Toward Generative AI-Assisted Model Transformation in Model-Driven Engineering

Duy Dao
Mälardalen University
Sweden
khanh.duy.dao@mdu.se

Abstract

Model transformation is a central automation mechanism in Model-Driven Engineering (MDE), but transformation development and adaptation still require expertise in metamodeling, transformation languages, execution engines, serialization formats, and validation tools. Large Language Models (LLMs) may reduce this entry barrier by allowing users to express transformation intent through prompts, examples, and structured artifacts. However, model transformation is not only a generation task. Transformation artifacts must preserve source-target structure, conform to metamodel constraints, and remain consistent when related artifacts evolve.

My doctoral research investigates how LLMs can support model transformation in MDE while preserving explicit checks for conformance, structure, and consistency. The work is organized around two research questions: first, what capabilities and limitations LLMs exhibit when performing model transformation; and second, how LLM-assisted transformation can be improved through source-target examples and explicit MDE services. Completed studies show that LLMs can support regular and structurally simple transformations, but their reliability decreases with structural complexity, semantic richness, and change propagation. The remaining work, therefore, moves from direct generation toward workflows where LLMs coordinate MDE services for metamodel access, example selection, checking, execution, diagnostics, validation, and repair.

Keywords

Model-Driven Engineering, Model Transformation, Large Language Models, Model Transformation by Example, Round-Trip Consistency

ACM Reference Format:
Duy Dao. 2026. Toward Generative AI-Assisted Model Transformation in Model-Driven Engineering. In *ACM/IEEE 29th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion 2026), October 04–09, 2026, Málaga, Spain.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3837062.3839543

1 Problem

Model-Driven Engineering (MDE) promotes models as first-class artifacts for abstraction, automation, analysis, and evolution [4, 19]. Model transformation is one of its main automation mechanisms, supporting derivation, synchronization, migration, and generation across modeling languages, abstraction levels, and technical spaces [9, 10, 20]. Typical cases include generating implementation artifacts from UML models, deriving schemas from conceptual models, transforming engineering models into digital-twin representations, and keeping related artifacts consistent as they evolve.

Despite this role, transformation development and adaptation remain difficult in practice. The main barrier is not the execution of an already implemented transformation, which should be repeatable for a given input and configuration. The barrier lies in defining, adapting, configuring, validating, and maintaining transformation specifications. This requires knowledge of source and target metamodels, transformation languages such as ATL or QVT, execution engines, XMI serialization, validation procedures, and diagnostic messages from modeling tools [11]. As a result, there is a gap between stakeholders who understand the source and target domains and those who can implement dependable transformations.

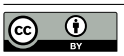
LLMs may reduce part of this barrier because they can process natural-language instructions, serialized models, metamodel fragments, mapping rules, examples, and tool diagnostics [2, 7, 17]. This makes them relevant for expressing transformation intent and supporting transformation development. However, model transformation is not only a generation task. A target artifact may look plausible while omitting required elements, breaking references, using wrong types, or violating the target metamodel. In evolving settings, an LLM may recognize a local edit but fail to propagate its consequences across related artifacts [8, 13, 16]. Direct LLM generation, therefore, improves accessibility but does not by itself provide metamodel conformance, structural correspondence, semantic adequacy, or consistency under evolution.

The overall research goal is:

> **RG:** To investigate how LLMs can reduce the entry barrier and accidental complexity for model transformation in MDE while preserving explicit checks for conformance, structure, and consistency.

The stakeholders in this research are domain engineers, MDE engineers, and transformation tool builders. Domain engineers may know the intended source-target correspondence but not the transformation language or toolchain. MDE engineers need support for developing and adapting reusable transformation specifications. Transformation tool builders need evidence on where LLM support is useful and where transformation development should remain under explicit MDE tool control, such as metamodel access, checking, execution, diagnostics, validation, and repair.

