actor
     names), **make reasonable assumptions** and
    annotate them inside with note right or note
    left. - Use PlantUML best practices: class for
     classes, actor for actors, state for states,
    usecase for use cases, participant for
    sequence lifelines, and activity flows via
    -->, ->, with fork/join as needed. - Each
    block must start with @startuml <diagram-type>
     and end with @enduml.
```

3.4 Syntactic Validation

We first assessed the syntactic validity of the generated PlantUML code. We decided to employ PlantUML due to its maturity, stability, and, more importantly, its support for syntactic analysis of UML diagrams. We considered a diagram syntactically valid if it could be successfully parsed and rendered. To verify structural correctness without generating images, we used the `checkonly` flag of PlantUML. Diagrams that passed this check were then rendered, whereas those that failed were recorded as invalid and excluded from semantic analysis. This automated validation ensured that the semantic assessment included only diagrams that conformed to the formal syntax of PlantUML. After the analysis, the correct diagrams were 3452, about 81%.

3.5 Semantic Validation

Semantic fidelity concerns whether the generated diagram preserved the meaning of the informal source [2, 20]. Because informal diagrams may admit multiple valid interpretations and no reference models were available, we conducted a structured manual evaluation. We randomly selected ten informal diagrams from the full dataset. For each informal diagram, we generated up to five UML diagram types across four models, yielding a maximum of twenty generated diagrams per input.

$$Candidates = Inf.\_Diagrams(10) \times LLMs(4) \times UML\_Diagrams(5) \quad (1)$$

We evaluated each element and relationship according to four criteria:

- Structure (st): proportion of elements mapped to correct UML constructs. Let E be the set of elements in the informal diagram and $E_{\text{ok}} \subseteq E$ the correctly mapped ones:

$$st = \frac{|E_{\text{ok}}|}{|E|} \times 100 \quad (2)$$

- Flow (f): proportion of relationships correctly preserved. Let R be the set of relationships and $R_{\text{ok}} \subseteq R$ the correctly mapped ones:

$$f = \frac{|R_{\text{ok}}|}{|R|} \times 100 \quad (3)$$

- Semantic (se): proportion of semantic descriptors (e.g., names, labels, properties) correctly retained. Let S be the set of descriptors and $S_{\text{ok}} \subseteq S$ the preserved ones:

$$se = \frac{|S_{\text{ok}}|}{|S|} \times 100 \quad (4)$$

For each diagram d, we computed semantic accuracy as the average of structure, flow, and semantic preservation:

$$\text{SemanticAccuracy}_d = \frac{st_d + f_d + se_d}{3} \quad (5)$$

Diagrams failing syntactic validation were recorded as *N/A* and excluded from percentage and further calculations.

4 Results

In this section, we discuss the performance of the benchmarked models in generating syntactically and semantically correct diagrams. In particular, we compare their ability to produce structurally valid outputs and to preserve the intended meaning of the source diagrams.

4.1 Results of the syntactic validation

The outcomes of the syntactic validation are summarized in Tables 2 and 3. In particular, these tables describe the mean result values from the LLMs and from the UML diagrams perspective.

Table 2: Mean syntactic accuracy per model.

LLM	Syntactic Correctness (%)
ChatGPT-4o	92.6
Claude 3.7 Sonnet	92.6
Gemini 2.5 Pro	59.8
Llama 4 Maverick	84.9

Overall, the evaluated models showed a strong ability to generate syntactically valid diagrams. Table 2 reports the global syntactic correctness for each model, which represents the capability of a Large Language Model to generate a UML model that passes the PlantUML validity check. Claude 3.7 achieved the highest proportion of

Table 3: Mean syntactic accuracy per UML diagram.

UML Diagram	Syntactic Correctness (%)
Activity	67.65
Class	83.4
Sequence	91.35
State Machine	77.95
Use Case	92

syntactically correct diagrams (92.6%), together with ChatGPT-4o (92.6%). Llama 4 reached 84.9% on average, whereas Gemini 2.5 Pro obtained 59.8%. These results indicate that current multimodal models can often produce structurally valid PlantUML code, although robustness varies across model families. Table 3 describes the syntactic correctness per UML diagram. This fashion highlight the intrinsic difficulty faced to generate the particular diagram. For example, the activity diagram, was the most difficult to generate, in fact only the 67.55% were syntactically correct. On the other hand, the use case achieved the highest percentage of correctness (92%).

