

Large Language Models in Low-Code and Model-Driven Platforms: A Systematic Literature Review

Vasco Nunes

NOVA FCT

2026

*Systematic literature review from Appendix A of the master's dissertation.
Original bibliography included.*

Large Language Models in Low-Code and Model-Driven Platforms: A Systematic Literature Review

Abstract

Context: Low-code development systems are designed to make it easier to create software applications by lowering the technical barriers associated with traditional coding. In most cases, this is achieved through graphical modelling systems. Large language models (LLMs) have emerged as tools for generating structured artefacts from natural-language input in low-code development. Thus, LLMs challenge the need for users to manually define their systems using traditional modelling paradigms.

Objective: The purpose of this systematic literature review is to analyse the application of Large Language Models in Low-Code and model driven platform systems. The main objective is to catalogue, categorise, and assess how large language models are integrated into these platform systems. Specifically, we seek to understand the current methodologies and intermediate representations employed to produce artefacts such as workflow, rules, and models from natural language prompts.

Method: We conducted a systematic literature review following the guidelines of Kitchenham and Charters. Database searches returned 69 records, of which four were duplicates. The final sample, identified through database searches and snowballing, comprised 35 studies published between 2022 and 2025. We extracted data on integration architectures, generated artefacts, intermediate representations, validation mechanisms, and evaluation results.

Conclusion: The reviewed studies describe approaches that combine LLM generation with intermediate representations, validation, and user review. Some studies report efficiency gains for the tasks evaluated, but semantic errors and the effort required for review and

correction remain concerns. Differences in tasks and evaluation methods limit direct comparisons between approaches.

Keywords: Low-Code Development, Large Language Models, Model-Driven Engineering, Systematic Literature Review, Generative AI.

A.1 Introduction

With the emergence of both Low-Code Development Platforms (LCDPs) [59], and Model-Driven Engineering (MDE) [13], there has been an important evolution in the field of software development. Both paradigms plan to facilitate the process of developing applications by simplifying the complexity of coding, allowing developers and domain experts to develop applications using visual interfaces and high-level models. Although, as noted by [60], historical automation in these types of environments, generally, rely on deterministic transformations and therefore can be very inflexible and difficult to scale.

However, another type of paradigm shift is occurring, as a result of the rapid growth of Generative AI. Large Language Models (LLMs) have demonstrated impressive capabilities for understanding natural language, as well as generating code. Therefore, according to a recent systematic literature review [61], LLMs have moved the focus of software engineering research from simple code completion to more complex generative tasks and therefore, raise questions about how they can best be utilised in structured modelling environments.

Although the possibilities of LLMs may be promising, utilising them in engineering environments encounters many of the same obstacles as they do in general purpose programming. Unlike the relatively flexible nature of standard scripting, utilising LLMs to create structured outputs within MDE is much more restrictive because, unlike simply following syntactic rules, the generated output must conform to the strict metamodels of engineering models. This is emphasised by [62], who state that one of the major challenges facing the utilisation of LLMs in MDE is bridging the gap between probabilistic AI models and the rigid requirements of engineering models. Furthermore, although the field of study concerning the intersection of these two areas is growing rapidly, it remains fragmented and scattered throughout different disciplines.

Therefore, this systematic literature review examines how Large Language Models are being utilised in both Low-Code and model-driven platforms. It identifies the current methods, representations and challenges associated with the automated creation of structured artefacts. Specifically, the review intends to identify, categorise and analyse the approaches that utilise LLMs in these environments. Additionally, the review will examine the methods, representations, and validation techniques employed in creating structured artefacts automatically.

The remainder of this review is organised as follows: Section [A.2](#) introduces the

background and related work. Section A.3 describes the review methodology, including the research questions in Section A.3.1. Section A.4 presents the results, and Section A.5 discusses their implications. Section A.6 addresses threats to validity, and Section A.7 concludes the review.

A.2 Background and Related Work

This section provides details on the main topics addressed by this review: Low-Code Development (LCD), Model-Driven Engineering (MDE), and Large Language Models (LLMs). It also discusses relevant articles and other Systematic Literature Reviews found in the literature to contextualise the present study.

A.2.1 Low-Code and Model-Driven Engineering

The Model-Driven Engineering (MDE) methodology represents a paradigm shift within software engineering where the focus is on models as the primary development artefacts, using them to define how the system works, instead of just describing it [13]. In doing so, MDE seeks to address complexity issues while improving productivity through the generation of executable systems using automated transformations of abstract models to concrete models or other representations. Automation was at the core of MDE, however, it was implemented via strict model transformations that translate abstract specifications into concrete code or other model representations [14].

Similarities exist between MDE and Low-Code Development Platforms (LCDPs), specifically the use of graphical user interfaces, visual metaphors, and configuration tools to rapidly deliver applications with minimal hand coding [59]. Richardson and Rymer [5] emphasise that LCDPs allow non-programmers to create applications without requiring extensive programming knowledge. Additionally, LCDPs can leverage model-driven paradigms under the hood to generate executable applications from visual specifications. Therefore, LCDPs represent an industrial realisation of MDE methodologies [63].

A.2.1.1 Challenges of Automation in MDE

While traditional automation in MDE relies on deterministic transformations, i.e., Model-to-Model (M2M) and Model-to-Text (M2T) transformations defined by strict transformation rules [64], they have been successful for well-defined domains. However, they are typically inflexible and difficult to scale, thus requiring maintenance efforts.

More recently, researchers have demonstrated a growing requirement for more flexible and intent-based automation strategies, as seen in the studies [62], [48]. Research by Hagel et al. [9] reinforces this aspect, identifying that traditional Low-Code platforms still require users to write code in DSLs and understand metamodels, which creates a barrier for them to have a true "No-Code" experience and also does not allow the environment to flexibly interpret vague user input.

A.2.2 Large Language Models in Software Engineering

The emergence of Large Language Models (LLMs) represents a significant step forward in the development of Artificial Intelligence (AI). As a result of advancements in transformer architecture, developed by Vaswani et al. [28], LLMs are able to recognise and utilise large amounts of training data for both text and source code in order to identify numerous patterns and structures within multiple programming languages. Unlike previous deep learning models, LLMs have demonstrated an ability to perform well in few-shot learning and reasoning, and therefore require little to no additional training data to be effective in new areas.

As stated previously, the utilisation of AI in the field of Software Engineering (SE) has progressed significantly. The ability of AI to perform complex generative functions has grown, and LLMs have become a viable option to automate tasks throughout all aspects of the software lifecycle. LLMs have been explored for tasks such as code generation, testing, program repair, and requirements engineering. These models are currently being utilised for a variety of SE-related tasks such as automated coding, automated testing, automated program repair [44], and automated requirements engineering.

While there are many benefits to utilising LLMs for engineering-related tasks, there are also several challenges associated with this method of operation. As mentioned by Buchmann et al. [65], one challenge that exists is the "black-box" nature of LLMs. The authors state that it can be difficult for engineers to understand how the model arrived at a particular piece of code, and thus there are concerns surrounding trust and maintainability. In addition, Wang et al. [66] observed that while LLMs perform very well at creating general-purpose scripting, the performance of LLMs can vary depending on the specific domain and the prompting strategy used. Due to the probabilistic nature of LLMs, they tend to generate code based on the probability of the next token, rather than based on determinism [67]. As a result, LLMs can exhibit "hallucinations," where they create code that is syntactically correct, but functionally incorrect.

A.2.3 The Intersection: AI-Augmented Modelling

With the convergence of Generative Artificial Intelligence (AI), and Model-Driven Engineering (MDE), a new paradigm for engineering is evolving. This paradigm is typically referred to as "AI-Augmented Modelling". Unlike the automation of traditional MDE, which relies on the manual definition of transformation rules, AI-Augmented Modelling utilises Large Language Models (LLMs) to define those transformations based upon either informal or semi-formal input provided to them. This offers a new method to connect natural language requirements with formal models.

The combination of these two disciplines has the potential to result in huge changes.

It was suggested in [48] that LLMs can serve as translation layers, turning ambiguously worded textual descriptions into Domain Specific Languages (DSLs). This would enable modelling to be accessible to users who do not have programming skills, similar to

Low-Code tools, however would allow the user to express their modelling intent in conversational format. In addition, [68] suggested that the integration of a chat-based interface with a low-code platform will allow designers to move past the complexity of the UI interaction, enabling designers to concentrate on the high-level design logic of the design.

However, the integration of these two disciplines presents a new challenge: how to connect the probabilistic nature of LLMs with the deterministic nature of engineering models. Unlike the semantics of natural languages which can be somewhat vague or variable, models in MDE must stick to their metamodels and structural constraints in order to be executable. [38] empirically demonstrated in the generation of Systems Modeling Language (SysML) models that while LLMs can generate valid code structures with a very high degree of syntactic correctness, they can also frequently produce semantic hallucinations. The findings presented by this research support the need for additional validation checks on the generated models by humans, as well as the need for intermediate validation checks to ensure the dependability of the AI-generated artefacts in safety-critical systems.

A.2.4 Related Work

Existing secondary research (Systematic Literature Reviews and Surveys) that are positioned at the interface between AI and Software Engineering were analysed to place this study in the larger body of academic research. The majority of reviews addressing both areas separately do not provide an overall classification of integration techniques for Generative AI in Low-Code environments.

LLMs for Software Engineering: The systematic literature review [61], along with the extensive survey by Fan et al. [26], provided an overview of the use of LLMs in software engineering in general. These works covered in detail many of the software engineering tasks, including code generation, test synthesis, and repair of programs.

These reviews address software engineering more broadly. The present review focuses on the integration of LLMs into Low-Code and model-driven platforms, including intermediate representations and the validation of generated models.

MDE for Machine Learning: [69] performed a review of how model-driven engineering could be applied to the creation of Machine Learning (ML) components (named "MDE4ML"), utilising modelling concepts to manage ML complexity. Unlike the work presented in [69], which explores applying engineering models to AI, our review investigates the inverse relationship of "AI for MDE" (utilising AI to create software engineering models), which has a different set of validation and correctness issues.

LLMs can generate models that are syntactically valid but do not meet the intended requirements. Their outputs therefore need to be checked for both syntactic and semantic correctness.

Automation in MDE: Bucchiarone et al. [70] pointed out that the complexity of

formal languages and usability issues are still significant barriers to the adoption of MDE. Addressing these issues, the article [62] has documented how MDE has evolved towards greater use of automation by identifying LLMs as a key area of advancement in the field of MDE and suggesting that hybrid methods using both symbolic processing techniques and generative models are an appropriate way to move forward.

LLMs in MDE: Di Rocco et al. [71] conducted a systematic review of LLM applications in MDE and the use of MDE to support LLMs. They organised the reviewed work by modelling task and discussed technical integration, evaluation challenges, and future research directions.

Current Fragmented Landscape: Even though research about the inclusion of Generative AI in Low-Code is growing, it remains fragmented. There are many studies centred on how developers can assure the usability and performance of AI-assisted programming tools. Bucchiarone et al. [70] and Burgueño et al. [62] provide examples of how the technical aspects of current modelling languages are a major barrier in terms of flexibility and complexity when adopting Visual Low-Code Platforms. Therefore, there is a need to develop a next generation of flexible and automated systems to address these problems.

This SLR provides a systematic categorisation of the technical methods used to integrate LLMs into Low-Code and MDE systems. Specifically, we seek to classify how structured intermediate artefacts are created using LLMs and the mechanisms used to check syntactic and semantic correctness.

A.3 Methodology

This SLR was performed using the guidelines for performing systematic literature reviews in software engineering developed by Kitchenham and Charters [12]. The method is intended to provide a reproducible approach to the review process, and the entire review process is supported by Parsifal, a web based tool to support researchers in the planning, execution, and reporting of systematic literature reviews.

The review process is organised into three major components: planning, conducting, and documenting. The planning component identified the research objectives, defined the research questions, search strategy, and the inclusion/exclusion criteria, which were used to identify the most relevant studies. The conducting component implemented the search strategy across the selected digital libraries, filtered the results according to the previously determined criteria, and systematically extracted data from the selected studies. Finally, the documentation component analysed and synthesised the extracted data to determine answers to the research questions and report the findings as they appear in this document.

