

Towards LLM-Based Context Engineering for Model-Driven Vibe-DevOps-Engineering of Cross-Platform CI/CD Pipeline Support

Uldis Karlovs-Karlovskis
0009-0008-7816-3770
Riga Technical University
Latvia
Uldis.Karlovs-Karlovskis@rtu.lv

Abstract—Existing model-driven DevOps automation either depends on handcrafted transformations that do not scale to new CI/CD platforms or on prompt-based LLM generation that lacks determinism and reliability for production. This study introduces a proof of concept for *vibe* DevOps engineering, where LLM-based context engineering synthesizes model-driven transformation chains for CI/CD pipelines from existing MDE artifacts and platform documentation. Metamodels, ATL rules, Acceleo templates, and CI/CD specifications are decomposed and organized to prepare the ground for a future Graph-RAG and multi-agent architecture that will derive platform-specific DevOps artifacts from a platform-independent DevOps metamodel. In the current phase, the roles of the envisioned agents are emulated by a human researcher executing the chain of M2M2M2C steps and Eclipse-based runtime testing. Cross-platform constraint transfer reduces total refinement rounds from 11 (GitLab CI/CD) to 6 (Atlassian Bamboo) to 5 (Azure DevOps) across successive runs, and assess syntactic correctness, semantic alignment, and executability, indicating potential and current limitations of *vibe* DevOps engineering.

Index Terms—model-driven engineering, CI/CD, LLM, context engineering, ATL, Acceleo, RAG, multi-agent systems, *vibe* DevOps engineering, *vibe*, DevOps

I. INTRODUCTION

SOFTWARE production is accelerating at a rate with no precedent in the industry's history [1]. The global enterprise software market has continued to expand, driven by cloud adoption, generative AI, and long-running digitalization initiatives, and was estimated by Gartner to reach 900 billion USD in 2024 with an annual growth rate of 11.9%¹. As software systems mature, they require structured DevOps processes to sustain delivery frequency and product quality [2].

The Continuous Integration and Continuous Delivery (CI/CD) pipeline is the operational core of any DevOps process, automating the build, test, and deployment stages. Recent industry data indicate that the CI/CD tooling ecosystem has become more heterogeneous: a 2025 JetBrains survey reports that 32% of organizations use two CI/CD tools in parallel and 9% use at least three [3], often due to client constraints, inherited environments, or financial considerations rather than a unified tooling strategy. Prior work shows that

moving from one CI/CD platform to another, or adding a new platform to an existing toolchain, tends to introduce significant cost and operational disturbance [4].

Model-Driven Engineering (MDE) addresses software complexity by placing abstract models at the center of the development process rather than implementation artifacts [5]. A metamodel in MDE can formally describe any domain, including DevOps workflows and CI/CD pipelines. A platform-independent metamodel (PIM) for generic CI/CD pipelines has been proposed [6], from which platform-specific models (PSM) can be derived for individual CI/CD platforms [7].

The concept of stepwise model transformation from platform-independent to platform-specific models and finally to executable artifacts is a foundational principle of Model-Driven Architecture (MDA) [8]. This transformation chain, from software architecture model through PIM and PSM to executable pipeline code, enables systematic cross-platform pipeline generation. The practical barrier is the construction of the chain itself. Using ECORE metamodels, Atlas Transformation Language (ATL) rules, and Acceleo code generation templates for development demands deep MDE expertise, and extending the chain to a new platform requires that expertise to be applied from scratch [9].

Large language models (LLMs) have become widely adopted across software engineering [10], [11], with demonstrated capacity to generate CI/CD pipeline configurations from natural language prompts. However, LLM output is non-deterministic and subject to syntactic and semantic errors. CI/CD pipelines, as executable artifacts governing production delivery, must be correct and reproducible. These two properties are in direct tension. The proposed approach resolves this tension by applying LLM-based context engineering to generate the MDE transformation chain itself rather than pipeline code directly. Grounding generation in existing metamodels, transformation rules, and platform documentation produces deterministic MDE artifacts from which pipeline code is then generated by the existing MDDOAI [12] engine. This constitutes what the author terms as the *vibe DevOps engineering*: the application of context-engineered LLM generation to formal MDE artifacts in the DevOps domain, extending the

¹<https://www.gartner.com/en/documents/6648534> Accessed March 2026

concept of *vibe coding*² and its model-oriented counterpart, *vibe modeling* [13], to a domain that demands deterministic, verifiable outputs.

Hypothesis: If LLM-based context engineering is applied using examples of existing DevOps model-driven transformation artifacts (metamodels, transformation rules, and code generation templates) together with target CI/CD platform documentation, then it is possible to iteratively generate a complete and correct transformation chain for CI/CD pipeline scenarios.

Research Question (RQ): To what extent can LLM-based context engineering generate correct model-driven DevOps artifacts for a new CI/CD platform?

The proof-of-concept in this study follows the Design Science Research (DSR) methodology [14], treating the transformation chain as the designed artefact and using evaluate-and-refine cycles to reach a correct result. This work builds on two earlier systems: [12], which introduced MDDOAI, a model-to-code solution using ATL and Acceleo to automate CI/CD pipeline synthesis, and [15], which extended MDDOAI with LLM code generation. Neither have addressed multi-agent orchestration, Graph Retrieval Augmented Generation (Graph-RAG) grounding, nor cross-platform transformation chain synthesis. Those are the novel contributions of this work.

The remainder of this paper is organized as follows. Section II surveys related work and identifies the gap this paper fills. Section III describes the MDDOAI multi-agent architecture and its context engineering strategy. Section IV details the experimental protocol. Section V reports results and discusses threats to validity. Section VI concludes and outlines future work.

II. RELATED WORK

This section examines prior work across four areas to establish that no existing work combines model-driven CI/CD transformation chains, LLM-based context engineering, and multi-agent architecture for MDE artifact generation within a single system.

A. Model-Driven CI/CD Pipeline Generation

Model-driven approaches to CI/CD pipeline generation have been explored through platform-independent metamodels, ATL model-to-model transformations, and Acceleo model-to-code templates. Flores et al. proposed the ACICDTrip PIM for the ultimate DevOps PIM metamodel as a platform-neutral representation of CI/CD pipeline structure, from which PSM transformations can be derived for specific platforms [7]. These artifacts are handcrafted: each new platform requires a new adaptation of a metamodel, a set of ATL rules, and Acceleo templates, developed by practitioners with deep MDE expertise [9]. Scalability to new platforms without that expertise, and without manual effort, is not addressed by any existing work in this category.