Figure 2 reports syntactic correctness by diagram type and reveals clear performance differences. Activity diagrams were the most challenging overall. Gemini produced valid activity diagrams in 37.6% of cases and Llama 4 in 54.9%, while Claude 3.7 achieved 80.0% and ChatGPT-4o 98.1%. In contrast, sequence and use case diagrams yielded consistently high validity rates. For these diagram types, all models except Gemini exceeded 90.0% syntactic correctness; Gemini reached 74.6% for sequence diagrams and 80.1% for use case diagrams. These findings show that syntactic reliability depends strongly on diagram type.

4.2 Results of the semantic validation

Tables 4 and **??** reports the mean values for the semantic accuracy, similarly to what we presented for the syntactic analysis.

Table 4: Mean semantic accuracy per model.

LLM	Semantic Accuracy (%)
ChatGPT-4o	57
Claude 3.7	62
Gemini 2.5 Pro	70
Llama 4 Maverick	50

Table 5: Mean semantic accuracy per UML diagram.

UML Diagram	Semantic Accuracy (%)
Activity	71.75
Class	53.25
Sequence	62.75
State Machine	61.75
Use Case	49

The table describes a general difficulty of the model in achieving good results. Interestingly, this time Gemini obtained the best results (70%) followed by Claude (62%) that performed well also in the syntactic analysis. Concerning the accuracy per diagram, Table **??** describes the status. In particular, the more semantic accurate diagram was the Activity diagram with (71.75%).

We next report the semantic evaluation. Figure 3 summarizes semantic accuracy by model and diagram type. Semantic accuracy was consistently lower than syntactic correctness across all models and diagram categories. While syntactic validity frequently approached or exceeded 90%, semantic fidelity remained substantially lower, exposing a persistent gap between producing renderable diagrams and preserving conceptual alignment.

Only one model–diagram combination exceeded 80% semantic accuracy (Gemini on activity diagrams). Across models, activity diagrams achieved the highest semantic scores, whereas use case diagrams consistently yielded the lowest, averaging around 50%. Despite their relatively simple syntax, use case diagrams appear more sensitive to ambiguities in roles and interaction semantics, which weakened conceptual correspondence.

Across all models, the mean semantic accuracy remained above 50%, indicating partial preservation of conceptual content. However, the consistent gap between syntactic validity and semantic fidelity shows that generating renderable diagrams does not imply correct interpretation of informal design intent. Overall, the results provide comparative empirical evidence that current multimodal models are robust at generating structurally valid UML representations, but still limited in reliably preserving semantic meaning.

5 Discussion

In this section, we discuss the qualitative analysis of the results, their implications, and the main threats to validity of the study.

5.1 Qualitative Analysis

The results provide systematic empirical evidence of the current capabilities and limitations of vision–language models as enablers of AI-assisted model formalization. In particular, the consistently high syntactic validity across most models shows that generating structurally well-formed PlantUML representations from informal visual inputs is feasible and robust under our experimental conditions. This capability directly addresses one practical barrier to bring informal design artifacts closer to canonical models: producing tool-processable representations without manual transcription.

In contrast, the semantic evaluation exposes a persistent and measurable gap between structural validity and conceptual fidelity. Although models frequently produced renderable diagrams, they often misrepresented relationships, abstractions, or key entities from the source visuals. This divergence demonstrates that syntactic pattern generation does not imply correct interpretation of design intent. With semantic accuracy typically ranging between 50% and 70%, current models preserved only part of the conceptual content embedded in informal diagrams. This stable syntax–semantics gap aligns with the long-recognized distinction between syntactic conformance and semantic validity in modeling theory, where formal correctness does not guarantee conceptual adequacy.