Below are the subsections that detail the protocol applied in this study, which begin with an identification of the research objectives and the questions that guided this investigation.

A.3.1 Goals and Research Questions

This review is primarily focused on analysing the current state of the art concerning the combination of LLMs with Low-Code and Model Driven Engineering. We have identified that our research questions will focus upon identifying how LLMs create artefacts/intermediate representations, the ones that are utilised for bridging a language model’s output and an executable model’s input, and validation techniques that can be used to confirm that the generated model is correct.

To organise the investigation and determine the scope of the literature review, we employed the PICOC framework (Population, Intervention, Comparison, Outcome, Context) as shown in Table A.1.

Table A.1: PICOC Analysis.

Criteria	Description
Population	Studies, frameworks, and platforms related to Low-Code and model-driven development.
Intervention	The use of Large Language Models to generate, assist, or validate structured artefacts such as rules, workflows, and models.
Comparison	Traditional rule/model generation methods without AI assistance, or non-LLM approaches.
Outcome	Identified methods, intermediate representations, validation techniques, and challenges in integrating LLMs.
Context	Low-code and model-driven development environments, practical application within the DISME platform.

Based on the results of our search scope, we defined four Research Questions (RQs), which are used to define the data extraction and synthesis process:

- **RQ1: How are Large Language Models (LLMs) integrated into Low-Code and model-driven platforms?** This question will investigate the different design and technical methods that authors have employed to include LLMs in a Low-Code environment. It analyses the architectural coupling, that is, how the model is connected to the environment. In addition, it explores the interaction between the LLM’s generative capability and the modelling tool’s determinism.
- **RQ2: What types of intermediate or structured representations are used between natural language inputs and executable artefacts?** Since direct generation of a model is very difficult at times, we want to determine how some intermediate representations (like JSON, DSLs, or graph structures) help separate the probabilistic representation of LLMs from the determinism required by most platforms.
- **RQ3: What mechanisms ensure syntactic and semantic correctness of LLM-generated artefacts?** Since LLMs can produce false information due to hallucination,

understanding the current validation techniques (such as human-in-the-loop validation, compiler-based validation, etc.) that are being utilised to ensure the model's reliability is critical.

- **RQ4: How are these approaches evaluated, and what benefits or limitations are reported in AI-assisted Low-Code development?** We want to measure the level of maturity within this area through empirical data on the improvements in efficiency, usability, and issues associated with this approach versus traditional old methods.

A.3.2 Search Strategy and Study Selection

We have developed a search strategy for identifying all relevant studies by focusing on the intersection of three primary concepts: (1) the paradigm of software development (Low-Code or Model Driven); (2) the underlying enabling technology (large language models); (3) the outcome we want from using this technology (automation or generation).

The search string is designed to combine each concept with their respective synonyms and alternative spellings in order to maximise the retrieval of all studies.

("Low-Code" OR "No-Code" OR "Model-Driven" OR "MDE") AND ("Large Language Model" OR "LLM" OR "Generative AI" OR "GPT") AND ("Generation" OR "Automation" OR "Modelling")

The search was conducted on 5th December 2025 across four major digital libraries recognised for their relevance in software engineering research: ACM Digital Library¹, IEEE Xplore², ScienceDirect³, and Springer Link⁴. This process resulted in a total of 69 initial articles. Figure A.1 provides a breakdown of the number of results retrieved from each database.

¹<https://dl.acm.org/>

²<https://ieeexplore.ieee.org/>

³<https://www.sciencedirect.com/>

⁴<https://link.springer.com/>

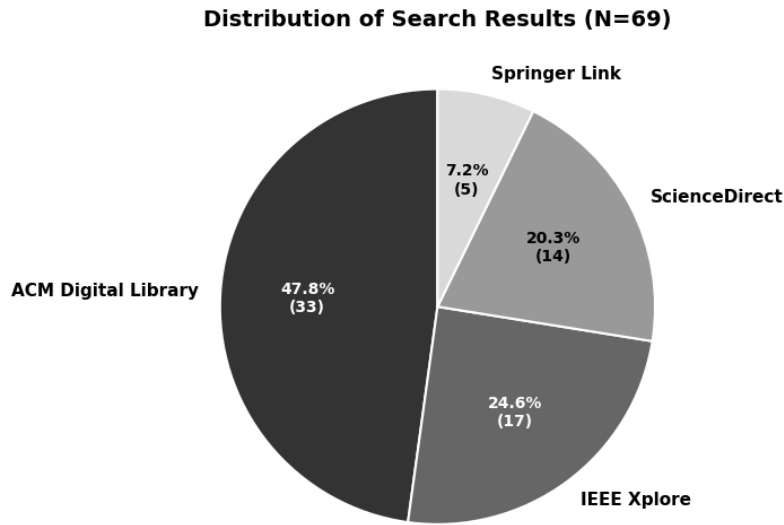


Figure A.1: Distribution of the 69 initial search results across the selected digital libraries.

To guarantee the results were focused on the research topic, the search was limited to the title, abstract, and keywords of the papers where possible.

Inclusion and Exclusion Criteria

To filter the initial set of results and select only the studies that directly address the research questions, a set of Inclusion Criteria (IC) and Exclusion Criteria (EC) were defined.

The **Inclusion Criteria** were as follows:

- IC1: The paper discusses or applies Large Language Models (LLMs) or transformer-based models.
- IC2: The paper must be written in English.
- IC3: The paper must have been published between 2020 and 2025.
- IC4: The study is peer-reviewed.

The **Exclusion Criteria** were defined to remove irrelevant studies:

- EC1: Papers not peer-reviewed (e.g., blogs, posters, editorials, or tutorials).
- EC2: Papers with insufficient technical detail.
- EC3: Papers written in languages other than English.

Figure A.2 summarises the study selection process.

The initial database search returned 69 records. After removing four duplicates, 65 records remained for screening. Of these, 37 were excluded after assessment against the inclusion and exclusion criteria, leaving 28 studies.

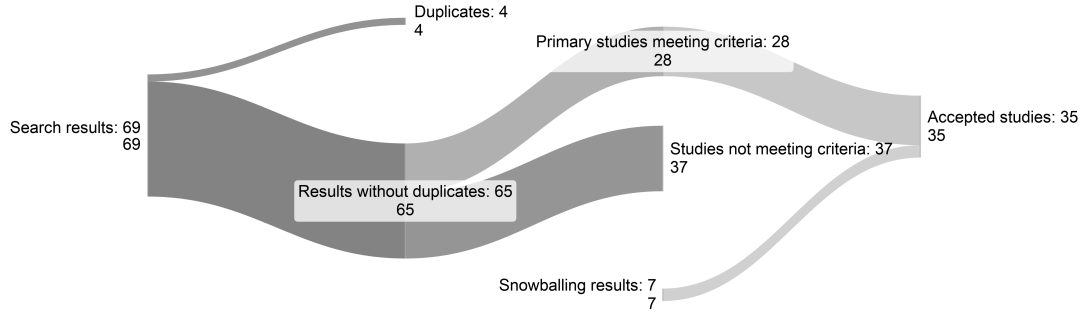


Figure A.2: Study selection flow from database searches and snowballing to the final sample.

Forward and backward snowballing added seven studies, resulting in a final set of 35 selected studies. These studies are identified as S01–S35 in Table A.2.

Table A.2: List of selected studies.

ID	Title	Authors	Year	Ref.
S01	LLM4Workflow: An LLM-based Automated Workflow Model Generation Tool	Xu et al.	2024	[41]
S02	AI for Low-Code for AI	Rao et al.	2024	[72]
S03	Auto-DSM: Using a Large Language Model to Generate a Design Structure Matrix	Koh	2024	[42]
S04	Automated Implementation Scheme for Low-Code Development Based on Large Language Models	Sun et al.	2024	[39]
S05	Automation in Model-Driven Engineering: A Look Back and Ahead	Burgueño et al.	2025	[62]
S06	ChatOps for Microservice Systems: A Low-Code Approach Using Service Composition and Large Language Models	Wang et al.	2024	[73]
S07	Complex Flowchart Text Generation Based on Large Language Models	Fang and Fan	2025	[74]
S08	Combining Low-Code Development with ChatGPT to Novel No-Code Approaches: A Focus-Group Study	Martins et al.	2023	[68]
S09	Enhancing Accessibility for Non-experts in Low-Code/No-Code Through the Addition of an AI Chatbot	Troyer et al.	2025	[75]
S10	D2: An LLM Agent Driving End-to-End Visual AI modeling in Energy Platforms	Li et al.	2025	[43]
S11	FlowPilot: A Generative AI-Driven Visual Language for Computational Thinking Education	Turchi et al.	2024	[40]
S12	Human-Centric Hybrid-AI for No-Code Development	Silva and Hildebrandt	2025	[48]
S13	Introducing LLMScript: A Turing Complete Prompt-Based Scripting Language for LLMs with No External Coding Required	Mowzoon and Faustini	2024	[76]
S14	Low-code solutions for business process dataflows: From modeling to execution	Eldin et al.	2025	[77]
S15	Prompt Sapper: A LLM-Empowered Production Tool for Building AI Chains	Cheng et al.	2024	[78]

Table A.2: List of selected studies (continued).

ID	Title	Authors	Year	Ref.
S16	Using Grammar Masking to Ensure Syntactic Validity in LLM-based modeling Tasks	Netz et al.	2024	[50]
S17	Turning Low-Code Development Platforms into True No-Code with LLMs	Hagel et al.	2024	[9]
S18	Neurosymbolic Repair for Low-Code Formula Languages	Bavishi et al.	2022	[46]
S19	InstructPipe: Generating Visual Blocks Pipelines with Human Instructions and LLMs	Zhou et al.	2025	[47]
S20	White-box LLM-supported Low-code Engineering: A Vision and First Insights	Buchmann et al.	2024	[65]
S21	Automated Generation of AML Models for Industrial Plants Using LLM Chat Applications	Kunze and Fay	2024	[79]
S22	Power Grid Model Generation Based on the Tool-augmented Large Language Model	Deng et al.	2025	[80]
S23	Toward efficient vibe coding: An LLM-based agent for Low-Code software development	Malamas et al.	2025	[51]
S24	MetaCloze: a schema-guided automated building metadata model generation system via information extraction	He et al.	2025	[81]
S25	Constructing Enterprise Digital Twins by Augmenting LLMs with MDE	Barat et al.	2025	[82]
S26	Using (Not-so) Large Language Models to Generate Simulation Models in a Formal DSL: A Study on Reaction Networks	Kreikemeyer et al.	2025	[83]
S27	ChainBuddy: An AI-Assisted Agent System for Generating LLM Pipelines	Zhang and Arawjo	2025	[84]
S28	Generating SysML Behavior Models via Large Language Models: an Empirical Study	Wang et al.	2025	[38]
S29	Fully Autonomous Programming with Large Language Models.	Liventsev et al.	2023	[44]
S30	Program Synthesis with LLM-Predicted Minimal Specialized Grammars	Zarb	2025	[85]
S31	Fully Autonomous Programming Using Iterative Multi-Agent Debugging with Large Language Models	Grishina et al.	2025	[45]
S32	Exploring Influence Factors on LLM Suitability for No-Code Development of End User Applications.	Wang et al.	2025	[66]
S33	Software Model Evolution with Large Language Models: Experiments on Simulated, Public and Industrial Datasets	Tinnes et al.	2025	[10]
S34	“What It Wants Me To Say”: Bridging the Abstraction Gap Between End-User Programmers and Code-Generating Large Language Models	Liu et al.	2023	[20]
S35	Accurate and Consistent Graph Model Generation from Text with Large Language Models	Chen et al.	2025	[49]

For the temporal analysis, publication years were harmonised using the earliest publication year documented in the publisher’s records, including advance online publication. Bibliographic references retain the year of the cited edition. S04 is counted in 2024 based on the proceedings date, although its bibliographic reference is dated 2025. S09, S14, and S32 are counted in 2025 based on their online publication, although the cited editions are dated 2026.