The author's own prior work constitutes the most direct predecessor to the present paper. Karlovs-Karlovs et al. [12]

introduced MDDOAI, a model-to-code solution that uses ATL and Acceleo to synthesize CI/CD pipeline configurations from software architecture models. MDDOAI targeted a single CI/CD platform and did not address automated generation of MDE artifacts for new platforms. Later it is extended with LLM generation of pipeline code [15], but that work did not incorporate multi-agent architecture, Graph-RAG retrieval, or cross-platform transformation chain synthesis.

B. LLM-Assisted Model-Based Engineering

Karpathy coined the term *vibe coding*³ to describe informal, intent-driven software generation through conversational LLMs, where the practitioner accepts generated output without inspecting intermediate artifacts. The subsequent term *vibe modeling*, introduced by Cabot, applies a comparable philosophy to formal model construction: a user describes a system in natural language, and an LLM-backed agent proposes a model that can be refined through conversation. *Vibe modeling* lowers the barrier to MDE participation by replacing manual model authoring with natural language intent. The BESSER project implements one concrete instantiation of *vibe engineering*, supporting Artificial Intelligence assisted (AI-assisted) construction of domain models from which application code is generated through deterministic rule-based generators [13]. Other research has explored LLM-assisted metamodel authoring and Domain Specific Language (DSL) construction for general domains [16], but these efforts remain at the model and metamodel level and do not address transformation chain synthesis or domain-specific CI/CD generation.

This study proposes *vibe DevOps engineering* as a specialization distinct from both *vibe coding* and *vibe engineering* in three aspects: 1) the artifact target is the MDE transformation chain, comprising ECORE metamodels, ATL rules, and Acceleo templates, rather than application code or domain models, 2) the domain scope is restricted to CI/CD pipeline generation, which imposes specific structural and syntactic correctness criteria on generated artifacts, 3) the generation process uses structured, iterative context engineering grounded in existing artifacts and platform documentation, not open-ended conversational prompting.

C. LLM-Based Artifact Generation and Context Engineering

LLM-based code generation has been studied extensively since the release of Codex [17] and later models. LLMs perform well on standard programming benchmarks, but direct prompting produces non-deterministic output that often fails to meet the structural and semantic requirements of formal MDE artefacts such as Ecore metamodels and ATL transformations [18].

Context engineering, meaning the intentional construction of prompts that include examples, constraints, and retrieved reference material, substantially improves LLM reliability for structured generation tasks [19]. Retrieval-Augmented Generation (RAG) and its graph-based variant Graph-RAG [20]

²<https://karpathy.bearblog.dev/year-in-review-2025/> Accessed March 2026

³<https://karpathy.bearblog.dev/year-in-review-2025/> Accessed March 2026

extend this by grounding generation in a structured knowledge graph that captures relationships between artefacts. No existing work applies Graph-RAG to MDE transformation chain synthesis for CI/CD platforms [18], where the knowledge graph must encode relationships between PIM concepts, PSM elements, ATL rule structure, and platform documentation.

D. Multi-Agent LLM Frameworks in Software Engineering

Multi-agent LLM systems have been applied across a range of software engineering tasks. MetaGPT assigns roles such as product manager, architect, and engineer to distinct agents and coordinates them through Standardized Operating Procedures to produce software artifacts from a high-level requirement [21].

Multi-agent LLM frameworks such as AutoGen⁴ and CrewAI⁵ show that dividing software engineering subtasks across specialised agents improves both output quality and traceability compared to single-agent approaches.

For now, no existing work has been found where multi-agent framework assigns roles that correspond to MDE-specific artifact types. To the best of the author's knowledge, existing multi-agent systems do not include agents whose responsibilities align with Ecore metamodel definition, ATL transformation development, or Acceleo template construction, and none of the surveyed frameworks realises the model-to-model-to-code (M2M2C) transformation chain as a coordinated multi-agent workflow.

E. Summary of the Gap

The literature covers MDE for CI/CD, LLM-based context engineering, and multi-agent software engineering. Each of these areas has been studied in isolation. Model-driven engineering approaches provide strong abstraction and formal transformation capabilities, but are often limited by high-level specifications, incomplete methodological support, and a lack of end-to-end automation and adaptability in evolving environments. LLM-based context engineering enables flexible synthesis of artifacts from natural language, but is constrained by limited context windows, lack of persistent state, and risks of inconsistency or hallucinated outputs when generating complex, multi-step artifacts. Multi-agent software engineering introduces coordination, decomposition, and iterative refinement capabilities, yet current systems struggle with reliability, error propagation, tool integration, and consistent production of correct executable artifacts. These limitations suggest that each technique alone addresses only part of the problem. Combining structured MDE transformations with LLM-driven context synthesis and coordinated multi-agent reasoning can enable both formal consistency and adaptive generation, while improving traceability, scalability, and robustness of the overall pipeline generation process. However, no existing research brings together a structured multi-agent architecture with Graph-RAG grounding to automate the synthesis of complete,

cross-platform MDE transformation chains for CI/CD pipeline generation. This paper fills that gap.

III. SOLUTION DESIGN

MDDOAI is a model-driven system for CI/CD pipeline generation that has been incrementally extended with LLM-based capabilities. In its current form, the architecture applies what is referred to in this paper as *vibe DevOps engineering*, where context-engineered LLMs are used to derive MDE artifacts for a new CI/CD platform, specifically Ecore metamodels, ATL transformation rules, and Acceleo code generation templates, and the resulting artifacts then serve as the basis for deterministic pipeline code generation. The following subsections present the solution from three distinct viewpoints.

A. High-Level User Workflow

The user interacts with MDDOAI in two distinct phases, illustrated in Fig. 1. In Phase 1, the user provides the Uniform Resource Locator (URL) of a target CI/CD platform's documentation and states the integration goal. MDDOAI retrieves and processes this documentation, maps its structure to the existing MDE knowledge base, and generates three artifact types: a PSM ECORE metamodel extension, ATL transformation rules from the PIM, and Acceleo code generation templates. The generated artifacts are validated against automated test case patterns. If any test fails, generation is restarted with an updated prompt that incorporates the failure as an additional constraint. When all tests pass, the integration is confirmed and the new platform is made available for pipeline generation. This loop realizes the DSR evaluate-and-refine cycle [14] at the artifact level.