This framing also clarifies the role of accidental and essential complexity in the thesis, as shown in Figure 1. The work aims to reduce accidental complexity caused by transformation languages,



MODELS Companion 2026, Málaga, Spain

ACM ISBN 979-8-4007-2903-4/2026/10
https://doi.org/10.1145/3837062.3839543

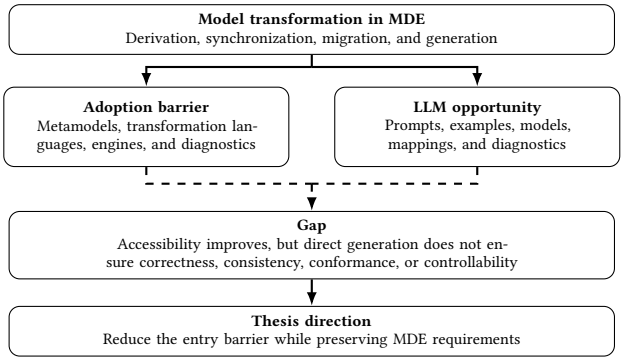


Figure 1: Problem framing of the thesis.

tool configuration, serialization conventions, and diagnostic interpretation. It does not assume that the essential complexity of source-target mapping can be removed. Instead, it studies how such complexity can be made explicit through examples, metamodels, transformation specifications, validation, and tool feedback. The problem addressed in this PhD project is therefore how to make transformation development more accessible without weakening the MDE checks needed for dependable transformation artifacts.

This goal is addressed through two research questions:

RQ1 What capabilities and limitations do LLMs exhibit when performing model transformation in MDE, particularly with respect to structural preservation, metamodel conformance, and consistency under artifact evolution?

RQ2 How can LLM-assisted model transformation be improved over the limitations observed in RQ1 while keeping transformation artifacts inspectable, executable, and reusable?

RQ1 evaluates LLMs as direct support for transformation. It covers zero-shot transformation and transformation under artifact evolution and identifies where stand-alone LLMs are useful, where they fail, and which MDE properties are difficult to preserve. RQ2 investigates improvement mechanisms for LLM-assisted transformation. It covers model transformation by example and service-mediated transformation workflows and studies whether examples, metamodel access, transformation checking, execution, diagnostics, validation, and repair can improve transformation quality without replacing transformation languages, execution engines, or validation tools.

2 Expected Contributions

The expected contributions combine empirical evidence on LLM-based model transformation with a constructive workflow for tool-supported transformation development. C1 and C3 address the capabilities and limitations of LLMs in direct and evolution-aware transformation settings. C2 and C4 address how transformation quality can be improved through examples and explicit MDE services.

C1: Evidence on the limits of stand-alone LLMs for model transformation. This contribution establishes a baseline for using LLMs as out-of-the-box model transformers. It uses controlled transformation settings, oracle-based comparison, repeated generation under fixed prompts, and error analysis to identify recurring failure modes, including missing elements, wrong types, unstable inheritance realization, enumeration errors, and weak reference handling. The contribution clarifies which structurally simple transformations can benefit from direct LLM generation and where stand-alone LLMs are insufficient for dependable transformation development.

C2: An evaluation of LLM-based model transformation by example. This contribution evaluates whether source-target examples can improve transformation quality without requiring users to write formal transformation specifications. Examples are selected from a repository of concrete source-target pairs and provided to the LLM as in-context transformation guidance, together with the new source model and, when available, mapping rules or metamodel fragments. The contribution shows when examples help the LLM preserve structural and semantic fidelity, and when they introduce ambiguity, irrelevant context, or prompt complexity. Its value is to assess model transformation by example as a low-barrier mechanism for guiding LLM-based transformation.

C3: A benchmark for consistency under artifact evolution. This contribution evaluates LLMs in transformation settings where related artifacts evolve. It uses controlled edit operations and round-trip transformation to assess whether changes are preserved across coupled artifacts without syntactic invalidity, structural inconsistency, or unintended drift. The contribution characterizes which edit types can be propagated reliably and which require coordinated updates across references, containment, inheritance, multiplicities, or declarations. Its value is to show the limits of LLMs for co-evolution scenarios where consistency must be maintained across repeated changes.

C4: A service-mediated workflow for LLM-assisted transformation development. This contribution investigates a workflow in which an LLM coordinates explicit MDE services instead of replacing transformation languages or engines. The services cover metamodel access, example selection, transformation checking, execution, diagnostics, validation, and repair. Candidate ATL or QVTo specifications may vary across runs, but checking, execution, loading, and diagnostics are delegated to MDE tools. The expected outcome is a workflow that supports reusable transformation development while keeping transformation artifacts inspectable, executable, and validated through explicit MDE checks.