A.3.3 Data Extraction and Synthesis

To ensure that we extracted and recorded the data systematically from each accepted study, we created an extraction form (Table A.3). The extraction form consists of two categories of fields. The first category includes fields that deal with the general information about each paper (i.e., Year Published, Study Type, Platform Characteristics), which allows us to see what areas of research are currently being pursued. The second category consists of fields that contain each study's contribution and enable us to answer our research question.

The fields in the first category include metadata related to each study, and therefore have a more structured format. This category's fields regarding Authors, Year, Source Venue allow us to understand where the current state of research is at concerning LLMs in Low-Code/MDE. The fields such as *Platform Type* and *License/Accessibility*, allow us to understand the openness of the platforms being researched and who they are intended to reach.

The second category includes fields tied to the research questions for this dissertation. For RQ1 (e.g., *LLM/Model Used*, *Model Access Type*, *Output Type*), we used fields to find architectural patterns and ways of integrating models. To address RQ2 regarding intermediate representations, we recorded data on *IR Type*, *IR Specification Method*, and *Persistence Format*. The fields such as *Syntax Enforcement Mechanism*, *Human-in-the-loop Level*, and *Validation Type* were used to address RQ3, which focuses on the correctness and reliability of the systems. Finally, the fields such as *Evaluation Method*, *Main Findings*, and *Reported Challenges* were used to address RQ4, and provided information on the *level of maturity and limitations of current solutions*.

Each study's extraction form was completed based upon the information available in each study. Some studies did not address one or more of the fields in the second category related to the research questions. When this occurred, the respective field was completed as "n/a" (not applicable) or "Not Reported". The fields in the second category that were qualitative (e.g., **Main Findings**, **Reported Challenges**) were completed with direct information from the studies. Since the fields were open-ended, and to provide a consistent interpretation of the results for the discussion in Section A.5, they were also summarised and grouped into common topics.

A.4 Results and Analysis

As outlined in Section A.3.3, we extracted both qualitative and quantitative data from the 35 selected sources detailed in Section A.3.2 by employing the data extraction forms. The purpose of this chapter is to report on the results of the qualitative and quantitative data extraction from the viewpoint of the research questions identified earlier. By doing this, it will synthesise the results to give an overall account of the current state-of-the-art of utilising LLMs in Low-Code and model-driven engineering systems.

Table A.3: Data extraction form fields.

Description	Type	Possible values (examples)	Associated RQs
<i>First group: General information</i>			
Title / Authors / Year	String Field	n/a	n/a
Source / Venue	Select One	ACM, IEEE, ScienceDirect, Springer, Other	n/a
Type of Study	Select One	Conference Paper, Journal, Workshop, Thesis	n/a
Platform Type	Select One	Low-code, No-code, Hybrid	n/a
License / Accessibility	Select One	Open Source, Commercial, Hybrid	n/a
<i>Second group: Contributions</i>			
LLM/Model Used	Select One	GPT, Llama, Claude, PaLM, Not used	RQ1
Model Access Type	Select One	API, Fine-tuned Local, Tool-Augmented	RQ1
Output Type	Select Many	Code, Model, Workflow, Rule	RQ1
Uses Intermediate Rep (IR) ?	Select One	Yes, No	RQ2
IR Type	Select Many	DSL, Graph, JSON, UML/Diagram, Grammar	RQ2
IR Specification Method	Select One	Grammar, Metamodel, No Specification	RQ2
Syntax Enforcement Mechanism	Select Many	Grammar Masking, Constrained Decoding, Parser-based	RQ3
Human-in-the-loop Level	Select One	Co-editing, Review-only, Iterative, Prompting	RQ3
Validation Type	Select One	Syntactic, Semantic, User Study, Formal	RQ3
Evaluation Method	Select One	Benchmark, Case Study, Experiment, Simulation	RQ4
Main Findings	String Field	n/a	RQ4
Reported Challenges	String Field	n/a	RQ4

The results are organised by research question. Section [A.4.2](#) presents integration approaches (RQ1). Section [A.4.3](#) examines intermediate representations (RQ2). Section [A.4.4](#) discusses correctness and validation mechanisms (RQ3). Section [A.4.5](#) summarises evaluation methods, benefits, and limitations (RQ4).

A.4.1 Study Metadata and Research Trends

The purpose of this section is to perform the quantitative analysis of the 35 selected studies included in the review. An overview of the temporal distribution, type of publication outlets, geographic location of the authors, and type of institution involved in the studies

will help us to gain a better understanding of the current state of the art in the area of LLMs applied to Low-Code development.

Figure A.3 shows the distribution of the 35 selected studies by year: one in 2022, three in 2023, twelve in 2024, and nineteen in 2025. Most selected studies were therefore published in the final two years of the review period.

The selected studies comprise 21 conference or workshop papers and 14 journal articles, as shown in Figure A.4. Conference and workshop papers are grouped together because they are published in proceedings. Figure A.5 shows the distribution of these publication types by year.

Figure A.6 groups the 35 studies by their authors' affiliations: 20 (57.1%) involve only universities or public/non-profit research institutions, 3 (8.6%) involve only companies, and 12 (34.3%) include both types. The mixed category also includes studies in which an author lists both an academic/research affiliation and a company affiliation.

Figure A.7 shows study counts by country, while Figure A.8 shows author counts based on the affiliations listed in each paper. Each author is counted once per paper and country, so authors may contribute to more than one country and are counted again when they appear in another paper. China has the highest author count (39), followed by the United States (32) and Germany (21). Hong Kong affiliations are included under China, while Taiwan is shown separately.

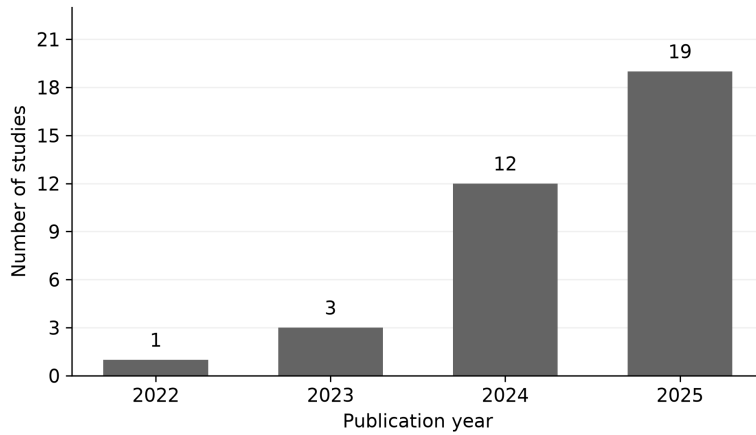


Figure A.3: Distribution of selected studies per year (2022–2025).

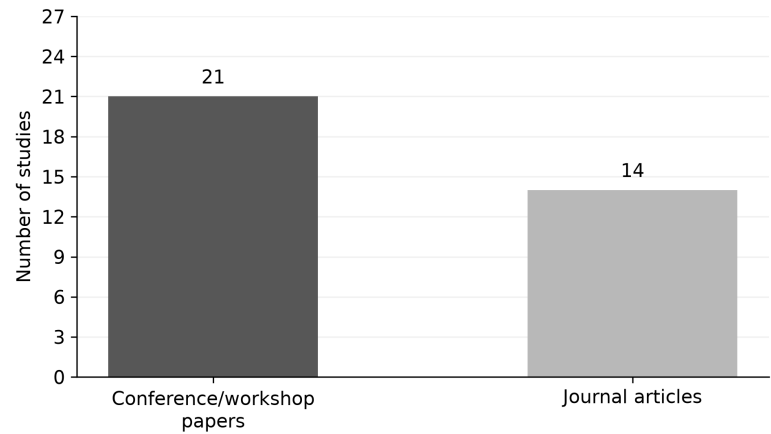


Figure A.4: Distribution of selected studies by publication venue type.

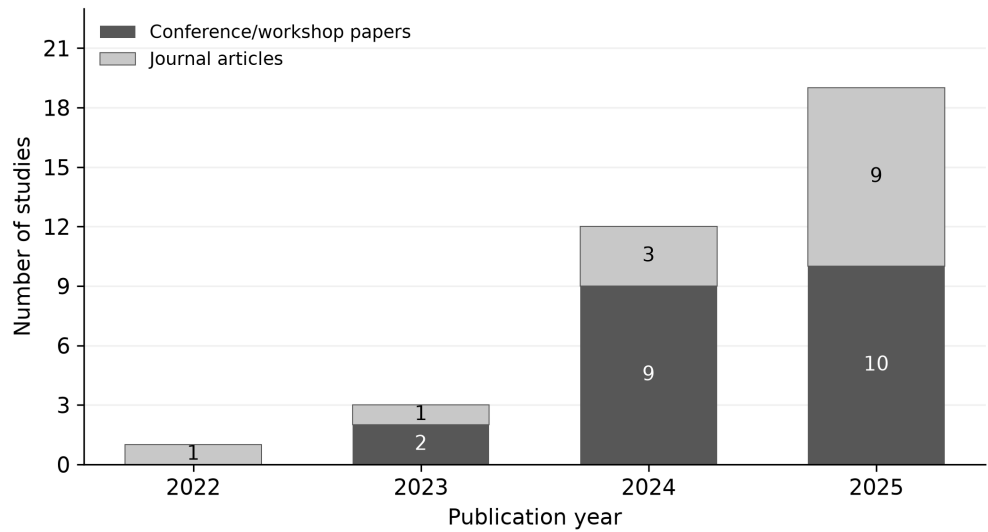


Figure A.5: Evolution of publication venues over the years.

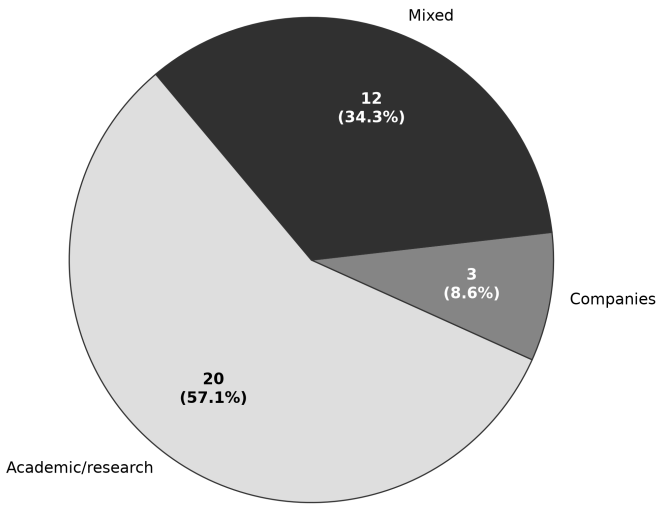


Figure A.6: Distribution of studies by author affiliation type.

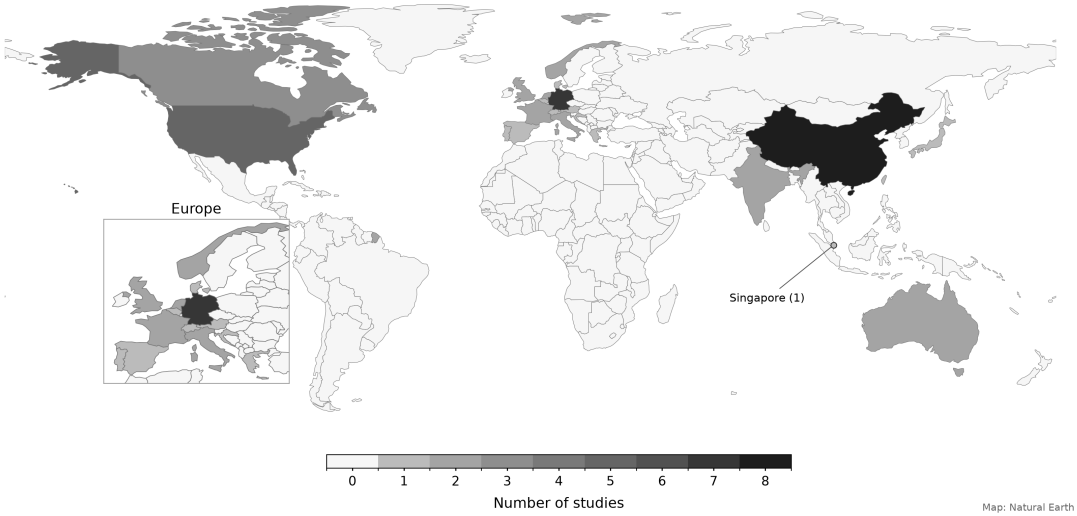


Figure A.7: Geographical distribution of selected studies on a world map.