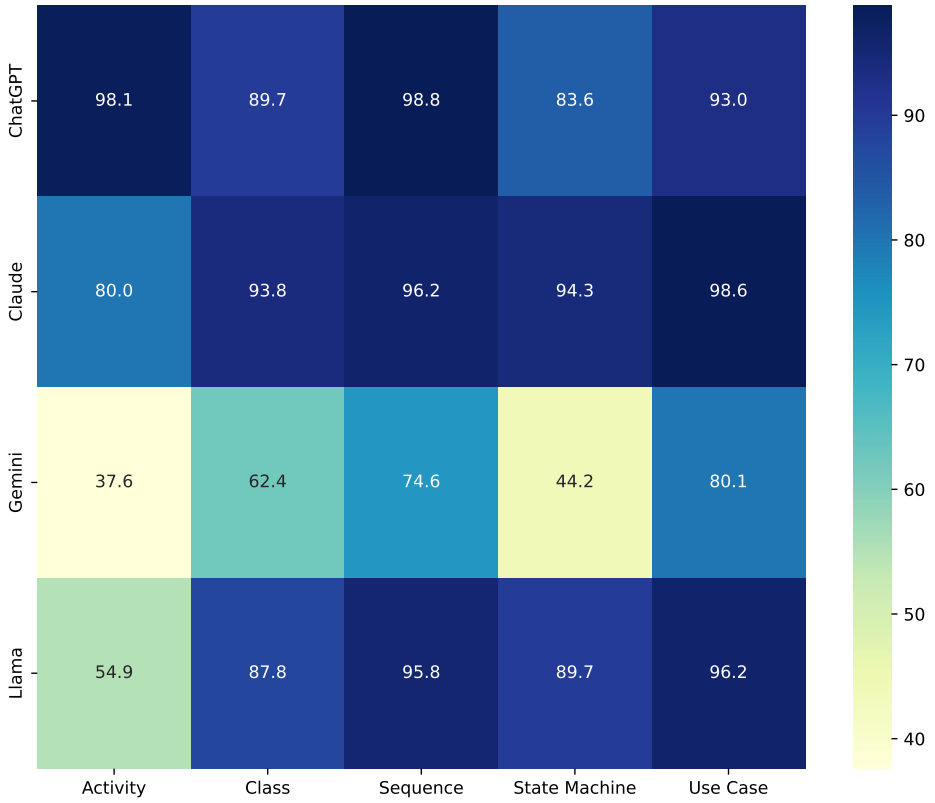

Figure 2: Syntactic correctness (%) of generated diagrams per language model.

To better characterize the nature of this gap, we performed a structured *qualitative inspection* of a subset of 100 generated diagrams, focusing on recurring design patterns and modeling choices across models. Rather than isolated mistakes, we observed consistent behavioral tendencies. We report the names on our repository [1] to facilitate the independent verification of our evaluation.

First, ChatGPT-4o and Llama frequently adopted a conservative translation strategy, simplifying diagrams by omitting structural complexity. In activity diagrams, swimlanes or logical partitions were often absent; in class diagrams, attributes and methods were commonly omitted *ChatGPT_paper_100_Figure2 activity and class diagram*. ChatGPT-4o also tended to avoid expressive constructs such as merge and fork nodes. This strategy often reduced syntactic complexity, but risked semantic under-specification.

Second, Claude and Gemini frequently applied an *enrichment* strategy, introducing additional structural elements such as partitions, attributes, and detailed state definitions *Gemini or Figure_443_Figure1 and Figure_100_Figure2*. This tendency occasionally produced well-structured state machine diagrams with explicit initial, final, and composite states *Figure_100_Figure2*. Gemini uniquely included lifelines in sequence diagrams, indicating a more explicit representation of interaction structure. However, additions also introduced risks: in some cases, Claude inferred incorrect class relationships while elaborating the structure *ClaudeFigure_100_Figure2*.

Interestingly, although Llama exhibited lower syntactic robustness overall, it often attempted to respect formal modeling conventions, such as distinguishing primary and secondary actors in use case diagrams or selecting appropriate relationship types in class diagrams *LlamaFigure_443_Figure1*. In isolated instances, Llama captured nuanced structural aspects that other models overlooked. Furthermore, when processing simpler diagrams, Claude and Gemini occasionally introduced aesthetic refinements such as color variations, while ChatGPT-4o and Claude more consistently produced renderable outputs, whereas Gemini and Llama more frequently failed syntactic validation.

These qualitative patterns complement the quantitative findings and provide an interpretation of the observed syntax–semantics gap. Conservative strategies favor structural safety at the cost of expressiveness, whereas enrichment strategies enhance structural richness but increase the risk of semantic deviation. The gap, therefore, is not merely a matter of accuracy but reflects different generative biases across models.