3 Related Work

This research is positioned at the intersection of model transformation, model transformation by example, LLMs for MDE, and artifact consistency.

Model transformation languages and engines provide explicit specifications, repeatable execution, and integration with modeling infrastructure [10, 20]. They support model-to-model transformation, model-to-text transformation, synchronization, migration, and code generation. Their main advantage is that transformation behavior can be inspected, executed, tested, and maintained. Their practical limitation is the expertise required to define transformations, configure tools, understand metamodels, and interpret diagnostics [11]. This motivates work on lowering the entry barrier without weakening the explicit checks provided by MDE tools.

Model transformation by example reduces part of this specification effort by deriving transformation behavior from source-target examples rather than manually written rules [12, 22]. Search-based and neural approaches have also been used to infer transformations [6, 14]. These approaches reduce explicit rule programming, but still depend on representative examples, refinement, or training data. My work builds on this line by studying whether LLMs can use source-target examples directly as transformation guidance, and where examples are insufficient for preserving structure and semantics.

LLMs have recently been applied to software engineering and MDE tasks, including code generation, domain modeling, model extraction, prompt engineering, and model transformation [2, 7, 8, 13, 16, 17]. Existing studies show that LLMs can produce useful modeling and transformation artifacts, but also report limitations in correctness, completeness, and metamodel conformance. This motivates task-specific evaluation of LLM-based transformation against MDE properties such as element coverage, source-target correspondence, and structural validity, rather than relying on plausible generated artifacts.

Artifact evolution adds a further MDE concern. Models, code, and schemas often co-evolve, and changes must be propagated consistently across related artifacts [18, 23]. Round-trip engineering and bidirectional transformations study how related artifacts can be kept consistent under change [1, 21]. These works motivate evaluating LLMs not only for one-time generation, but also for transformation settings where edits must be preserved across coupled artifacts without invalidity or unintended drift.

Finally, prompting and retrieval can improve the context given to an LLM [5, 15], but they do not replace transformation execution, metamodel conformance checking, or validation. My work therefore investigates LLM-assisted transformation as a service-mediated MDE workflow, where the LLM coordinates explicit services for metamodel access, example selection, checking, execution, diagnostics, validation, and repair, rather than replacing transformation languages and engines.

4 Proposed Approach

The PhD project follows an iterative software engineering research style inspired by Basili's experimental paradigm and quality improvement process [3]. This is suitable because the thesis combines empirical evaluation with constructive workflow design. Each stage characterizes a transformation setting, defines evaluation criteria, executes controlled studies, analyzes failure modes, and uses the results to refine the next stage. The approach uses established MDE methods and tools, including transformation languages, metamodel-based validation, oracle comparison, model transformation by example, round-trip consistency, and transformation testing and debugging.

Figure 2 shows the research process used throughout the thesis. The process starts from the MDE context and the limitations of LLM-based transformation. It then defines research questions and evaluation criteria, designs controlled studies, executes experiments or prototype assessments, and uses the observed failure modes to refine the next study. This process supports the thesis structure:

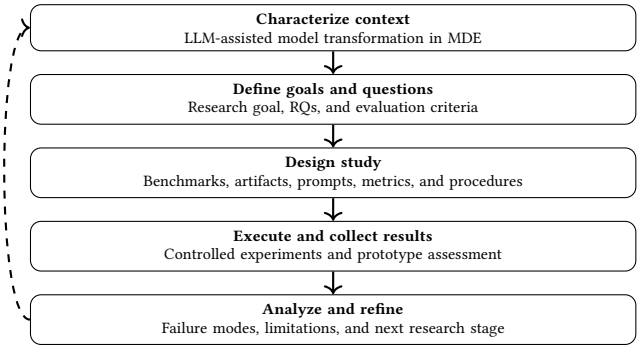


Figure 2: Iterative software engineering research style used in the thesis.