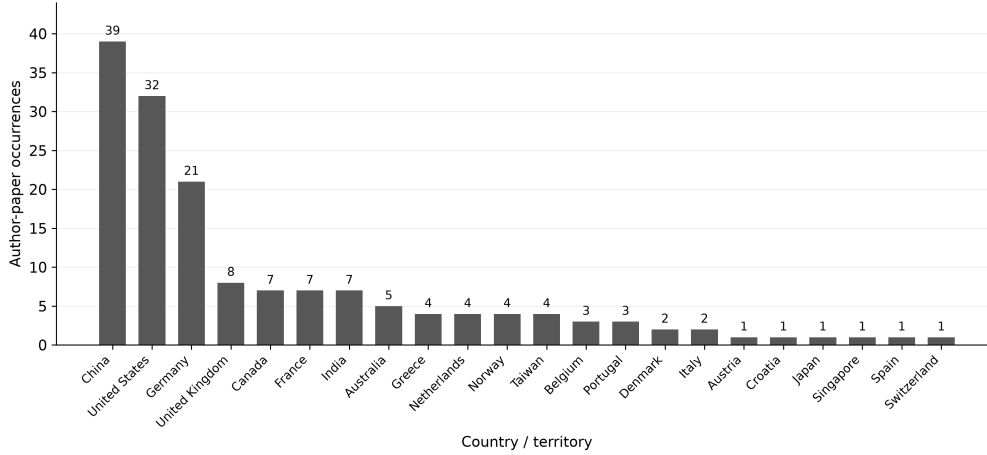


Figure A.8: Number of authors per country, based on their institutions.

A.4.2 RQ1: How are Large Language Models (LLMs) integrated into Low-Code and model-driven platforms?

The subject of RQ1 is how LLMs are being incorporated into Low-Code Model-Driven Systems. According to the literature, there was no single methodology for this; rather, there were many methodologies with varying degrees of technology and autonomy within the systems. Table A.4 shows the *integration methods* identified.

An evaluation of the selected studies shows that the integration of LLMs into modelling environments has evolved past simple interaction interfaces. As indicated in the literature, there appears to be an evolution in the architectural control flow from simple interaction interfaces to autonomous loops. Three types of integration levels were determined based on the degree of autonomy allowed to the model: stateless direct generation, retrieval augmented generation, and agentic feedback loops.

Stateless prompt engineering is one of the integration approaches identified in the review. Studies included in the review (i.e., [39], [38]) illustrate direct generation: a user enters a natural language description of their desired artefact, and the LLM generates the artefact in a single pass, without showing how it was generated. For example, Wang et al. [38] illustrate the application of stateless prompt engineering for generating SysML models directly from text. In a stateless interaction, the model does not retain information from previous requests unless it is included again in the input. The prompt can still contain examples, documentation, and project-specific constraints. These results typically require manual validation by the user.

RAG (retrieval augmented generation) is used in various research studies to provide additional project-specific context. The RAG strategy gives additional context (documentation, pre-existing code bases, metamodels) to the original input for the model to improve the quality of the produced outputs. Examples of how researchers have utilised RAG can be seen in [10] and [41]. Tinnes et al. [10] utilise the version history of a particular

project to serve as a knowledge base allowing the model to learn industry specific pattern recognition not available to models without this knowledge. Xu et al. [41] implemented a "Rewrite-Retrieve-Read" method to check if the correct API was retrieved before executing the generation phase thereby reducing the amount of error that will result from the specific domain.

Other reviewed approaches use agentic feedback loops or multi-agent systems. Different from usual linear pipeline architectures, these paradigms create a recursive environment where the system evaluates its own output. Liventsev et al. [44] propose a "Synthesise-Execute-Debug" loop where the system executes the generated code and uses the result of the compilation errors as input to the model to enable self-correction. Likewise, Grishina et al. [45] studied a system that generates programs, runs tests, uses failures to suggest fixes, and keeps the best programs for another round.

The selected studies describe both direct generation and approaches that combine generation with additional context, constraints, or feedback. Their results do not establish that one architecture is more effective across all tasks or that the generated artefacts are ready for production use without further validation.

Table A.4: Integration Techniques of LLMs in Low-Code Platforms

Technique	Studies
Prompt Engineering	[S04], [S06], [S07], [S08], [S09], [S11], [S13], [S14], [S17], [S20], [S21], [S25], [S28], [S30], [S32], [S34]
Multi-Agent Systems	[S01], [S10], [S15], [S19], [S22], [S23], [S27], [S29], [S31]
Grammar-Guided	[S12], [S16], [S18], [S26], [S30]
RAG (Retrieval-Augmented)	[S01], [S03], [S10], [S13], [S23], [S33]
Model Fine-Tuning	[S02], [S24], [S26], [S35]

The interaction paradigms employed define the usability of the platform (Table A.5). The Conversational (Chat) approach is common (13 studies), functioning as an assistant (i.e., [73]) that operates alongside the user.

However, in engineering contexts, the Human-Validated Generation (Human-in-the-loop) approach is critical (13 studies). Study [82] provides a good example of this paradigm, where the system produces a draft version of the model, which the engineer must validate before it is accepted.

On the other hand, Fully Automated approaches (8 studies), as exemplified in [49], attempt to create complete solutions without human intervention, which raises additional issues related to verifying consistency.

We evaluated the types of artefacts created as a result of using different integration models (Table A.6). The results indicated a variety of artefacts created. Code and logic

Table A.5: Interaction Paradigms between User and LLM

Paradigm	Studies
Conversational (Chat)	[S02], [S04], [S06], [S09], [S10], [S13], [S15], [S17], [S21], [S22], [S23], [S27], [S34]
Human-Validated (Draft & Approve)	[S01], [S08], [S11], [S12], [S14], [S18], [S19], [S20], [S25], [S26], [S28], [S32], [S33]
Fully Automated	[S03], [S07], [S16], [S24], [S29], [S30], [S31], [S35]

were identified in 10 studies, with an example being [40], where the model lets the user define a new block in the visual language by describing the behaviour they want. The model will generate the code in JavaScript for that block, and the user can connect it visually to other blocks. We also have executable system models (10 studies) with [79] as an important example, where the purpose is to automatically generate the topology of an industrial system in AutomationML (AML) by reading legacy physical documentation from brownfield plants.

The workflow area (8 studies) shows the ability to use LLMs to create process orchestrations. In [77] for example, this study proposes BPDML (Business Process Data Modeling Language), an extension of BPMN/DF-BPMN. The language introduces the concept of "Activity DataFlow", where each activity explicitly specifies inputs, outputs, data stores, and data operations. Lastly, conceptual visual models/graphs were identified in 5 studies. Study [49] addresses the generation of consistent graph models.

Table A.6: Types of Artefacts Generated by the Integration

Artefact Category	Studies
Conceptual Visual Models	[S03], [S10], [S28], [S33], [S35]
Executable System Models	[S01], [S12], [S16], [S17], [S21], [S22], [S23], [S24], [S25], [S26]
Workflow and Orchestration	[S01], [S06], [S07], [S14], [S15], [S19], [S27], [S32]
Code and Logic	[S02], [S04], [S11], [S13], [S18], [S20], [S29], [S30], [S31], [S34]

In conclusion, the integration of LLMs has evolved from simple text prompts to hybrid architectures restricted by grammars [50]. It is becoming increasingly clear that the generation of models for critical applications should always be structured, and validated by a human [48].

A.4.3 RQ2: What types of intermediate or structured representations are used?

The second Research Question was to determine what types of representations were used to bridge the natural language input and the final modelling artefacts. This question aimed to understand how the representation would facilitate the validation of the input and make it easier to visualise the structure of the input. To answer the second research question, the results were categorised into two groups: Language-Based Representations (Table A.7) and Data-Structure Based Representations (Table A.8).

Language-Based Representations are based on a formal textual syntax to define the models. As can be seen in Table A.7, the most frequent type of representation is **Domain Specific Languages** (DSLs), which are designed for a specific context, allowing for more concise code compared to general languages. A very good example is [48], where the proposal follows the premise that the LLMs alone are not trustworthy for critical systems, because they are non-deterministic and hallucinate. The solution is a hybrid approach, using the LLM just to translate the user's natural language to a DSL legible by humans, allowing traditional formal methods to handle the actual execution safely.

To verify that the generated language is valid, many studies utilise Grammars and Schemas (3 studies). [85] is the principal study that uses Backus-Naur Form (BNF) as a key intermediate representation. In this study, the LLM receives examples of problems and their respective "mini grammar" (BNF) and generates a new BNF grammar for the target problem. This reduces the search space and thus lets the algorithm find a solution significantly faster.

The standard modelling languages, such as Unified Modeling Language (UML) and SysML, are commonly utilised (3 studies), as demonstrated in [38]. This study employs PlantUML as a textual intermediate representation to generate SysML models. By using this format, the LLM can describe the behavioural semantics of complex systems, such as states and transitions, which are then rendered into the final visual diagrams.

At last, Controlled Natural Language (3 studies), provides an interface between the users and their code. Rather than presenting users with raw code [20], it presents "grounded utterances", structured natural language that explains the fundamental Python operations. Users can then see the intent (i.e., "Step 1: Create column 'Year' from 'Date'") before the actual execution of the code. Grounded utterances can help users understand the planned operations and check whether they match their requirements.

On the other hand, Data-Structure Based Representations are typically used when a user needs to execute their model directly through machine-based processing, or if they want to integrate it onto web based platforms (as listed in Table A.8). Among Data-Structure Based Representations, JSON and Structured Commands are the most common representations (9 studies). For example, [80] uses JSON object as an intermediate representation of the electrical model that it wants to represent.

Also, unlike linear code, Serialised Graphs and Diffs are used to represent graphs,

Table A.7: Formal and Language-Based Representations (Language-Based)

Representation Type	Studies
Domain-Specific Languages (DSLs)	[S06], [S12], [S14], [S16], [S17], [S23], [S26]
Modelling Languages (UML/SysML/AML)	[S21], [S28], [S35]
Controlled Natural Language	[S13], [S19], [S34]
Grammars & Schemas (BNF/EBNF/Ontology)	[S16], [S24], [S30]

as we can see in [49]. These formats include both the nodes and edges to preserve the topological integrity of the graph.

Table A.8: Structural and Data Representations (Data-Structure Based)

Representation Type	Studies
JSON and Structured Commands	[S01], [S02], [S04], [S10], [S15], [S19], [S22], [S27], [S32]
Serialised Graphs and Diffs	[S01], [S07], [S12], [S33], [S35]
Matrices	[S03]
Abstract Syntax Tree (AST)	[S18], [S20]

A.4.4 RQ3: What mechanisms ensure syntactic and semantic correctness of LLM-generated artefacts?

The third Research Question investigates how the quality of the created models can be ensured. The studies have employed several methods for ensuring that the generated output is correct (Table A.9).

The most common method is Automated Verification and Repair, which is used in 8 studies. This method verifies the correctness of the LLMs generated artefacts by testing them. When they contain errors, such as incorrect results due to hallucination, it attempts to repair them automatically. A notable study using an automated verification and repair methodology was [43]. This study uses a "Validation by Execution" technique. A predictive workflow is generated and tested for accuracy. If the model performance is low, the agent enters an optimisation loop, automatically suggesting repairs, from adjusting parameters to modifying features, until the results meet the required standard.

Three studies used constraint-based generation. In Netz et al. [50], grammar masking restricts generation to outputs that follow the specified grammar. This prevents the syntactic errors covered by that grammar, but does not establish semantic correctness or remove the need to check whether the generated model represents the intended requirements.