5.2 Implications

From a practical perspective, these findings position vision–language models as assistive instruments rather than autonomous modeling agents. They can reduce the upfront effort of modeling by generating canonical scaffolds that engineers refine and validate, which

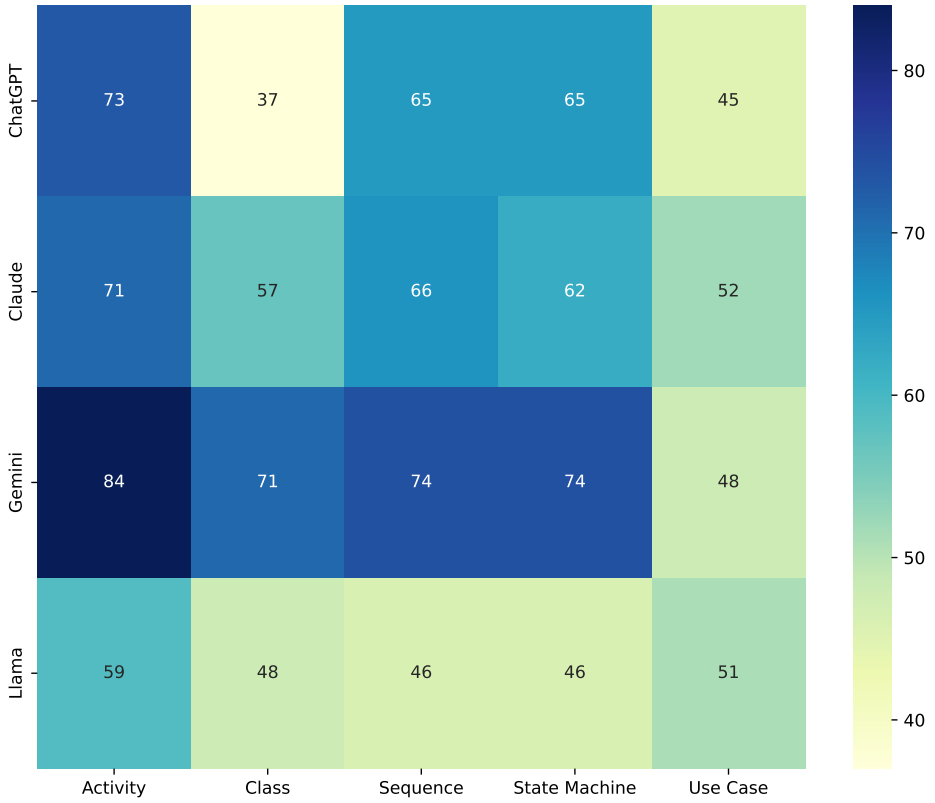

Figure 3: Semantic correctness (%) of generated diagrams per language model.

aligns with the flexible-modeling continuum from free-form artifacts to conformant models. However, because enrichment and simplification strategies introduce different tradeoffs, practitioners must select models based on whether conservative translation or generative augmentation better suits their context. In scenarios where models drive analysis or transformation, systematic human oversight remains essential.

Methodologically, the main strength of this study lies in its comparative and multi-diagram evaluation design. By benchmarking four models across five UML diagram types under a unified validation protocol, we provide reproducible evidence of cross-diagram generalization behavior. This broader perspective complements prior single-task studies and offers a clearer empirical basis for assessing whether multimodal models can serve as a practical bridge between informal visuals and tool-processable models.

These findings should be interpreted within the study boundaries. The dataset was derived from diagrams in scientific publications, which may differ from industrial artifacts in style and complexity. In addition, model outputs depend on prompt formulation and input quality, and alternative prompting strategies may yield different outcomes. These constraints delimit generalizability but do not change the central empirical observation: current models reliably satisfy syntactic constraints, while semantic preservation remains the dominant limitation.

Overall, this study advances empirical understanding of how generative AI can support the flexible-modeling continuum in practice. It shows that structural generation is largely reliable, that semantic fidelity remains the central unresolved challenge, and that distinct generative strategies underlie this limitation. Future research should therefore focus on standardized semantic benchmarks, scalable evaluation procedures, and hybrid human–AI workflows that integrate explicit reasoning mechanisms to narrow the observed gap between syntactic validity and semantic fidelity.

5.3 Threats to Validity

We report on the main threats to validity and their mitigation strategies.

Internal validity. With respect to internal validity, errors may arise in evaluating generated diagrams. We mitigated syntactic threats through automated validation using PlantUML. Semantic evaluation relied on manual assessment by the authors, which may introduce bias. To support transparency and independent verification, we released all informal diagrams and generated outputs in the replication package.