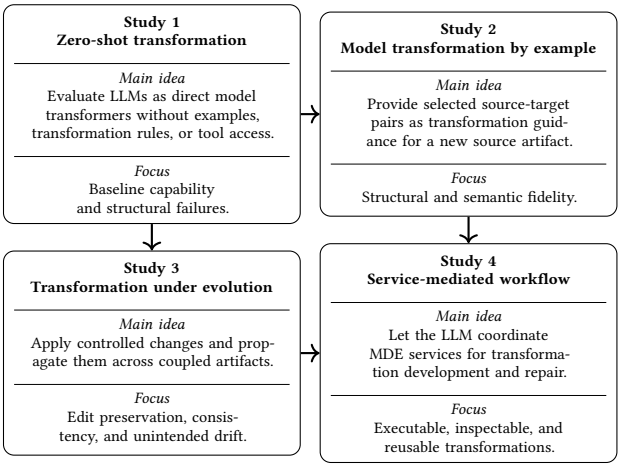


Figure 3: Four connected studies in the thesis, moving from direct LLM transformation to LLM-assisted transformation development with explicit MDE services.

the empirical studies identify limitations of direct LLM-based transformation, and the workflow study addresses them through explicit MDE services.

Figure 3 summarizes the four connected studies. Study 1 evaluates zero-shot transformation as the lowest-barrier setting: the LLM receives a source artifact and produces a target artifact without examples, transformation rules, or tool access. This establishes a baseline for the capabilities and limits of stand-alone LLMs.

Study 2 evaluates LLM-based model transformation by example. Concrete source-target pairs from the studied scenarios are provided as in-context examples together with the new source artifact. The study examines whether selected examples improve structural and semantic fidelity, and whether additional examples introduce ambiguity or prompt complexity.

Study 3 evaluates transformation under artifact evolution. A controlled change is introduced in one artifact, propagated to the coupled artifact, and then propagated back. The evaluation focuses on edit preservation, structural validity, and unintended drift. This captures MDE settings where related artifacts must remain consistent across repeated changes.

Study 4 develops a service-mediated transformation workflow. The LLM interacts with MDE services for metamodel access, example selection, transformation checking, execution, diagnostics, validation, and repair. Candidate ATL or QVTo specifications may be generated and revised by the LLM, but transformation-critical operations remain delegated to MDE tools. In this study, repeatability refers to tool-controlled operations: for the same transformation specification, input model, metamodels, and runtime configuration, checking, execution, loading, and diagnostic reporting are expected to produce repeatable results. The aim is, therefore, not to make LLM generation deterministic but to constrain LLM variability through explicit MDE operations and diagnostics. The resulting transformation specification can then be inspected, tested, repaired, and reused.

Risks and ethical issues. The main research risk is overclaiming what LLM-generated artifacts can guarantee. I therefore separate parsability, loadability, metamodel conformance, structural completeness, semantic preservation, consistency under change, and executable success. The use of LLMs also raises sustainability concerns, since repeated prompting, regeneration, and repair can increase computational cost. This risk is partially addressed by emphasizing reusable transformation artifacts, including examples, mapping knowledge, transformation specifications, diagnostics, and evaluation scripts, rather than treating each transformation as a one-off generation task. The proposed workflow is also not intended to replace transformation engineers. Instead, it keeps generated artifacts inspectable, executable, and subject to explicit MDE checks so that engineers can validate, adapt, and control the resulting transformations. Reproducibility is addressed by archiving prompts, datasets, outputs, model identifiers, and evaluation scripts. The current studies use controlled artifacts and do not involve sensitive personal data. If later user studies are conducted, informed consent and institutional ethical procedures will be followed.

5 Evaluation Plan

The evaluation combines controlled experiments, oracle comparison, structural validation, error analysis, round-trip checks, and prototype assessment. Across the studies, the evaluation focuses on MDE-relevant properties: metamodel conformance, element coverage, source-target correspondence, semantic adequacy, consistency under change, and inspectability. The purpose is not only to measure whether LLMs produce valid-looking artifacts, but to assess whether the generated artifacts and transformation specifications preserve properties required in MDE workflows.