In Phase 2, the user specifies a pipeline generation goal for a concrete software project, for example generating CI/CD pipeline configurations for an application composed of a FastAPI backend and a React frontend. MDDOAI applies the platform-specific MDE artifacts produced in Phase 1 to generate executable pipeline configurations. The user receives these configurations without any additional manual authoring of transformation rules.

Phase 3 denotes direct use of the generated pipeline artifacts in the target CI/CD environment. Beyond initial deployment, MDDOAI can be integrated into an existing master pipeline as an upstream stage, enabling automated pipeline regeneration whenever platform requirements change or new components are introduced. This closes the automation loop: rather than manually re-authoring transformation rules, the system re-executes the MDE chain and redeploys updated configurations automatically.

B. High-level Solution Architecture

The MDDOAI architecture is structured in layers, as illustrated in Fig. 2. At the top, a Sirius-based graphical editor serves as the interface for defining software architecture models. Beneath it, an AI agent layer groups agents with responsibilities for orchestration, retrieval, generation, validation, and Graph-RAG coordination. The LLM-based context

⁴<https://microsoft.github.io/autogen/stable/index.html> Accessed March 2026

⁵<https://docs.crewai.com/> Accessed March 2026

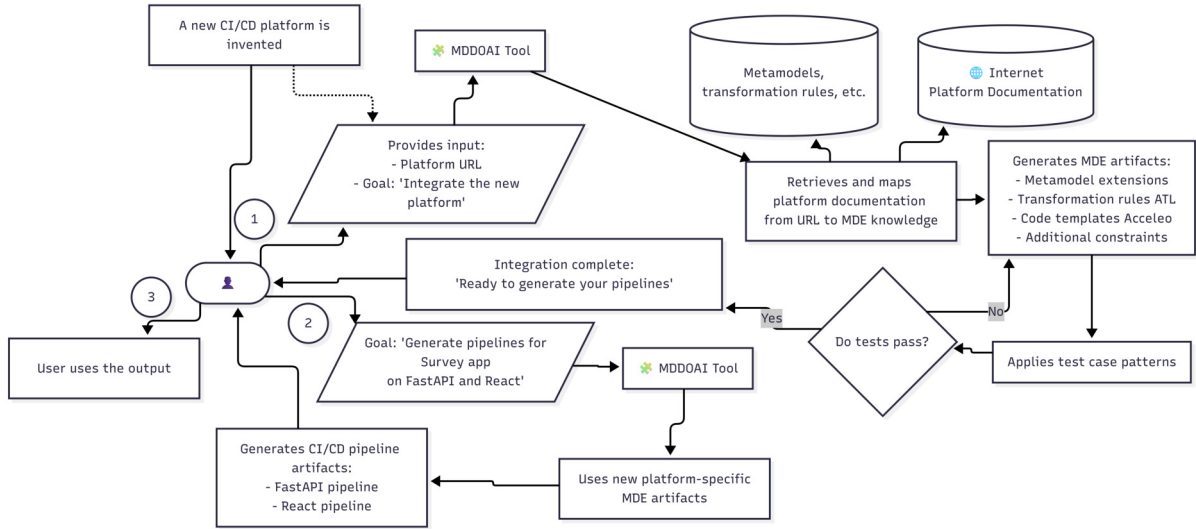


Fig. 1. MDDOAI high-level user workflow across three phases.

engineering layer lies between the agents and the artifact stores and provides mechanisms for assembling prompts, injecting context, and collecting constraints. Two parallel layers follow: a models layer that holds software architecture models, PIM instances, PSM instances, and CI/CD pipeline models, and a transformation layer that runs ATL and Aceleo, complemented by Python utilities. Below these, the metamodel layer defines the type systems for all model classes, and the lowest layer comprises the Eclipse IDE and the associated build toolchain. This end-to-end flow corresponds to the concept referred to as *vibe DevOps engineering*.

On the right-hand side of the architecture, a vertical block represents the generated outputs for the supported CI/CD platforms, and the agent layer is invoked to validate these artifacts and register them in the Graph-RAG knowledge base for use in later generation runs.

C. Low-Level Technical Architecture

The multi-agent architecture governing LLM-based context engineering in MDDOAI is shown in Fig. 3. Two inputs enter the system: a user prompt describing the target platform or feature to be integrated, and a URL pointing to the platform's documentation. A Chat Interface agent collects task details from the user and passes them to the Orchestrator agent.

The Orchestrator decomposes the request into a sequential plan of one-feature-at-a-time generation steps. Sequential execution is required by the monolithic ECORE metamodel architecture, which prohibits safe concurrent modification. The Orchestrator coordinates all downstream agents, tracks progress, and requests additional input from the Chat Interface agent when ambiguities arise during execution.

A Retrieval agent fetches content from the provided URL or document and passes it to a Documentation Parser agent. The Parser maps raw text onto the structure of the relevant metamodels and distributes the result to all six domain expert

agents: SWArch, PIM, PSM, CI/CD Pipeline, ATL, and Aceleo. For visual clarity, the architecture diagram omits three of these connections; the full routing is described here. The six agents are organized into three functional groups. The first group contains the SWArch agent, which handles the software architecture model and provides the structural foundation for all downstream agents. The second group covers the model-driven layer: the PIM agent operates on the platform-independent model, and the PSM agent refines that representation into platform-specific constructs. The third group covers artifact generation: the CI/CD Pipeline agent maps PSM constructs onto platform-specific pipeline structures; the ATL agent produces model-to-model transformation rules; and the Aceleo agent produces model-to-code generation templates. Each of the six agents retrieves grounding examples from the Graph-RAG knowledge base via the RAG agent [20], and the RAG agent updates the knowledge base when new validated artifacts are added.

The Generation agent drives the artifact creation loop. For a given platform feature, it invokes all six expert agents in sequence, each producing the MDE artifact for its domain. The SWArch agent generates the structural foundation artifact first. The PIM and PSM agents then produce their respective model-layer artifacts. The CI/CD Pipeline agent maps PSM constructs onto platform-specific pipeline structures, the ATL agent produces model-to-model transformation rules, and the Aceleo agent produces model-to-code generation templates. Each agent retrieves grounding examples via the RAG agent, which updates the knowledge base as new validated artifacts are added.