Construct validity. Concerning construct validity, we focused on syntactic validity and semantic fidelity. Other quality aspects, such as visual clarity or stylistic quality, were outside the scope of this exploratory study.

External validity. Our dataset consisted of open-access papers from arXiv.org, which may not represent the full diversity of industrial design practices and affect external validity. Furthermore, manual semantic evaluation limits scalability to larger datasets. In fact, in our evaluation, we limit ourselves to 10 informal diagrams. For each extracted informal diagram, we queried four LLMs, generating five different UML models. We eased this limitation by selecting the input diagrams randomly.

Another aspect pertains to the diversity of the tested prompt. In our evaluation, we employed a single shared prompt used over the entire set of LLMs. In this way, we reduced potential biases arising from unpredictable results or differences in comprehension. On the other hand, we acknowledge that an ad-hoc prompt, specifically conceived to support one diagram (e.g., Class Diagram, Use Case Diagram), could improve the overall quality. However, in this work, we proved the capability of the LLMs to act as an assisting tool to support developers and practitioners in generating human-readable diagrams, aiming at reducing the distance between domain experts, stakeholders, and developers. The creation and the analysis of an advanced prompt to generate compilable, ready-to-use UML diagrams were out of the scope of the paper. Nonetheless, we consider this aspect as an urgent future research line.

Conclusion validity. The exploratory design and limited semantic sample size restrict statistical generalization and, therefore, conclusion validity. However, the systematic validation pipeline and structured evaluation procedure provide confidence in the observed trends regarding current multimodal model capabilities.

6 Conclusion

In this paper, we evaluated the capability of vision–language models to generate UML diagrams from informal, sketch-like visual inputs. We constructed a dataset of diagrams extracted from scientific publications on arXiv and benchmarked four contemporary multimodal models: Claude 3.7 Sonnet, ChatGPT-4o, Llama 4 Maverick, and Gemini 2.5 Pro, across five UML diagram types. We assessed the generated outputs along two complementary dimensions: syntactic validity and semantic fidelity.

The results showed that most models consistently produced syntactically valid diagrams, often exceeding 90% correctness. In contrast, semantic accuracy remained lower, typically ranging between 50% and 70%, revealing a persistent gap between structural compliance and conceptual preservation. While many generated diagrams captured central elements of the source sketches, semantic inconsistencies were frequent. These findings demonstrate that multimodal models are reliable at producing well-formed representations but are not yet robust at faithfully preserving design intent.

Future work will focus on narrowing this gap through reasoning- and retrieval-based strategies, including structured and advanced prompting, retrieval-augmented generation, and controlled multi-agent orchestration. We also plan to develop scalable semantic evaluation mechanisms and standardized benchmarks to enable reproducible assessment across datasets and model generations. Ultimately, advancing automated design formalization will require integrating empirical benchmarking with hybrid human–AI workflows that explicitly address semantic reasoning, thereby strengthening both methodological rigor and practical applicability in software engineering.

7 Data Availability

All data, and generated diagrams are publicly available in the following replication package, enabling independent verification, validation, and replication of our study [1].

Acknowledgment

This work is supported by the Swedish Agency for Innovation Systems through the project "iSecure: Developing Predictable and Secure IoT for Autonomous Systems" (2023-01899), by the Key Digital Technologies Joint Undertaking through the project "MATISSE: Model-based engineering of digital twins for early verification and validation of industrial systems" (101140216), and by the Clean Energy Transition Partnership through the project "FLEXI: Human-centered AI and digital twin powered energy system integration for flexibility markets" (101069750).

References

[1] Anonym. 2026. Replication Package. https://zenodo.org/records/19851707.
[2] Don Baisley, Morgan Björkander, Conrad Bock, Steve Cook, Philippe Desfray, Nathan Dykman, Anders Ek, David Frankel, Eran Gery, Oystein Haugen, Sridhar Iyengar, Cris Kobryn, Birger Møller-Pedersen, James Odell, Gunnar Övergaard, Karin Palmkvist, Guus Ramackers, Jim Rumbaugh, Bran Selic, and Larry Williams. 2005. *Unified Modeling Language: Superstructure. Version 2.0.*
[3] Alessio Bucaioni, Antonio Cicchetti, and Federico Ciccozzi. 2022. Modelling in low-code development: a multi-vocal systematic review. *Software and Systems Modeling* 21, 5 (2022), 1959–1981.