For the zero-shot and example-guided studies, generated artifacts are compared with oracle artifacts. The metrics distinguish correct, incorrect, additional, and missing elements, with higher penalties for missing or structurally harmful elements. Structural preservation is evaluated because the studied scenarios, UML-to-Java, RDBMS-to-UML, SysML-to-AAS, and model/schema co-evolution, require preservation of references, containment, types, and domain-specific elements. Error analysis is used to identify recurring failure modes such as incomplete element coverage, incorrect type mapping, unstable inheritance realization, and weak reference handling.

For the evolution study, controlled edit operations are evaluated in round-trip transformation settings. The main criteria are edit persistence, parsability or loadability, structural validity, and run-to-run consistency. This evaluates whether changes can be propagated across coupled artifacts without invalidity or unintended drift. The analysis distinguishes local edits, such as renaming or deletion, from changes that require coordinated updates across references, containment, inheritance, multiplicities, or declarations.

For the service-mediated workflow, the evaluation compares direct LLM generation with tool-supported transformation development. The criteria include executable success, diagnostic recovery, metamodel-related failures, service-call traces, and structural output quality. Executable success means that a transformation can be checked, executed, serialized, and loaded against the registered target metamodel. Since executable success does not guarantee mapping completeness or semantic adequacy, the evaluation also examines cases where transformations are executable but produce incomplete or semantically weak target artifacts.

6 Current Status

The project is in the early-to-middle stage of the PhD. Three empirical studies have been completed, and the service-mediated workflow is under development. The completed studies establish the empirical basis for the thesis by identifying where LLMs are useful for model transformation and where explicit MDE support is needed.

For **RQ1**, I have completed two studies on the capabilities and limitations of LLMs for model transformation. The first is a zero-shot UML-to-Java benchmark with 116 UML class diagram models serialized in XMI/Ecore. The results show that LLMs can produce correct outputs for simpler models, but correctness decreases as structural complexity increases. The main errors concern type and collection mapping, inheritance realization, enumeration handling, and element coverage. The second is a change-aware round-trip benchmark using paired class-oriented data models and relational schemas. The results show that LLMs can often handle small local edits, such as renaming or deletion, but struggle with changes that require coordinated propagation, including splitting or merging structures, introducing inheritance, changing multiplicities, and introducing enumerations. Together, these studies show that standalone LLMs provide useful low-barrier support in structurally limited settings, but are not sufficient for dependable transformation development.

For **RQ2**, I have completed an LLM-based model transformation by example study across RDBMS-to-UML, UML-to-Java, and SysML-to-AAS, and I am developing a service-mediated transformation workflow. The example-based study shows that LLMs perform well in regular and syntactically predictable transformations, especially RDBMS-to-UML. UML-to-Java is more sensitive to syntax and implementation details, while SysML-to-AAS is more difficult because it requires semantic interpretation of engineering concepts and mapping to a domain-specific target structure. More examples do not consistently improve the results, which indicates that examples alone are not sufficient as transformation guidance. The current workflow, therefore, exposes MDE operations as services for metamodel access, example selection, checking, execution, diagnostics, and repair. Preliminary assessment with UML-to-Java and SysMLv2-to-AAS using ATL and QVTo shows that an LLM can

generate and repair executable transformations in several configurations. However, executable success does not guarantee structural completeness or semantic adequacy, so validation beyond execution remains necessary.

The next stage of the PhD will focus on completing the service-mediated workflow, extending its evaluation, and consolidating the empirical and workflow results. Later PhD work will study broader model transformation and model management workflows, including validation, repair, and evolution beyond the current scenarios.

Acknowledgments

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

References

[1] Yuan An, Xiaohua Hu, and Il-Yeol Song. 2008. Round-Trip Engineering for Maintaining Conceptual-Relational Mappings. In *Advanced Information Systems Engineering*. Springer, 296–311.

[2] Sathurshan Arulmohan, Marie-Jean Meurs, and Sébastien Mosser. 2023. Extracting Domain Models from Textual Requirements in the Era of Large Language Models. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 580–587. doi:10.1109/MODELS-C59198.2023.00096

[3] Victor R. Basili. 1993. The experimental paradigm in software engineering. In *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, H. Dieter Rombach, Victor R. Basili, and Richard W. Selby (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–12.