Static Analysis and Filtering (5 studies) are used for validating the generated output (rules, syntax, grammar) without running the model in a real-world environment. The

studies [9] and [51] can be seen as examples of how this method is applied. In Hagel et al. [9], as soon as the DSL code is generated, it is validated directly by the platform’s compiler to catch syntax errors. Similarly, [51] focused on achieving the maximum "Valid Model Rate" by validating that the output meets the DSL grammar, thus ensuring that it could run.

Tool-assisted human verification is reported in seven studies. It allows users to inspect generated artefacts and request corrections. This provides an opportunity to identify errors, but does not guarantee that every error will be detected or corrected.

Table A.9: Correctness Assurance Mechanisms

Strategy / Mechanism	Studies
Automated Verification and Repair	[S10], [S18], [S22], [S28], [S29], [S31], [S32], [S35]
Tool-Assisted Human Verification	[S01], [S02], [S04], [S07], [S12], [S33], [S34]
Static Analysis and Filtering	[S02], [S14], [S17], [S23], [S24]
Constraint-Based Generation	[S16], [S26], [S30]

A.4.5 RQ4: How are these approaches evaluated, and what benefits or limitations are reported?

This final research question aims to complement the technical discussion of the integration methods by looking at the practical impact of LLMs in Low-Code development. It examines the evaluation methodologies used and the reported advantages and limitations, as seen in Tables A.10 and A.11, respectively.

The selected studies report both benefits and limitations. Some report improvements in efficiency for the tasks evaluated, while others identify semantic errors and the effort required to review or correct generated artefacts. Because the studies use different tasks and evaluation methods, their results do not establish a general productivity advantage.

Reported efficiency gains need to be interpreted in the context of each evaluation. One of the ways improvements in efficiency were evaluated was through the use of a formula proposed by Sun et al. [39] for measuring improvement, where it is considered *Time*, *Cost*, *Errors* as main factors: $E = (W_t \times \Delta\mathbf{Time}) + (W_c \times \Delta\mathbf{Cost}) + (W_e \times \Delta\mathbf{Errors})$.

Using this expression applied to data from JD.com/China Unicom, they reported a 60% increase in Efficiency (the **E** in the formula) in contrast to the traditional ways of developing. Likewise, comparative studies on accessibility by Hagel et al. [9] and Malamas et al. [51], measured "Valid Model Rate" and "time-on-task", both found that the use of AI-based tools has greatly lowered the threshold of entry for non-expert developers as well as reduced the amount of time required for developing first-draft DSL code.

However, the increased productivity can be negatively impacted by the lack of quality and reliability concerning semantic consistency. Although grammar-guided decoding has

greatly reduced the number of syntactic errors (as shown in [50]) the semantic hallucination continues to be a major problem. Both Wang et al. [38] and Tinnes et al. [10] reported that the model generates syntactically correct artefacts, but does not capture the project-specific logic or industry constraints unless provided with external context (RAG). Reviewing and debugging the generated artefact takes additional time and can reduce the time saved during generation.

Liventsev et al. [44] evaluated a Synthesize-Execute-Debug approach that tests generated programs and uses feedback from failed tests to guide repair. Using OpenAI Codex on Program Synthesis Benchmark 2, their framework outperformed generation without a repair phase and the genetic programming approaches used for comparison.

Ultimately, the evaluation results suggest that although LLMs provide a demonstrable increase in both speed and accessibility, there is a need for strong verification methods to ensure that LLMs will be viable in industry. It appears that the transition from "generation", to "iterative repair" is the key to transforming unreliable probabilistic output into reliable engineering models.

Table A.10: Evaluation Methodologies

Evaluation Methodology	Studies
Automated Benchmarks and Datasets	[S07], [S16], [S18], [S24], [S26], [S28], [S29], [S30], [S31], [S35]
Industrial Case Studies & Pilots	[S01], [S03], [S04], [S10], [S11], [S13], [S14], [S20], [S21], [S22], [S25], [S33]
Controlled User Study	[S02], [S06], [S09], [S15], [S17], [S19], [S23], [S27], [S32], [S34]
Focus Groups, Surveys & Vision	[S05], [S08], [S12]

The literature has identified a trade-off between the efficiency and reliability in using LLMs as described in Table A.11. As stated earlier, productivity was found to be the most common positive outcome when using LLMs for development, because they automate many repetitive tasks associated with code generation and configuring workflows. However, reviewing and correcting generated output also takes time and must be considered when assessing productivity.

Although Natural Language Processing (NLP) has greatly enhanced the Accessibility and Usability of software engineering (making it easier for many people to develop applications) as well as the participation of non-technical users in the application development process through enabling them to describe the application's requirements using natural language (as seen in [68] and [20]), there are still usability barriers that exist.

For example, vague user prompts produce poor results and because most users are unaware of the abstraction gap between the user's intent and the machine-readable form of the user input [[72], [51]], they may have difficulty obtaining useful results from an NLP based tool.

The reviewed studies also report limitations in the quality and reliability of generated artefacts, despite the use of automated checks and repair [[44], [45]]. The non-deterministic nature of LLMs is the primary reason for hallucinations, which can create considerable threats to high-stakes applications. As seen in [38], syntactic errors were mostly resolved, but semantic issues persist, suggesting that the feedback mechanism using formal verification techniques needs further investigation to help LLMs better understand complex requirements and logical relationships.

Table A.11: Reported Benefits and Limitations

Impact Dimension	Benefits (+)	Limitations (-)
Efficiency and Productivity	[S03], [S04], [S10], [S14], [S15], [S17], [S19], [S25], [S32], [S33]	[S16], [S27]
Accessibility and Usability	[S02], [S08], [S09], [S11], [S20], [S23], [S34]	[S02], [S11], [S19], [S27]
Quality and Reliability	[S29], [S31], [S33], [S35]	[S07], [S13], [S21], [S26], [S27], [S28]

A.5 Discussion

While we managed to acquire some insights regarding our research questions thanks to the selected studies, it is pertinent to look at what these findings mean for the LLM use in Low-Code and model-driven platforms.

A.5.1 Common Architectures for Integrating AI

We verified that the integration of Large Language Models (LLMs) into Low-Code platforms has evolved over time in terms of their degree of complexity and how their integration technique, ranging from simple external assistants to deep, contextualised integration.

One approach is to integrate the LLM as a copilot or side-by-side assistant. This type of architecture was used by [40], for example, in which the LLM is positioned as a separate service or a ChatBot or code completion tool layered over the existing IDE. The benefits of using this type of architecture are numerous, including being easy to implement since no modifications to the underlying platform are needed. However, the main drawback of this type of architecture is that the LLM is unaware of the larger project state. [10] states that, with a lack of understanding of the larger project state or the modelling context, the LLM fails to predict correct evolutions, generating artefacts that may be syntactically correct but are functionally incorrect relative to the rest of the application.

Because of this, a significant number of researchers recommend using retrieval-augmented generation (RAG) architecture to address the drawbacks associated with the copilot architecture. RAG architectures align with much of the more extensive AI

literature Lewis et al. [86] that uses context to provide additional information to the model prior to generating output. As highlighted in [10] that the LLM needs to understand the environment and context it is in, so this study found that the use of RAG architecture significantly improves the performance of industrial applications of LLM. RAG can provide project-specific documentation and examples before generation. The generated artefacts still require validation against the target language and platform constraints.

Some reviewed approaches organise generation, testing, and repair into an iterative process. Liventsev et al. [44] and Grishina et al. [45] use test results to guide changes to generated programs. These approaches allow candidate solutions to be revised after execution, rather than relying on a single generation step.

Liventsev et al. [44] found that combining code repair with the generation of new alternatives gave better results than using either strategy alone in their experiments. The reviewed studies also indicate a trend toward hybrid neuro-symbolic architectures in which the probabilistic output of the LLM is constrained and validated by the deterministic output of the Low-Code engine.

A.5.2 The Importance of Intermediate Formats

Traditional Model-Driven Engineering uses extremely complex file formats, such as XML Metadata Interchange (XMI) and large JSON schema definitions. The issue with these large and rigidly-structured files is that they are difficult for LLMs to process correctly, their complex formatting makes it harder for the model to understand the structure. Similarly, trying to have LLMs generate the final file using these formats, generally results in failures. An effective solution to this problem is to define an (IR), which will act as a bridge between the LLM and the actual final system that is being created, and it helps to understand what the model actually generates

An important realisation regarding IRs is that they should also be readable by humans. This requirement aligns with many of the more recent principles of Human-Centred AI [87]. Specifically, one of the primary principles of Human-Centred AI is that to achieve both safety and trust, AI systems must provide users with high levels of interpretability and controllability. However, if the AI generates a complex block of code, then, the user will not be able to verify whether or not the code is correct. Therefore, to overcome this issue, researchers such as Liu et al. [20] proposed using "Grounded Utterances," i.e., structured text-based steps that describe the logical flow of the code in plain English. Thus, the user can quickly determine the intent behind the AI-generated code and evaluate whether or not the code is correct by looking at it, reducing significantly the cognitive barrier between vague and informal requirements and formal syntax. Along with this, researchers such as Silva and Hildebrandt [48] proposed developing a DSL that is easier for the average user to read and understand than complex graphs, thus providing more clarity into how the system behaves from the perspective of the domain expert.

In addition, using simpler intermediate formats reduces the risk that the AI will become

confused about the structure of the data, consistent with Amershi et al. [88] guidelines for minimising ambiguity when interacting with humans. Additionally, researchers such as Zhou et al. [47] demonstrated that having the model generate "pseudocode" prior to generating a visual graph (or other output) is far more likely to produce reliable results than attempting to generate a visual graph directly from the model, due to the reduced amount of syntactical errors present in larger JSON files.

Ultimately, while the selection of an appropriate intermediate representation is a technical decision, it also means an important non-technical decision, since the selected IR must be simple enough for the LLM to process, but also clear enough for the human user to trust.

A.5.3 Ensuring Reliability in AI-Generated Models

Intermediate formats represent a step towards improving the interpretability of Generative Models, however, they do not overcome the fundamental probabilistic nature of generative models. Unlike the usual, deterministic tools, LLMs will produce plausible, although incorrect outputs, an issue famously described by Bender et al. [89] as the "Stochastic Parrots" problem. The review has demonstrated that the reliability of generative models represents a multi-layered challenge that evolves from simple syntax validation to complex hybrid validation methods.

One of the important distinctions found within the literature is between syntactic validity (is the code valid? Does the code compile?) and semantic correctness (does the code perform the function requested by the user?). Netz et al. [50] demonstrate that techniques such as "Grammar Masking" can be used to enforce the strictest adherence to a language's syntax, resulting in near perfect syntactic validity. However, as Wang et al. [38] point out, a model may generate syntactically correct code that still does not perform as expected due to hallucinations, where the model produces incorrect logic despite valid syntax. [44] refer to this as "Near Miss Syndrome", where the output appears correct upon initial inspection, however, upon execution it performs incorrectly.

Some reviewed approaches use iterative repair to address semantic errors. Liventsev et al. [44] use feedback from failed tests to guide the repair of generated programs. Grishina et al. [45] generate candidate programs, test them, use the detected failures to guide repair, and rank the candidates for further iteration.

The reviewed literature also explores combining neural models with symbolic methods to support reliability. Silva and Hildebrandt [48] propose connecting generative AI and symbolic models through a formal DSL that users can understand. Bavishi et al. [46] use neural techniques to locate errors and rank possible repairs, while symbolic methods generate repairs for low-code formulas.

A.5.4 Balancing Speed with the Need for Review

Although LLMs are capable of reducing the physical effort involved in coding, the gain in productivity is largely counterbalanced by the cognitive cost of verification. Zhou et al., [47] found that most of the time, users "pause" and evaluate whether the AI's result was intended to achieve what the user originally wanted to do, thus turning the user into an "auditor" rather than an author. The fact that users need to validate the results also complicates the evaluation process because, as Tinnes et al., [10] point out, although the generated model may be syntactically correct, it may still fail when executed in the context of the current project. In the end, the effectiveness of AI-assisted platforms is based upon how quickly and efficiently they can reduce the effort needed by humans to verify their work.

Productivity gains should therefore be assessed using the total time required for generation, review, and correction. Faster generation does not necessarily reduce the overall effort if extensive correction is still needed.