Three validation mechanisms are applied within this loop:

- An LLM-as-a-Judge component performs an LLM-based syntactic and semantic assessment of each generated artifact [22].
- An Integration Test agent generates a PSM model in-

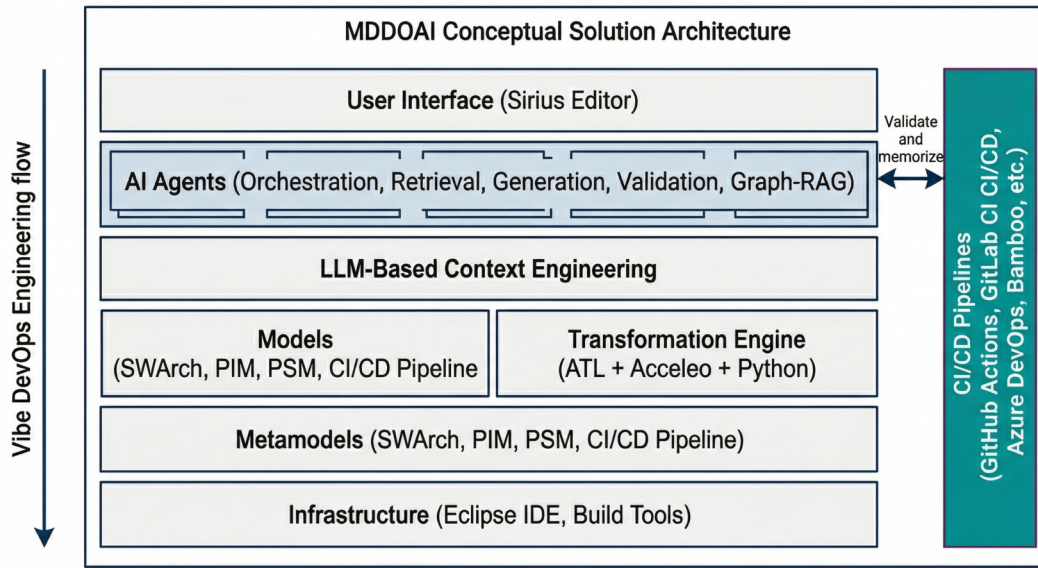


Fig. 2. Layered solution architecture of MDDOAI.

stance from the produced PSM change and executes in-memory tests against it.

- A Validation agent invokes Eclipse via the command line, applies the generated MDE artifacts through the MD-DOAI pipeline, and verifies that a pipeline configuration file is produced without errors.

If any validation step fails, a Prompt Improvement agent analyzes the error, formulates a revised prompt with a new targeted constraint, and returns control to the Generation agent.

When the generation is complete, the Grounding agent decomposes the finished CI/CD feature into its constituent MDE artifacts and instructs the RAG agent to store them in the Graph-RAG knowledge base, making them available as context for future generation runs.

Section IV evaluates this chain by applying it to three CI/CD platforms of decreasing documentation coverage.

IV. EXPERIMENTS WITH THE LLM-BASED SOLUTION

This section evaluates the three-step artifact generation chain described in Section III across three CI/CD platforms, following the DSR evaluate-and-refine cycle [14]. All experiments were conducted manually by a human researcher, with LLM outputs reviewed and validated at each stage of the artifact generation process. The experimental conditions are established first, covering platform selection and input context composition. The generation procedure and cross-platform constraint transfer mechanism are then described, followed by the evaluation instruments used to assess convergence cost and artifact correctness.

A. Experimental Setup

The subsections explain the rationale for choosing the target platforms and the fixed-context strategy used for composing LLM inputs in all runs.

Platform selection. Three CI/CD platforms were chosen such that they fall into different tiers of publicly available documentation, approximated here by the number of Stack Overflow questions associated with their tags, as a proxy for LLM training data exposure⁶. Run 1 targets GitLab CI as a high-coverage baseline [3]. Run 2 targets Atlassian Bamboo, an enterprise-niche platform with substantially lower coverage. Run 3 targets Azure DevOps Pipelines, which has a distinct agent-pool model and typed task-plugin syntax, serving as validation that constraint transfer holds across structurally different platforms.

Fixed context. The input context supplied to the LLM at each step is intentionally composed from a fixed set of artifacts:

- Step 1 receives the existing GitHub Actions PSM meta-model (GHA.ecore) as master example, which provides PSM structural conventions and modeling patterns, and the target platform documentation, which defines platform-specific concepts and relationships.
- Step 2 receives pimMM.ecore, which defines the ATL source classes for rule left-hand sides; ccd2gha.atl, which provides ATL syntax, helper style, and mapping pattern examples; and the Step 1 metamodel, which defines the target classes ATL rules must produce.
- Step 3 receives the Step 1 metamodel, which defines the classes and attributes queried by the template; gha_generate.mtl, which provides module structure, query syntax, and file block patterns; and the target platform documentation, which defines the YAML schema and required output structure.

Each context piece has a single defined role: no piece is redundant, and no step receives context that does not serve

⁶<https://stackoverflow.com/questions> Accessed March 2026

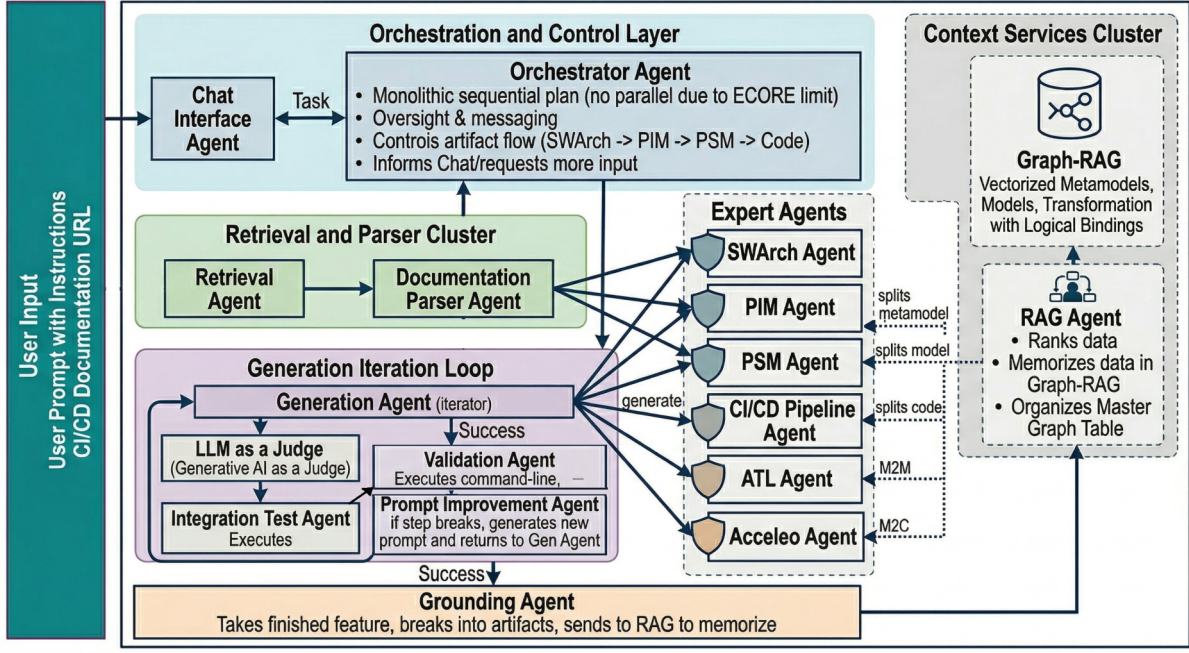


Fig. 3. Multi-agent technical architecture for LLM-based context engineering.

a specific grounding purpose.