[4] Alessio Bucaioni, Antonio Cicchetti, Gordana Dodig-Crnkovic, Romina Spalazzese, Emma Söderberg, and Dániel Varró. 2026. Engineering Future Critical CPSs with Trustworthy GenAI Across the Lifecycle. In *48th IEEE/ACM International Conference on Software Engineering*. http://www.es.mdu.se/publications/7308-
[5] Alessio Bucaioni, Amleto Di Salle, Ludovico Iovino, Leonardo Mariani, and Patrizio Pelliccione. 2024. Continuous conformance of software architectures. In *2024 IEEE 21st International Conference on Software Architecture (ICSA)*. IEEE, 112–122.
[6] Bruno Cartaxo, Gustavo Pinto, and Sergio Soares. 2020. Rapid reviews in software engineering. In *Contemporary empirical methods in software engineering*. Springer, 357–384.
[7] Aaron David Conrardy and Jordi Cabot. 2024. From image to UML: First results of image-based UML diagram generation using LLMs *(CEUR Workshop Proceedings, Vol. 3727)*. CEUR-WS.org, 55–65. https://ceur-ws.org/Vol-3727/paper5.pdf
[8] Daniele De Bari, Giacomo Garaccione, Riccardo Coppola, Marco Torchiano, and Luca Ardito. 2024. Evaluating Large Language Models in Exercises of UML Class Diagram Modeling. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (Barcelona, Spain) *(ESEM '24)*. 393–399.
[9] Meryem Elallaoui, Khalid Nafil, and Raja Touahni. 2018. Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques. In *The 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018)*. Elsevier. doi:10.1016/J.PROCS.2018.04.010
[10] George T Heineman and William T Councill. 2001. *Component-based software engineering*. Springer.
[11] Mohd Ibrahim and Rodina Ahmad. 2010. Class Diagram Extraction from Textual Requirements Using Natural Language Processing (NLP) Techniques. In *2010 Second International Conference on Computer Research and Development*. 200–204. doi:10.1109/ICCRD.2010.71
[12] Munima Jahan, Mohammad Mahdi Hassan, Reza Golpayegani, Golshid Ranjbaran, Chanchal Roy, Banani Roy, and Kevin Schneider. 2024. 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* (Linz, Austria) *(MODELS '24)*. 138–148.
[13] Robbert Jongeling, Antonio Cicchetti, and Federico Ciccozzi. 2025. How are informal diagrams used in software engineering? An exploratory study of open-source and industrial practices. *Software and Systems Modeling* 24, 3 (2025), 601–613.

[14] Robbert Jongeling and Federico Ciccozzi. 2024. Flexible Modelling: a Systematic Literature Review. *J. Object Technol.* 23, 3 (2024), 1–14.
[15] Hatice Koç, Ali Mert Erdoğan, Yousef Barjakly, and Serhat Peker. 2021. UML Diagrams in Software Engineering Research: A Systematic Literature Review. *Proceedings* 74, 1 (2021). doi:10.3390/proceedings2021074013
[16] Grischa Liebel, Nadja Marko, Matthias Tichy, Andrea Leitner, and Jörgen Hansson. 2018. Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. *Software & Systems Modeling* 17, 1 (2018), 91–113.
[17] Mert Ozkaya. 2019. Are the UML modelling tools powerful enough for practitioners? A literature review. *IET Softw.* 13, 5 (2019), 338–354. doi:10.1049/IET-SEN.2018.5409
[18] Mantas Razinskas, Benas Miliunas, Mantas Jurgelaitis, Lina Ceponiene, and Lina Bisikirskiene. 2024. Transforming Sketches of UML Use Case Diagrams to Models. *IEEE Access* 12 (2024), 185826–185837. doi:10.1109/ACCESS.2024.3514455
[19] Gianna Reggio, Maurizio Leotta, Filippo Ricca, and Diego Clerissi. 2013. What are the used UML diagrams? A Preliminary Survey. *CEUR Workshop Proceedings* 1078.
[20] Lijun Shan and Hong Zhu. 2008. A Formal Descriptive Semantics of UML. In *Formal Methods and Software Engineering*, Shaoying Liu, Tom Maibaum, and Keijiro Araki (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 375–396.