A.5.5 Current Trends and Industry Interest

The main aspect of the reviewed literature is the shift from theoretical research on integrating Large Language Models (LLMs) into practical, industrial research applications. Some of the reviewed studies were conducted in collaboration with industrial partners and originated from leading industrial firms, suggesting a shared commitment among major market participants to invest in this area and democratise software modelling.

In addition to the "Big Tech" players, significant interest exists in specialised engineering areas. Examples include the development of Enterprise Digital Twins using metamodel-augmented LLMs (to ensure that generated simulations really show complex constraints inherent in an organisation) by researchers working at TCS Research [82].

Industry investment is pushing the trend toward greater autonomy. While earlier versions of such systems operated as passive copilot systems, the latest version of the research being done (e.g., ChainBuddy [84] and the iterative loop-based study done by [44]), suggests that future versions of such systems will operate as semi-autonomous. These semi-autonomous agents will not only be able to generate code, but also to plan, execute, and correct workflow sequences independently of human operation, therefore, human oversight of their operation will be necessary.

Silva and Hildebrandt [48] propose a research agenda for no-code development that combines generative AI with symbolic models. Their approach uses an intermediate domain-specific language to describe behavioural scenarios in a formal representation designed for users to read and understand.

The reviewed examples combine LLM generation with structured representations, execution feedback, or validation [44, 48, 82, 84]. They explore different architectures, and the review does not establish a single approach suitable for all industrial applications.

A.6 Threats to validity

In the next subsections we discuss the threats to validity of this work under several perspectives, following the classification of construct, internal, external, and conclusion validity.

A.6.1 Construct validity

Regarding the search string, we included "Low-Code", "Large Language Models" and "Model-driven Engineering". The terminology related to this subject is changing very quickly, so it is possible that there are some additional studies that we missed because they were using other names for these types of technology. In addition, we limited our searches to four major online libraries: ACM, IEEE, ScienceDirect and Springer. Since research in Artificial Intelligence moves very fast, we might have missed some important recent work that has not yet been published in traditional journals or conferences.

A.6.2 Internal validity

To ensure a rigorous process, we utilised Parsifal tool and worked within a well-defined protocol. As such, the decision to include or exclude articles as having insufficient technical detail may be somewhat subjective. We helped to mitigate some of these issues by performing a snowballing process to locate other relevant articles in the references cited in each article we located during our first search. However, we believe that the selected articles cover most of the high-quality publications in LLMs for Low-Code and model-driven platforms.

A.6.3 External validity

One major limitation is that the rate of technological advancement is so rapid that many of the articles we reviewed were published between 2024 and 2025. The studies evaluated specific models and versions, so their findings may not generalise to later models. Another limitation of prior research is that most of the studies have been conducted by researchers who work in universities, for that reason, they typically test prototypes rather than commercially available products. In addition, because university-based testing environments tend to be much less complicated than commercial environments, the results may be very different when applied to complex legacy systems used in a company setting.

A.6.4 Conclusion validity

Finally, this validity refers to the relationship between our conclusions and the extracted data. We used a specific form to extract data from every paper systematically. However, some fields, such as the "Reported Challenges", required us to interpret the text written by the original authors. Different researchers might interpret these descriptions differently. Additionally, the various papers used very different ways to evaluate their tools, ranging from user surveys to automated benchmarks. This variety makes it difficult to make a direct comparison and say for sure which method is the best. Ultimately, we expect more research in these areas to benefit both academia and the Low-Code industry, promoting the development of trustworthy and efficient AI-enhanced modelling platforms.

A.7 Conclusions and future work

We aimed to learn and understand more about how Large Language Models (LLMs) have been integrated into Low-Code and Model-Driven Engineering (MDE) systems. This review analysed 35 selected studies published between 2022 and 2025. We followed a defined protocol to identify the integration architectures, the intermediate representations used to structure the output, and the validation mechanisms required to ensure the correctness of the generated models.

The reviewed studies describe approaches ranging from direct prompting to pipelines with structured intermediate representations and validation. Intermediate representations, such as DSLs, JSON, XML, and graphs, connect natural-language input with the formats required by modelling platforms. They can also make generated artefacts easier to inspect and check before use.

Some studies report reductions in the time needed to create initial drafts and models. However, reviewing and correcting generated output also takes time and should be considered when assessing productivity. Human review, automated repair loops, and deterministic checks are among the approaches used to identify and address errors. Differences in tasks, tools, and evaluation methods limit direct comparison between these approaches.

For future work, we suggest three main directions based on the gaps we identified. Firstly, there is currently no standardisation regarding the benchmarks used to evaluate the relative qualities of various tools. Because each of the studies we reviewed utilised a unique methodology to assess success, it is difficult to compare the results of the individual studies. Establishing a common set of tasks that will be used to benchmark modelling capabilities will enable the community to identify which architectural approaches offer advantages over others. Second, we believe that future investigations should focus on developing "semantic validation" capabilities. Most existing tools are capable of identifying syntactical errors (i.e., grammar errors) in models, however, they continue to struggle with

identifying logical errors (i.e., semantic errors). Third, we believe that a greater emphasis should be placed upon developing intermediate representations that are both clearer and more intuitive to users, even if they do not have any coding experience. Developing a clear and easy to envision intermediate representation is important because it enables users to clearly visualise the output produced by the model, facilitating inspection and correction of the generated output. Ultimately, this helps to align the model's production of output with the user's intended outcome.

These findings informed the design of the DISME wizard presented in Chapters 4 and 5. The wizard combines LLM generation with structured representations, deterministic checks, and user review. These mechanisms support the detection and correction of errors before import, while the evaluation in Chapter 6 examines the limitations that remain in practice.

BIBLIOGRAPHY

- [1] J. M. Lourenço. *The NOVAtesis L^AT_EX Template User's Manual*. NOVA University Lisbon, 2021. URL: <https://github.com/joaomlourenco/novathesis/raw/main/template.pdf> (cit. on p. i).
- [2] The L^AT_EX Project team. *L^AT_EX – A document preparation system*. 1985. URL: <https://www.latex-project.org> (cit. on p. v).
- [3] J. M. Lourenço. *The aidisclose package: Generative AI disclosure checklist and statements*. Version 1.13.0. 2025. URL: <https://github.com/joaomlourenco/aidisclose> (cit. on p. v).
- [4] Y. Suchikova et al. “GAIDeT (Generative AI Delegation Taxonomy): A Taxonomy for Humans to Delegate Tasks to Generative Artificial Intelligence in Scientific Research and Publishing”. In: *Accountability in Research* (2025), pp. 1–27. DOI: [10.1080/08989621.2025.2544331](https://doi.org/10.1080/08989621.2025.2544331) (cit. on p. v).
- [5] C. Richardson and J. R. Rymer. *New Development Platforms Emerge For Customer-Facing Applications*. Report. Forrester Research, 2014-06. URL: <https://www.forrester.com/report/RES113411> (cit. on pp. 1, 6, 110).
- [6] V. Freitas et al. “The DISME low-code platform – from simple diagram creation to system execution”. In: *Proceedings of the 22nd CIAO! Doctoral Consortium, and Enterprise Engineering Working Conference Forum 2022*. Vol. 3388. CEUR Workshop Proceedings. CEUR-WS.org, 2023. URL: <https://ceur-ws.org/Vol-3388/paper4.pdf> (cit. on pp. 1, 8–12, 15).
- [7] J. L. Dietz and H. B. Mulder. *Enterprise Ontology: A Human-Centric Approach to Understanding the Essence of Organisation*. Springer International Publishing, 2020. DOI: [10.1007/978-3-030-38854-6](https://doi.org/10.1007/978-3-030-38854-6) (cit. on pp. 1, 7, 9).
- [8] V. H. S. Freitas. “Extension of Action Rule Grammar and Implementation of Processing Engine of a DEMO Based Low-Code Platform”. MA thesis. Universidade da Madeira, 2023 (cit. on pp. 1, 2, 9–12, 15, 22, 23, 164).

-
- [9] N. Hagel, N. Hili, and D. Schwab. “Turning Low-Code Development Platforms into True No-Code with LLMs”. In: *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*. MODELS Companion ’24. Linz, Austria: Association for Computing Machinery, 2024, pp. 876–885. ISBN: 9798400706226. DOI: [10.1145/3652620.3688334](https://doi.org/10.1145/3652620.3688334). URL: <https://doi.org/10.1145/3652620.3688334> (cit. on pp. 2, 21, 110, 118, 129).
 - [10] C. Tinnes, A. Welter, and S. Apel. “Software Model Evolution with Large Language Models: Experiments on Simulated, Public, and Industrial Datasets”. In: *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering*. IEEE Press, 2025, pp. 950–962. ISBN: 9798331505691. URL: <https://doi.org/10.1109/ICSE55347.2025.00112> (cit. on pp. 2, 17–21, 24, 118, 124, 130–132, 134).
 - [11] K. Peffers et al. “A Design Science Research Methodology for Information Systems Research”. In: *Journal of Management Information Systems* 24.3 (2007-12), pp. 45–77. ISSN: 0742-1222. DOI: [10.2753/MIS0742-1222240302](https://doi.org/10.2753/MIS0742-1222240302). URL: <https://doi.org/10.2753/MIS0742-1222240302> (cit. on p. 3).
 - [12] B. Kitchenham and S. Charters. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Tech. rep. EBSE 2007-001. Version 2.3. Keele University and Durham University Joint Report, 2007 (cit. on pp. 3, 16, 113).
 - [13] D. C. Schmidt. “Guest Editor’s Introduction: Model-Driven Engineering”. In: *Computer* 39.2 (2006-02), pp. 25–31. ISSN: 0018-9162. DOI: [10.1109/MC.2006.58](https://doi.org/10.1109/MC.2006.58). URL: <https://doi.org/10.1109/MC.2006.58> (cit. on pp. 6, 109, 110).
 - [14] S. Sendall and W. Kozaczynski. “Model transformation: the heart and soul of model-driven software development”. In: *IEEE Software* 20.5 (2003), pp. 42–45. DOI: [10.1109/MS.2003.1231150](https://doi.org/10.1109/MS.2003.1231150) (cit. on pp. 6, 110).
 - [15] A. Bucaioni, A. Cicchetti, and F. Ciccozzi. “Modelling in Low-Code Development: A Multi-Vocal Systematic Review”. In: *Software and Systems Modeling* 21 (2022), pp. 1959–1981. DOI: [10.1007/s10270-021-00964-0](https://doi.org/10.1007/s10270-021-00964-0). URL: <https://doi.org/10.1007/s10270-021-00964-0> (cit. on p. 6).
 - [16] N. Prinz, C. Rentrop, and M. Huber. “Low-Code Development Platforms – A Literature Review”. In: *AMCIS 2021 Proceedings*. 2021-08. URL: https://aisel.aisnet.org/amcis2021/adv_info_systems_general_track/adv_info_systems_general_track/2/ (cit. on p. 6).
 - [17] A. C. Bock and U. Frank. “Low-Code Platform”. In: *Business & Information Systems Engineering* 63.6 (2021-12), pp. 733–740. ISSN: 1867-0202. DOI: [10.1007/s12599-021-00726-8](https://doi.org/10.1007/s12599-021-00726-8). URL: <https://doi.org/10.1007/s12599-021-00726-8> (cit. on p. 6).