Reproducibility. The complete experimental trail is publicly available in the project repository.⁷ The repository is organised by CI/CD platform and by step within each platform. For each refinement round, the repository contains: the full prompt submitted to the LLM, the generated artifact (.ecore, .atl, or .mtl), and a round note recording the identified root cause and the exact constraint appended to the subsequent prompt. The generated MDE artifacts for all three platforms are stored alongside the YAML pipeline outputs produced by applying the completed chain to each of the three test cases, allowing independent verification of the transformation artifacts and the end-to-end pipeline generation results.

B. Artifact Generation Procedure

The procedure is uniform across all three runs and consists of a fixed three-step artifact chain, an iterative refinement protocol, and a cross-platform constraint transfer mechanism applied between runs.

Three-step artifact chain. Table I defines the three generation steps, their inputs, and the pass criterion applied at each step.

Iterative refinement protocol. The refinement loop is the experimental mechanism.

- 1) Assemble the prompt: fixed base context plus all constraints accumulated from prior rounds.
- 2) Submit to the LLM and record the generated artifact.
- 3) Apply the pass criterion: if met, the step is complete.

- 4) If not met: identify the root cause, formulate exactly one constraint targeting it, append it to the prompt, increment the round counter, and return to step 1.

No manual edits are made to the generated artifact. The one-constraint-per-round rule isolates the effect of each corrective instruction. The round count per step per platform is the primary measured outcome.

Cross-platform constraint transfer. All constraints accumulated during Run 1 are pre-loaded into the corresponding steps of Run 2, and likewise from Run 1 and Run 2 into Run 3. This tests the central claim: prior constraint knowledge reduces the rounds required on subsequent platforms.

C. Evaluation Design

Two complementary measures are used to assess the chain output: round count, which captures the cost of convergence, and concept coverage, which captures the correctness of the generated artifacts against the DevOps PIM concept set. Both measures concern the correctness of the output; artifact quality dimensions such as maintainability are addressed in the discussion rather than measured here.

Test cases. Three pipelines are used across all runs. A three-job pipeline and a four-job pipeline confirm the chain produces valid output on minimal and naturally-occurring inputs respectively. The third is an eleven-job benchmark pipeline constructed to contain at least one instance of each of the nine DevOps PIM concepts [7], used for the coverage evaluation because any concept absent from the output reflects a generation gap rather than an absent input. Each concept appears in a single structural form, so the benchmark measures

⁷<https://github.com/modeldrivendevopsai/mddoi>

TABLE I
THREE-STEP ARTIFACT GENERATION CHAIN WITH PASS CRITERIA.

Step	Description	Inputs	Pass Criterion
Step 1 (Metamodel)	Generates the PSM metamodel	gha.ecore, target platform documentation	Loads in Eclipse EMF without validation errors
Step 2 (ATL)	Generates the ATL transformation	cicd2gha.atl, pimMM.ecore, Step 1 metamodel	Executes without runtime errors; all XMI class references resolve
Step 3 (Acceleo)	Generates the Acceleo template	gha_generate.mt, Step 1 metamodel, target platform documentation	Generated YAML is syntactically valid per the platform schema

whether a concept can be generated correctly at least once, not whether every variant of it is handled.

Coverage evaluation. Round count measures the cost of convergence but not the quality of what the chain produces. To answer the RQ quantitatively, an LLM-as-a-Judge assessment [22] is applied to the full-concept benchmark output for each platform. The judge receives the DevOps PIM concept definition, the platform’s official schema documentation for that concept, and the generated YAML, and rates each of the nine concepts on a single criterion: correctness, defined as whether the generated construct uses the correct platform syntax, has the required fields present, and preserves the semantic intent of the PIM. Ratings are Full, Partial, or None. Executability is assessed separately by YAML syntax validation for all platforms and additionally by the GitLab CI built-in linter for Run 1. Using an externally defined concept set makes the evaluation independent of the author’s own modeling choices.

AI model. All generation and LLM-as-a-Judge steps use Claude Sonnet 4.6⁸, selected on the basis of its demonstrated strength on code and structured text generation benchmarks and its support for long-context prompts, which are necessary to accommodate multi-artifact grounding contexts exceeding 10,000 tokens in several rounds.

Fig. 4 summarises the complete experimental design, showing the fixed input context, the three-step artifact chain applied across all three platform runs, cross-platform constraint transfer, and the pass criteria applied at each step.

V. RESULTS AND DISCUSSION

The transformation chain reached a workable state for all three platforms. The following results focus on where the effort to reach convergence was greatest, which DevOps PIM concepts were produced correctly, and how the observed error patterns indicate the current limits of using LLM-based context engineering for generating MDE artifacts.

A. Round Counts

Table II reports the number of refinement rounds required per step and per platform.

The reduction in total rounds from Run 1 (11 rounds) to Run 2 (6 rounds) and Run 3 (5 rounds) is explained by the reuse of constraints learned on earlier platforms when working on later ones. The three-job and four-job pipelines yielded syntactically valid YAML on all three platforms, as confirmed by YAML

 TABLE II
REFINEMENT ROUNDS PER STEP AND PLATFORM

Step	GitLab CI	Bamboo	Azure DevOps
Step 1 (Metamodel)	1	3	2
Step 2 (ATL)	6	1	2
Step 3 (Acceleo)	4	2	1
Total	11	6	5

validators, which indicates that the transformation chain also works for inputs other than the initial benchmark scenario.

Run-by-Run Summary. Run 1 (target platform: GitLab CI, 11 rounds) establishes the baseline constraint library. Step 1 is clean in a single round: the LLM correctly identifies GitLab-specific constructs and produces a valid metamodel without guidance. Step 2 requires six rounds, with the dominant cost arising from ATL and Object Constraint Language (OCL) constructs that are rejected by the Eclipse ATL engine but accepted in standard OCL: the `texttt{and}` operator does not short-circuit and triggers a runtime crash in filter expressions; `texttt{->max() }` and `texttt{oclAsType() }` are unsupported and produce runtime failures; and `texttt{def}`, `texttt{rule}`, and `texttt{module}` are reserved ATL keywords that cannot be used as variable names. Each is discovered in a separate round and encoded as an explicit constraint. A recurring violation of the reserved keyword constraint persists through three rounds despite the constraint being present in the prompt. In Round 6, the only required change is repositioning all constraints to appear immediately after the 754-line in the ATL file, right before the generation task. Step 3 requires four rounds to resolve two interacting issues: the `texttt{def}` keyword, which is also reserved in Acceleo, and the `texttt{[comment @main/]}` annotation, which must appear as the first line inside the template body and not before it. Both require multiple constraint refinements before they stop regressing.

Run 2 (target platform: Bamboo, 6 rounds) demonstrates direct constraint transfer. Step 1 requires three rounds to resolve EMF-specific enum naming rules: the LLM uses Java enum name casing for `texttt{defaultValueLiteral}` values (`texttt{GLOBAL}` instead of `texttt{global}`) and uses hyphenated strings as enum `texttt{name=}` values, which are valid YAML literals but not valid Java identifiers for EMF code generation. Step 2 passes on the first attempt: all six constraints accumulated during Run 1 are pre-loaded, and none of the previ-

⁸<https://www.anthropic.com/news/claude-sonnet-4-6> Accessed March 2026

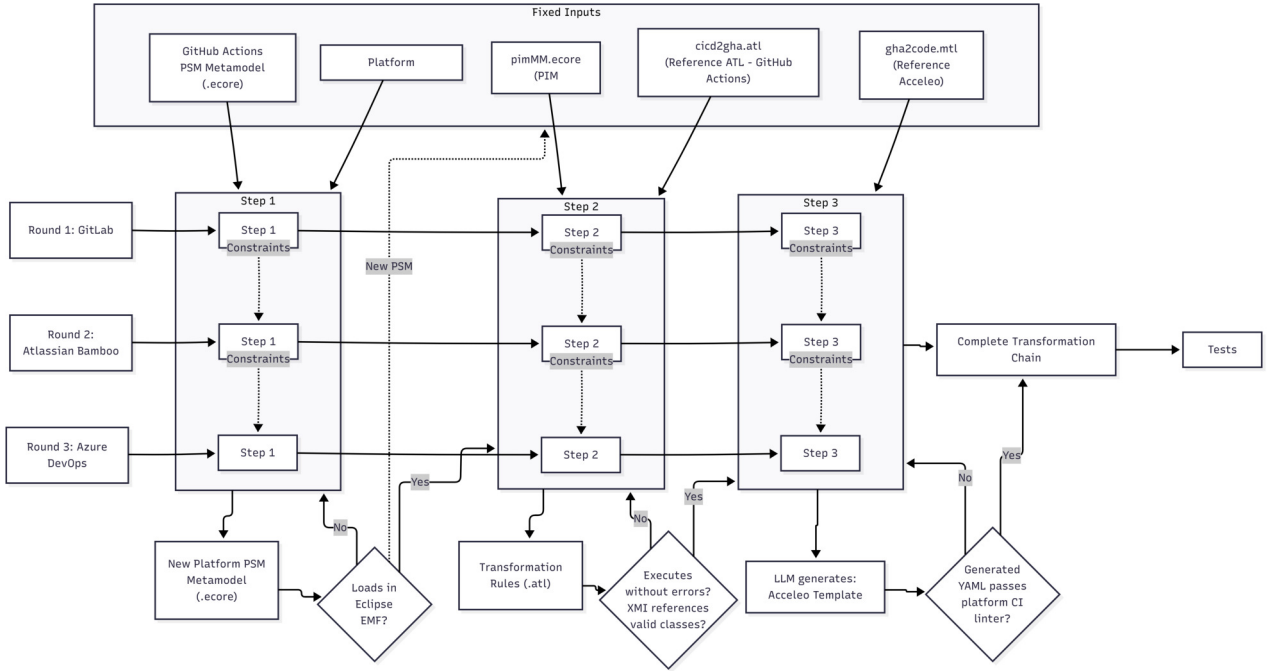


Fig. 4. Complete experiment flow.

ously observed ATL errors recur. Step 3 requires two rounds, also benefiting from the GitLab Acceleo constraints; the single new error is the over-application of `texttt{post(trim())}` to sub-templates that receive an indentation string as a parameter. The added constraint is: do not apply `texttt{post(trim())}` to any template that receives an indent parameter or is called from within an indented block and it passes.

Run 3 (target platform: Azure DevOps, 5 rounds) confirms that all accumulated constraints transfer without regression. Step 1 requires two rounds: the LLM generates a variable map entry class without the EMF-required `texttt{value} EAttribute`, producing a validation error. One constraint resolves it. Step 2 requires two rounds: the LLM attempts to construct the string `texttt{variables['name']}` using ATL's `texttt{''''}` single-quote embedding pattern, which the ATL parser cannot resolve at token boundaries. One constraint prohibiting this pattern resolves it. Step 3 passes in one round.

B. Concept Coverage

Table III reports LLM-as-a-Judge correctness ratings for each of the nine DevOps PIM concepts across all three platforms.

Matrix and Services gaps in Bamboo reflect platform scope rather than generation failure and are identifiable from documentation before generation begins.

C. Discussion

The hypothesis stated that iterative LLM-based context engineering, grounded in existing MDE artifacts and platform documentation, can produce a complete and correct transformation chain for CI/CD pipeline scenarios. The experiment confirms this: convergence was achieved for every step on every platform, and the generated YAML satisfied the pass criterion in every case. The hypothesis is supported within the boundaries established below.