- [18] Y. Luo et al. "Characteristics and Challenges of Low-Code Development: The Practitioners' Perspective". In: *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ESEM '21. Bari, Italy: Association for Computing Machinery, 2021. ISBN: 9781450386654. DOI: [10.1145/3475716.3475782](https://doi.org/10.1145/3475716.3475782). URL: <https://doi.org/10.1145/3475716.3475782> (cit. on p. 6).
- [19] A. J. Ko et al. "The state of the art in end-user software engineering". In: *ACM Computing Surveys* 43.3 (2011-04). ISSN: 0360-0300. DOI: [10.1145/1922649.1922658](https://doi.org/10.1145/1922649.1922658). URL: <https://doi.org/10.1145/1922649.1922658> (cit. on p. 6).
- [20] M. X. Liu et al. "'What It Wants Me To Say': Bridging the Abstraction Gap Between End-User Programmers and Code-Generating Large Language Models". In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. CHI '23. Hamburg, Germany: Association for Computing Machinery, 2023. ISBN: 9781450394215. DOI: [10.1145/3544548.3580817](https://doi.org/10.1145/3544548.3580817). URL: <https://doi.org/10.1145/3544548.3580817> (cit. on pp. 7, 14, 19, 118, 127, 130, 132).
- [21] J. L. G. Dietz. *Enterprise Ontology: Theory and Methodology*. Berlin, Heidelberg: Springer, 2006. DOI: [10.1007/3-540-33149-2](https://doi.org/10.1007/3-540-33149-2) (cit. on pp. 7, 8).
- [22] J. L. G. Dietz and J. A. P. Hoogervorst. "Enterprise ontology in enterprise engineering". In: *Proceedings of the 2008 ACM Symposium on Applied Computing*. ACM, 2008, pp. 572–579. DOI: [10.1145/1363686.1363824](https://doi.org/10.1145/1363686.1363824) (cit. on p. 8).
- [23] J. W. Yoder and R. E. Johnson. "The Adaptive Object-Model Architectural Style". In: *Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance*. WICSA 3. NLD: Kluwer, B.V., 2002, pp. 3–27. ISBN: 1402071760 (cit. on pp. 8, 15).
- [24] JGraph. *draw.io: Online Diagram Software*. Accessed 28 September 2026. 2024. URL: <https://www.draw.io/> (cit. on p. 11).
- [25] Form.io. *Form.io Documentation*. Accessed 28 September 2026. 2024. URL: <https://help.form.io/> (cit. on p. 11).
- [26] A. Fan et al. "Large Language Models for Software Engineering: Survey and Open Problems". In: *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. 2023, pp. 31–53. DOI: [10.1109/ICSE-FoSE59343.2023.000008](https://doi.org/10.1109/ICSE-FoSE59343.2023.000008) (cit. on pp. 14, 112).
- [27] D. Zan et al. "Large Language Models Meet NL2Code: A Survey". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2023, pp. 7443–7464. DOI: [10.18653/v1/2023.acl-long.411](https://doi.org/10.18653/v1/2023.acl-long.411). URL: <https://aclanthology.org/2023.acl-long.411/> (cit. on p. 14).

-
- [28] A. Vaswani et al. “Attention is all you need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964 (cit. on pp. 14, 111).
- [29] T. B. Brown et al. “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 1877–1901. URL: <https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfc4967418bfb8ac142f64a-Abstract.html> (cit. on p. 14).
- [30] A. Holtzman et al. “The Curious Case of Neural Text Degeneration”. In: *International Conference on Learning Representations*. 2020. URL: <https://openreview.net/forum?id=rygGQyrFvH> (cit. on p. 14).
- [31] A. Wright et al. *JSON Schema: A Media Type for Describing JSON Documents*. Draft 2020-12. 2022. URL: <https://json-schema.org/draft/2020-12/json-schema-core.html> (cit. on p. 14).
- [32] S. Geng et al. “Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, 2023, pp. 10932–10952. DOI: 10.18653/v1/2023.emnlp-main.674. URL: <https://aclanthology.org/2023.emnlp-main.674/> (cit. on p. 14).
- [33] M. Fowler. *Domain-Specific Languages*. Addison-Wesley Professional, 2010. ISBN: 978-0321712943 (cit. on p. 14).
- [34] OutSystems. *Generative AI App Builder: Create Apps in Minutes with Mentor*. Accessed 25 September 2026. 2026. URL: <https://www.outsystems.com/generative-ai-app-builder> (cit. on p. 15).
- [35] Mendix. *Mendix AI Assistance (Maia)*. Accessed 25 September 2026. 2026. URL: <https://docs.mendix.com/refguide/mendix-ai-assistance/> (cit. on p. 15).
- [36] Microsoft. *Build Apps Through Conversation with Copilot*. Accessed 25 September 2026. 2026. URL: <https://learn.microsoft.com/en-us/power-apps/maker/canvas-apps/ai-conversations-create-app> (cit. on p. 15).
- [37] P. Pattnayak and H. Bohra. *Review of Tools for Zero-Code LLM Based Application Development*. 2025. arXiv: 2510.19747. URL: <https://arxiv.org/abs/2510.19747> (cit. on p. 15).
- [38] Y. Wang et al. “Generating SysML Behavior Models via Large Language Models: an Empirical Study”. In: *Proceedings of the 16th International Conference on Internetware*. Internetware ’25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 366–377. ISBN: 9798400719264. DOI: 10.1145/3755881.3755926. URL: <https://doi.org/10.1145/3755881.3755926> (cit. on pp. 17, 20, 21, 24, 112, 118, 124, 127, 130, 131, 133).

- [39] T. Sun et al. “Automated Implementation Scheme for Low-Code Development Based on Large Language Models”. In: *Proceedings of the 2024 4th International Conference on Signal Processing and Communication Technology*. SPCT '24. New York, NY, USA: Association for Computing Machinery, 2025, pp. 237–241. ISBN: 9798400710636. DOI: [10.1145/3712464.3712507](https://doi.org/10.1145/3712464.3712507). URL: <https://doi.org/10.1145/3712464.3712507> (cit. on pp. 17, 19, 21, 117, 124, 129).
- [40] T. Turchi. “FlowPilot: A Generative AI-Driven Visual Language for Computational Thinking Education”. In: *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 2024, pp. 353–355. DOI: [10.1109/VL/HCC60511.2024.00046](https://doi.org/10.1109/VL/HCC60511.2024.00046) (cit. on pp. 17, 117, 126, 131).
- [41] J. Xu et al. “LLM4Workflow: An LLM-based Automated Workflow Model Generation Tool”. In: *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. ASE '24. Sacramento, CA, USA: Association for Computing Machinery, 2024, pp. 2394–2398. ISBN: 9798400712487. DOI: [10.1145/3691620.3695360](https://doi.org/10.1145/3691620.3695360). URL: <https://doi.org/10.1145/3691620.3695360> (cit. on pp. 18, 117, 124, 125).
- [42] E. C. Koh. “Auto-DSM: Using a Large Language Model to generate a Design Structure Matrix”. In: *Natural Language Processing Journal* 9 (2024), p. 100103. ISSN: 2949-7191. DOI: [10.1016/j.nlp.2024.100103](https://doi.org/10.1016/j.nlp.2024.100103). URL: <https://www.sciencedirect.com/science/article/pii/S2949719124000517> (cit. on pp. 18, 19, 117).
- [43] Y. Li et al. “D2: An LLM agent driving end-to-end visual AI modeling in energy platforms”. In: *Energy and AI* 21 (2025), p. 100582. ISSN: 2666-5468. DOI: [10.1016/j.egyai.2025.100582](https://doi.org/10.1016/j.egyai.2025.100582). URL: <https://www.sciencedirect.com/science/article/pii/S2666546825001144> (cit. on pp. 18, 117, 128).
- [44] V. Liventsev et al. “Fully Autonomous Programming with Large Language Models”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '23. Association for Computing Machinery, 2023, pp. 1146–1155. DOI: [10.1145/3583131.3590481](https://doi.org/10.1145/3583131.3590481). URL: <https://doi.org/10.1145/3583131.3590481> (cit. on pp. 18, 20, 21, 111, 118, 125, 130–134).
- [45] A. Grishina et al. “Fully Autonomous Programming Using Iterative Multi-Agent Debugging with Large Language Models”. In: *ACM Transactions on Evolutionary Learning and Optimization* 5.1 (2025-03). ISSN: 2688-299X. DOI: [10.1145/3719351](https://doi.org/10.1145/3719351). URL: <https://doi.org/10.1145/3719351> (cit. on pp. 18, 19, 21, 118, 125, 131–133).
- [46] R. Bavishi et al. “Neurosymbolic repair for low-code formula languages”. In: *Proceedings of the ACM on Programming Languages* 6.OOPSLA2 (2022-10). DOI: [10.1145/3563327](https://doi.org/10.1145/3563327). URL: <https://doi.org/10.1145/3563327> (cit. on pp. 18, 118, 133).

- [47] Z. Zhou et al. “InstructPipe: Generating Visual Blocks Pipelines with Human Instructions and LLMs”. In: *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. CHI '25. New York, NY, USA: Association for Computing Machinery, 2025. ISBN: 9798400713941. DOI: [10.1145/3706598.3713905](https://doi.org/10.1145/3706598.3713905). URL: <https://doi.org/10.1145/3706598.3713905> (cit. on pp. 18, 19, 118, 133, 134).
- [48] T. R. Silva and T. T. Hildebrandt. “Human-Centric Hybrid-AI for No-Code Development”. In: *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. FSE Companion '25. Clarion Hotel Trondheim, Trondheim, Norway: Association for Computing Machinery, 2025, pp. 1520–1521. ISBN: 9798400712760. DOI: [10.1145/3696630.3731669](https://doi.org/10.1145/3696630.3731669). URL: <https://doi.org/10.1145/3696630.3731669> (cit. on pp. 18, 110, 111, 117, 126, 127, 132–134).
- [49] B. Chen et al. “Accurate and Consistent Graph Model Generation from Text with Large Language Models”. In: *2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 2025, pp. 130–141. DOI: [10.1109/MODELS67397.2025.00018](https://doi.org/10.1109/MODELS67397.2025.00018) (cit. on pp. 19, 118, 125, 126, 128).
- [50] L. Netz, J. Reimer, and B. Rumpe. “Using Grammar Masking to Ensure Syntactic Validity in LLM-based Modeling Tasks”. In: *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*. MODELS Companion '24. Linz, Austria: Association for Computing Machinery, 2024, pp. 115–122. ISBN: 9798400706226. DOI: [10.1145/3652620.3687805](https://doi.org/10.1145/3652620.3687805). URL: <https://doi.org/10.1145/3652620.3687805> (cit. on pp. 19, 118, 126, 128, 130, 133).
- [51] N. Malamas et al. “Toward efficient vibe coding: An LLM-based agent for low-code software development”. In: *Journal of Computer Languages* 85 (2025), p. 101367. ISSN: 2590-1184. DOI: [10.1016/j.cola.2025.101367](https://www.sciencedirect.com/science/article/pii/S259011842500053X). URL: <https://www.sciencedirect.com/science/article/pii/S259011842500053X> (cit. on pp. 21, 118, 129, 130).
- [52] W. Zhang et al. “Large Language Models in Model-Driven Engineering: A Systematic Mapping Study”. In: *Empirical Software Engineering* 32, 3 (2027). DOI: [10.1007/s10664-026-10921-4](https://doi.org/10.1007/s10664-026-10921-4) (cit. on p. 21).
- [53] A. Jedlitschka, M. Ciolkowski, and D. Pfahl. “Reporting Experiments in Software Engineering”. In: *Guide to Advanced Empirical Software Engineering*. Ed. by F. Shull, J. Singer, and D. I. K. Sjøberg. Springer, 2008, pp. 201–228 (cit. on pp. 64, 70, 93).
- [54] International Organization for Standardization. *Ergonomics of Human-System Interaction—Part 11: Usability: Definitions and Concepts*. ISO 9241-11:2018. 2018. URL: <https://www.iso.org/standard/63500.html> (cit. on p. 70).
- [55] J. Brooke. “SUS: A “Quick and Dirty” Usability Scale”. In: *Usability Evaluation in Industry*. Ed. by P. W. Jordan et al. Taylor & Francis, 1996, pp. 189–194 (cit. on pp. 73, 142).

- [56] A. I. Martins et al. “European Portuguese Validation of the System Usability Scale (SUS)”. In: *Procedia Computer Science* 67 (2015), pp. 293–300. DOI: [10.1016/j.procs.2015.09.273](https://doi.org/10.1016/j.procs.2015.09.273). URL: <https://doi.org/10.1016/j.procs.2015.09.273> (cit. on pp. 73, 142).
- [57] S. G. Hart and L. E. Staveland. “Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research”. In: *Human Mental Workload*. Ed. by P. A. Hancock and N. Meshkati. Vol. 52. Advances in Psychology. North-Holland, 1988, pp. 139–183. DOI: [10.1016/S0166-4115\(08\)62386-9](https://doi.org/10.1016/S0166-4115(08)62386-9) (cit. on pp. 73, 141).
- [58] A. Bangor, P. T. Kortum, and J. T. Miller. “An Empirical Evaluation of the System Usability Scale”. In: *International Journal of Human–Computer Interaction* 24.6 (2008), pp. 574–594. DOI: [10.1080/10447310802205776](https://doi.org/10.1080/10447310802205776). URL: <https://doi.org/10.1080/10447310802205776> (cit. on p. 74).
- [59] A. Sahay et al. “Supporting the understanding and comparison of low-code development platforms”. In: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2020, pp. 171–178. DOI: [10.1109/SEAA51224.2020.00036](https://doi.org/10.1109/SEAA51224.2020.00036) (cit. on pp. 109, 110).
- [60] M. Brambilla, J. Cabot, and M. Wimmer. *Model-Driven Software Engineering in Practice*. 2nd ed. Synthesis Lectures on Software Engineering. Cham: Springer, 2017. DOI: [10.1007/978-3-031-02549-5](https://doi.org/10.1007/978-3-031-02549-5) (cit. on p. 109).
- [61] X. Hou et al. “Large Language Models for Software Engineering: A Systematic Literature Review”. In: *ACM Transactions on Software Engineering and Methodology* 33.8 (2024-12). ISSN: 1049-331X. DOI: [10.1145/3695988](https://doi.org/10.1145/3695988). URL: <https://doi.org/10.1145/3695988> (cit. on pp. 109, 112).
- [62] L. Burgueño et al. “Automation in Model-Driven Engineering: A Look Back, and Ahead”. In: *ACM Transactions on Software Engineering and Methodology* 34.5 (2025-05). ISSN: 1049-331X. DOI: [10.1145/3712008](https://doi.org/10.1145/3712008). URL: <https://doi.org/10.1145/3712008> (cit. on pp. 109, 110, 113, 117).
- [63] J. Cabot. “Positioning of the low-code movement within the field of model-driven engineering”. In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS ’20. Virtual Event, Canada: Association for Computing Machinery, 2020. ISBN: 9781450381352. DOI: [10.1145/3417990.3420210](https://doi.org/10.1145/3417990.3420210). URL: <https://doi.org/10.1145/3417990.3420210> (cit. on p. 110).
- [64] K. Czarnecki and S. Helsen. “Feature-based survey of model transformation approaches”. In: *IBM Systems Journal* 45.3 (2006-07), pp. 621–645. ISSN: 0018-8670. DOI: [10.1147/sj.453.0621](https://doi.org/10.1147/sj.453.0621). URL: <https://doi.org/10.1147/sj.453.0621> (cit. on p. 110).

- [65] T. Buchmann, R. Peinl, and F. Schwägerl. “White-box LLM-supported Low-code Engineering: A Vision and First Insights”. In: *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems. MODELS Companion '24*. Linz, Austria: Association for Computing Machinery, 2024, pp. 556–560. ISBN: 9798400706226. DOI: [10.1145/3652620.3687803](https://doi.org/10.1145/3652620.3687803). URL: <https://doi.org/10.1145/3652620.3687803> (cit. on pp. 111, 118).
- [66] M. Wang et al. “Exploring Influence Factors on LLM Suitability for No-Code Development of End User Applications”. In: *Software: Practice and Experience* 56.1 (2026), pp. 96–118. DOI: [10.1002/spe.70027](https://doi.org/10.1002/spe.70027) (cit. on pp. 111, 118).
- [67] M. Chen et al. “Evaluating Large Language Models Trained on Code”. In: *arXiv preprint arXiv:2107.03374* (2021). URL: <https://arxiv.org/abs/2107.03374> (cit. on p. 111).
- [68] J. Martins, F. Branco, and H. Mamede. “Combining low-code development with ChatGPT to novel no-code approaches: A focus-group study”. In: *Intelligent Systems with Applications* 20 (2023), p. 200289. ISSN: 2667-3053. DOI: [10.1016/j.iswa.2023.200289](https://doi.org/10.1016/j.iswa.2023.200289) (cit. on pp. 112, 117, 130).
- [69] H. Naveed et al. “Model driven engineering for machine learning components: A systematic literature review”. In: *Information and Software Technology* 169, 107423 (2024). DOI: [10.1016/j.infsof.2024.107423](https://doi.org/10.1016/j.infsof.2024.107423). URL: <https://doi.org/10.1016/j.infsof.2024.107423> (cit. on p. 112).
- [70] A. Bucchiarone et al. “Grand challenges in model-driven engineering: an analysis of the state of the research”. In: *Software and Systems Modeling* 19.1 (2020-01), pp. 5–13. ISSN: 1619-1366. DOI: [10.1007/s10270-019-00773-6](https://doi.org/10.1007/s10270-019-00773-6). URL: <https://doi.org/10.1007/s10270-019-00773-6> (cit. on pp. 112, 113).
- [71] J. Di Rocco et al. “On the use of large language models in model-driven engineering”. In: *Software and Systems Modeling* 24.3 (2025), pp. 923–948. DOI: [10.1007/s10270-025-01263-8](https://doi.org/10.1007/s10270-025-01263-8) (cit. on p. 113).
- [72] N. Rao et al. “AI for Low-Code for AI”. In: *Proceedings of the 29th International Conference on Intelligent User Interfaces. IUI '24*. Greenville, SC, USA: Association for Computing Machinery, 2024, pp. 837–852. ISBN: 9798400705083. DOI: [10.1145/3640543.3645203](https://doi.org/10.1145/3640543.3645203). URL: <https://doi.org/10.1145/3640543.3645203> (cit. on pp. 117, 130).
- [73] S.-K. Wang et al. “ChatOps for microservice systems: A low-code approach using service composition and large language models”. In: *Future Generation Computer Systems* 161.C (2024-12), pp. 518–530. ISSN: 0167-739X. DOI: [10.1016/j.future.2024.07.029](https://doi.org/10.1016/j.future.2024.07.029). URL: <https://doi.org/10.1016/j.future.2024.07.029> (cit. on pp. 117, 125).

- [74] Y. Fang and Z. Fan. "Complex Flowchart Text Generation Based on Large Language Models". In: *2025 5th International Conference on Neural Networks, Information and Communication Engineering (NNICE)*. 2025, pp. 737–742. DOI: [10.1109/NNICE64954.2025.11064395](https://doi.org/10.1109/NNICE64954.2025.11064395) (cit. on p. 117).
- [75] N. De Troyer, R. De Croon, and K. Verbert. "Enhancing Accessibility for Non-experts in Low-Code/No-Code Through the Addition of an AI Chatbot". In: *Human-Computer Interaction – INTERACT 2025*. Vol. 16108. Lecture Notes in Computer Science. Cham: Springer, 2026, pp. 203–212. DOI: [10.1007/978-3-032-04999-5_12](https://doi.org/10.1007/978-3-032-04999-5_12) (cit. on p. 117).
- [76] S. Mowzoon and M. Faustini. "Introducing LLMScript: A Turing Complete Prompt Based Scripting Language for LLMs with No External Coding Required". In: *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*. 2024, pp. 116–124. DOI: [10.1109/FLLM63129.2024.10852477](https://doi.org/10.1109/FLLM63129.2024.10852477) (cit. on p. 117).
- [77] A. Nour Eldin et al. "Low-code solutions for business process dataflows: From modeling to execution". In: *Information Systems* 135 (2026), p. 102577. ISSN: 0306-4379. DOI: [10.1016/j.is.2025.102577](https://doi.org/10.1016/j.is.2025.102577). URL: <https://www.sciencedirect.com/science/article/pii/S0306437925000614> (cit. on pp. 117, 126).
- [78] Y. Cheng et al. "Prompt Sapper: A LLM-Empowered Production Tool for Building AI Chains". In: *ACM Transactions on Software Engineering and Methodology* 33.5 (2024-06). ISSN: 1049-331X. DOI: [10.1145/3638247](https://doi.org/10.1145/3638247). URL: <https://doi.org/10.1145/3638247> (cit. on p. 117).
- [79] F. C. Kunze and A. Fay. "Automated Generation of AML Models for Industrial Plants Using LLM Chat Applications". In: *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2024, pp. 1–4. DOI: [10.1109/ETFA61755.2024.10710654](https://doi.org/10.1109/ETFA61755.2024.10710654) (cit. on pp. 118, 126).
- [80] K. Deng et al. "Power Grid Model Generation Based on the Tool-Augmented Large Language Model". In: *IEEE Transactions on Power Systems* 40.6 (2025), pp. 5487–5490. DOI: [10.1109/TPWRS.2025.3609049](https://doi.org/10.1109/TPWRS.2025.3609049) (cit. on pp. 118, 127).
- [81] F. He et al. "MetaCloze: a schema-guided automated building metadata model generation system via information extraction". In: *Knowledge-Based Systems* 330.Part B (2025), p. 114596. ISSN: 0950-7051. DOI: [10.1016/j.knosys.2025.114596](https://doi.org/10.1016/j.knosys.2025.114596). URL: <https://www.sciencedirect.com/science/article/pii/S0950705125016351> (cit. on p. 118).
- [82] S. Barat et al. "Constructing Enterprise Digital Twins by Augmenting LLMs with MDE". In: *Proceedings of the 18th Innovations in Software Engineering Conference. ISEC '25*. New York, NY, USA: Association for Computing Machinery, 2025. ISBN: 9798400714245. DOI: [10.1145/3717383.3717391](https://doi.org/10.1145/3717383.3717391). URL: <https://doi.org/10.1145/3717383.3717391> (cit. on pp. 118, 125, 134).

-
- [83] J. N. Kreikemeyer et al. "Using (Not-so) Large Language Models to Generate Simulation Models in a Formal DSL: A Study on Reaction Networks". In: *ACM Transactions on Modeling and Computer Simulation* 35.4 (2025-09). ISSN: 1049-3301. DOI: [10.1145/3733719](https://doi.org/10.1145/3733719). URL: <https://doi.org/10.1145/3733719> (cit. on p. 118).
- [84] J. Zhang and I. Arawjo. "ChainBuddy: An AI-assisted Agent System for Generating LLM Pipelines". In: *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. CHI '25. New York, NY, USA: Association for Computing Machinery, 2025. ISBN: 9798400713941. DOI: [10.1145/3706598.3714085](https://doi.org/10.1145/3706598.3714085). URL: <https://doi.org/10.1145/3706598.3714085> (cit. on pp. 118, 134).
- [85] D. Vella Zarb, G. Parks, and T. Kipouros. "Program Synthesis with LLM-Predicted Minimal Specialized Grammars". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '25. NH Malaga Hotel, Malaga, Spain: Association for Computing Machinery, 2025, pp. 1072–1080. ISBN: 9798400714658. DOI: [10.1145/3712256.3726430](https://doi.org/10.1145/3712256.3726430). URL: <https://doi.org/10.1145/3712256.3726430> (cit. on pp. 118, 127).
- [86] P. Lewis et al. "Retrieval-augmented generation for knowledge-intensive NLP tasks". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS '20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546 (cit. on p. 132).
- [87] B. Shneiderman. "Human-centered AI: ensuring human control while increasing automation". In: *Proceedings of the 5th Workshop on Human Factors in Hypertext*. HUMAN '22. Barcelona, Spain: Association for Computing Machinery, 2022. ISBN: 9781450394017. DOI: [10.1145/3538882.3542790](https://doi.org/10.1145/3538882.3542790). URL: <https://doi.org/10.1145/3538882.3542790> (cit. on p. 132).
- [88] S. Amershi et al. "Guidelines for Human-AI Interaction". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19. Glasgow, Scotland Uk: Association for Computing Machinery, 2019, pp. 1–13. ISBN: 9781450359702. DOI: [10.1145/3290605.3300233](https://doi.org/10.1145/3290605.3300233). URL: <https://doi.org/10.1145/3290605.3300233> (cit. on p. 133).
- [89] E. M. Bender et al. "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '21. Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 610–623. ISBN: 9781450383097. DOI: [10.1145/3442188.3445922](https://doi.org/10.1145/3442188.3445922). URL: <https://doi.org/10.1145/3442188.3445922> (cit. on p. 133).