1) *Research Question Answer:* The RQ asks to what extent correctness is achievable. Three dimensions bound the answer:

- **Convergence cost:** between 5 and 11 total rounds were required per platform, confirming the approach is iterative rather than one-shot.
- **Concept coverage:** full correctness was reached for 4 to 6 of the 9 DevOps PIM concepts per platform.
- **Transferability:** cross-platform constraint transfer measurably reduces refinement effort across successive runs. Run 2 Step 2 passed on the first attempt with all Run 1 constraints pre-loaded, contributing zero new constraints. Across all three runs the experiment produced a library of 14 platform-agnostic constraints - 9 from Run 1, 3 from Run 2, and 2 from Run 3 - encoding reusable behavioral knowledge about the Eclipse ATL and Acceleo processors that any subsequent platform run can pre-load from round 1. This supports the interpretation that accumulated constraints encode platform-agnostic behavioural properties of the ATL and Acceleo processors rather than platform-specific semantic knowledge.

2) *Key Findings:*

TABLE III
 LLM-AS-A-JUDGE COVERAGE EVALUATION BY DEVOPS PIM CONCEPT

Concept	GitLab CI		Bamboo		Azure DevOps	
	Rating	Note	Rating	Note	Rating	Note
Pipeline	Full	—	Full	—	Full	—
Job	Full	—	Full	—	Full	—
Matrix	Full	—	None	No platform construct	Full	—
Agent	Full	—	Partial	Single capability type tested	Full	—
Services	Partial	Port binding absent	None	No platform construct	Partial	Redis service absent
Trigger	Partial	Branch and tag filters absent	Partial	Cron expression truncated	Partial	Cron expression truncated
Parameters	None	Not modeled	Full	—	Full	—
Steps	Partial	Plugin as invalid script line	Partial	ScriptTask and CacheTask correct	Partial	ConditionalStep not mapped
Expressions & Vars	Full	—	Full	—	Full	—

- **Domain knowledge is not the barrier.** No round addressed a misunderstanding of CI/CD concepts: every corrective constraint targeted tool-specific syntax behaviour in ATL or Acceleo. This is consistent with the view that the LLM holds sufficient CI/CD domain knowledge but requires explicit grounding for MDE tooling conventions.
- **Instruction placement is an independent variable.** Instruction placement within long-context prompts is a controllable variable independent of instruction content: repositioning constraints to appear immediately before the generation task resolved a recurring violation that three earlier rounds, using the same constraint at an earlier position, had not corrected.

3) *Limitations and Open Questions:* Remaining concept gaps are attributable to platform scope limits rather than generation failure. Concepts without a native platform construct produce absent output identifiable from documentation before generation begins and do not consume refinement rounds. It should be noted that the pass criterion targets YAML output, which is the dominant CI/CD pipeline DSL across all three evaluated platforms. The approach is not inherently limited to YAML: the Acceleo template mechanism supports any text-based output format, and platforms using non-YAML DSLs such as the Jenkinsfile Groovy DSL would require a different Step 3 template but would not invalidate the Step 1 or Step 2 artifacts.

The MDE artifacts (the ECORE metamodel, ATL rules, and Acceleo templates) are generated once per platform and maintained by MDE practitioners, whereas the YAML pipeline configurations they emit are regenerated rather than hand-edited. The transformation artifacts themselves, however, inherit no maintainability guarantee from the generation process. No structural quality metric (rule cohesion, template modularity, naming consistency) was applied, and assessing the maintainability of LLM-generated MDE artifacts against handcrafted baselines remains open.

The experiment does not establish that the approach can generate correct artifacts for a CI/CD platform entirely absent

from the LLM training data. All three platforms evaluated are well represented in publicly available data, and the degree to which generation correctness depends on implicit LLM knowledge rather than the explicitly supplied documentation context cannot be isolated from the present results. In a scenario where only platform documentation is available, convergence round counts could be substantially higher.

In consolidated terms: the approach is feasible, iterative, and transferable across platforms, with correctness bounded by platform scope, metamodel completeness, and LLM training data coverage.

D. Threats to Validity

Internal validity is limited by the manual identification of root causes per refinement round: the chosen constraint may not be the minimal sufficient one, and an alternative formulation could reduce or increase the round count. Coverage ratings were produced by an LLM-as-a-Judge assessment using the same model family as the generator, introducing potential self-evaluation bias; the extent of this bias is not independently quantified. External validity is constrained because a single LLM was used throughout: results may not generalize across model families or versions. The evaluation is further limited to platforms with YAML-based pipeline DSLs; results for non-YAML platforms such as Jenkinsfile cannot be inferred from the present data. Construct validity is limited by the platform selection: all three platforms have non-zero LLM training data coverage, so the approach has not been tested against a platform entirely absent from the LLM training corpus. Conclusion validity is constrained by the fact that the number of rounds alone does not reflect how complex the accumulated constraints are, nor how much the combined prompt content grows over successive rounds. Coverage validity is constrained by the construction of the test set. The eleven-job benchmark guarantees at least one occurrence of each of the nine DevOps PIM concepts, but a single occurrence exercises only one structural realization and does not span the concept interactions, nesting depths, and attribute-value variations of production pipelines. A concept rated as "Full" in a different

test may still fail on an untested variant, such as a trigger combining branch, tag, and filters.

VI. CONCLUSIONS AND FUTURE WORK

This paper proposes a multi-agent conceptual architecture for vibe DevOps engineering, realized as a proof-of-concept extension of MDDOAI - a novel paradigm for LLM-based context engineering applied to the automated synthesis of MDE transformation chains for CI/CD pipeline development support. The experiment, orchestrated by a human researcher emulating the envisioned agent roles, establishes the following:

- The three-step artifact chain converges on all three evaluated platforms within 5 to 11 refinement rounds, proving that the approach is sound.
- Cross-platform constraint transfer reduces refinement cost depending on the amount of successive runs.
- Correctness covers 4 to 6 of 9 DevOps PIM concepts per platform, with remaining gaps attributable to platform scope rather than generation failure.
- All evaluated platforms carry non-zero LLM training data coverage, leaving the unknown CI/CD platform scenario without validation.

Future work priorities are:

- Test the approach against a CI/CD platform entirely absent from the LLM training corpus, using only its official documentation as the knowledge source.
- Evaluate the approach against platforms with non-YAML DSLs, including the Jenkinsfile Groovy DSL.
- Implement the full multi-agent architecture described in Section III.

ACKNOWLEDGMENT

This research leading to these results has been supported by the EU Recovery and Resilience Facility within the Project No. 5.2.1.1.i.0/2/24/I/CFLA/003 "Implementation of consolidation and management changes at Riga Technical University, Liepaja University, Rezekne Academy of Technology, Latvian Maritime Academy and Liepaja Maritime College for the progress towards excellence in higher education, science and innovation" academic career PhD grant (ID 1017). AI is used to generate the first version of Figures, and then updated manually by authors. LLM is used for editorial correction, the changes made are minor.

REFERENCES

- [1] J. R. Blazejcs, O. Nikiforova, and O. Pastor, "A Framework for Model-Driven AI-Assisted Generation of IT Project Management Plan and Scope Documents," *Annals of Computer Science and Intelligence Systems*, no. 2025, pp. 121–132, 2025. doi: 10.15439/2025F8736
- [2] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*, ser. A Martin Fowler Signature Book. Addison-Wesley, 2010. ISBN 9780321601919. [Online]. Available: <https://books.google.lv/books?id=9CAxmQEACAAJ>
- [3] O. Bedrina, "The State of CI/CD in 2025: Key Insights from the Latest JetBrains Survey," JetBrains TeamCity Blog, 10 2025. [Online]. Available: <https://blog.jetbrains.com/teamcity/2025/10/the-state-of-cicd/>
- [4] U. Karlovs-Karlovskis, O. Nikiforova, O. Pastor, and N. Tran, "An Industrial Evaluation of Automated CI/CD Pipeline Generation Framework," *Proceedings of the 21st International Conference on Evaluation of Novel Approaches to Software Engineering*, pp. 930–937, 2026. doi: 10.5220/0015048200004015. [Online]. Available: <https://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0015048200004015>
- [5] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," *Computer*, vol. 39, no. 2, pp. 25–31, 2 2006. doi: 10.1109/MC.2006.58
- [6] H. da Gíao, V. Amaral, G. Engels, A. Flores, R. Pereira, S. Sauer, and J. Cunha, "A Metamodel for Reengineering CI/CD Pipelines," in *2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. Los Alamitos, CA, USA: IEEE Computer Society, 10 2025. doi: 10.1109/MODELS67397.2025.00026 pp. 221–231. [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/MODELS67397.2025.00026>
- [7] A. Flores, V. Amaral, H. Gíao, and J. Cunha, "A Two-Level Model-Driven Approach for Reengineering CI/CD Pipelines." Porto: University of Porto, 9 2024. doi: 10.54499/UIDB/50014/2020. [Online]. Available: <https://sciproj.ptcris.pt/157676UID>
- [8] A. Kleppe, J. Warmer, and W. Bast, "The Model Driven Architecture: Practice and Promise. 2003," *Ad-dison Wesley*, p. 170, 2003. [Online]. Available: <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:The+Model+Driven+Architecture:+Practice+and+Promise.+2003#6>
- [9] J. Whittle, J. Hutchinson, and M. Rouncefield, "The State of Practice in Model-Driven Engineering," *IEEE Software*, vol. 31, no. 3, pp. 79–85, 2014. doi: 10.1109/MS.2013.65
- [10] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large Language Models for Software Engineering: A Systematic Literature Review," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 8, 12 2024. doi: 10.1145/3695988. [Online]. Available: <https://doi.org/10.1145/3695988>
- [11] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang, "Large Language Model Based Multi-agents: A Survey of Progress and Challenges," in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*. California: International Joint Conferences on Artificial Intelligence Organization, 8 2024. doi: 10.24963/ijcai.2024/890. ISBN 978-1-956792-04-1 pp. 8048–8057.
- [12] U. Karlovs-Karlovskis, O. Nikiforova, O. Pastor, and K. Vēveris, "MDDOAI: A Model-Driven DevOps Approach to CI/CD Automation." Riga: IEEE, 11 2025. doi: 10.1109/itms67030.2025.11236630. ISSN 27716937 pp. 1–9. [Online]. Available: <https://ieeexplore.ieee.org/document/11236630>
- [13] J. Cabot, "Vibe Modeling: Challenges and Opportunities," in *Advances in Conceptual Modeling: ER 2025 Workshops, FCM, CMLS, LLM4Modeling, OntoCom, and QUAMES, Poitiers, France, October 20–23, 2025, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2025. doi: 10.1007/978-3-032-08620-4_7. ISBN 978-3-032-08619-8 pp. 105–118. [Online]. Available: https://doi.org/10.1007/978-3-032-08620-4_7
- [14] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS Q.*, vol. 28, pp. 75–105, 2004. [Online]. Available: <https://api.semanticscholar.org/CorpusID:13553735>
- [15] U. Karlovs-Karlovskis, "Development of a Model-Driven DevOps Solution Based on Context-Engineered LLM Code Generation: PROFES Doctoral Symposium." Salerno: Springer, Cham, 12 2025. doi: 10.1007/978-3-032-12092-2_10. ISBN 978-3-032-12092-2. ISSN 1611-3349 pp. 153–159. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-032-12092-2_10
- [16] J. Cabot, "Exploring the use of large language models in domain-specific language development: an experience report," in *STAF Workshops*, 2025. [Online]. Available: <https://api.semanticscholar.org/CorpusID:283888266>
- [17] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer,

- P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating Large Language Models Trained on Code," 7 2021. doi: 10.48550/arXiv.2107.03374. [Online]. Available: <http://arxiv.org/abs/2107.03374>
- [18] J. Rocco, D. Di Ruscio, C. Di Sipio, P. Nguyen, and R. Rubei, "On the use of large language models in model-driven engineering," *Software and Systems Modeling*, vol. 24, pp. 923–948, 4 2025. doi: 10.1007/s10270-025-01263-8
- [19] S. Schulhoff, M. Ilie, N. Balepur, K. Kahadze, A. Liu, C. Si, Y. Li, A. Gupta, H. Han, S. Schulhoff, P. S. Dulepet, S. Vidyadhara, D. Ki, S. Agrawal, C. Pham, G. C. Kroiz, F. Li, H. Tao, A. Srivastava, H. D. Costa, S. Gupta, M. L. Rogers, I. Goncarenco, G. Sarli, I. Galynker, D. Peskoff, M. Carpuat, J. White, S. Anadkat, A. Hoyle, and P. Resnik, "The Prompt Report: A Systematic Survey of Prompt Engineering Techniques," 2024. [Online]. Available: <https://api.semanticscholar.org/CorpusID:276617957>
- [20] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitan, R. O. Ness, and J. Larson, "From Local to Global: A Graph RAG Approach to Query-Focused Summarization," 2 2025. [Online]. Available: <http://arxiv.org/abs/2404.16130>
- [21] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber, "MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework," 11 2024. [Online]. Available: <http://arxiv.org/abs/2308.00352>
- [22] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena," in *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. [Online]. Available: <https://openreview.net/forum?id=uccHPGDlao>