[4] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2012. *Model-Driven Software Engineering in Practice*. Morgan & Claypool. doi:10.2200/S00441ED1V01Y201208SWE001

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL] https://arxiv.org/abs/2005.14165

[6] Loli Burgueño, Jordi Cabot, Shuai Li, and Sébastien Gérard. 2022. A Generic LSTM Neural Network Architecture to Infer Heterogeneous Model Transformations. *Software and Systems Modeling* 21, 1 (2022), 139–156. doi:10.1007/s10270-021-00893-y

[7] Kua Chen, Yujing Yang, Boqi Chen, José Antonio Hernández López, Gunter Mussbacher, and Daniel Varro. 2023. Automated Domain Modeling with Large Language Models: A Comparative Study. In *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 162–172. doi:10.1109/MODELS58315.2023.00037

[8] Robert Clarisó and Jordi Cabot. 2023. Model-Driven Prompt Engineering. In *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 47–54. doi:10.1109/MODELS58315.2023.00020

[9] Krzysztof Czarnecki and Simon Helsen. 2003. Classification of Model Transformation Approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of Model-Driven Architecture*.

[10] K. Czarnecki and S. Helsen. 2006. Feature-based survey of model transformation approaches. *IBM Systems Journal* 45, 3 (2006), 621–645. doi:10.1147/sj.453.0621

[11] Nafiseh Kahani, Mojtaba Bagherzadeh, James Cordy, Juergen Dingel, and Daniel Varro. 2019. Survey and Classification of Model Transformation Tools. *Software and Systems Modeling* 18 (2019). doi:10.1007/s10270-018-0665-6

[12] Gerti Kappel, Philip Langer, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. 2012. *Model Transformation By-Example: A Survey of the First Wave*. Springer, 197–215. doi:10.1007/978-3-642-28279-9_15

[13] Gabriel Kazai, Ronnie Agyeiwaa Osei, Alessio Bucaioni, and Antonio Cicchetti. 2025. Model Transformations Using LLMs Out-of-the-Box: Can Accidental Complexity Be Reduced?. In *First Workshop on Large Language Models For Generative Software Engineering*. http://www.es.mdu.se/publications/7201-

[14] Marouane Kessentini, Houari Sahraoui, Mounir Boukadoum, and Omar Omar. 2012. Search-Based Model Transformation by Example. *Software and Systems Modeling* 11 (2012), 1–18. doi:10.1007/s10270-010-0175-7

[15] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401 [cs.CL] https://arxiv.org/abs/2005.11401

[16] Zahra Moezkarimi, Kevin Eriksson, Albin Alm Johansson, Alessio Bucaioni, and Marjan Sirjani. 2024. Harnessing ChatGPT for Model Transformation in Software Architecture: From UML State Diagrams to Rebeca Models for Formal Verification. In *4th International Workshop of Model-Driven Engineering for Software Architecture*. http://www.ipr.mdu.se/publications/7130-

[17] Ipek Ozkaya. 2023. Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications. *IEEE Software* 40, 3 (2023), 4–8. doi:10.1109/MS.2023.3248401

[18] Dong Qiu, Bixin Li, and Zhendong Su. 2013. An Empirical Analysis of the Co-Evolution of Schema and Code in Database Applications. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. 125–135. doi:10.1145/2491411.2491431

[19] Douglas C. Schmidt. 2006. Guest Editor's Introduction: Model-Driven Engineering. *Computer* 39, 2 (2006), 25–31. doi:10.1109/MC.2006.58

[20] S. Sendall and W. Kozaczynski. 2003. Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software* 20, 5 (2003), 42–45. doi:10.1109/MS.2003.1231150

[21] Perdita Stevens. 2020. Maintaining Consistency in Networks of Models: Bidirectional Transformations in the Large. *Software & Systems Modeling* 19 (2020). doi:10.1007/s10270-019-00736-x

[22] Dániel Varró. 2006. Model Transformation by Example. In *Model Driven Engineering Languages and Systems*. Springer, 410–424. doi:10.1007/11880240_29

[23] Panos Vassiliadis, Fation Shehaj, George Kalampokis, and Apostolos V. Zarras. 2023. Joint Source and Schema Evolution: Insights from a Study of 195 FOSS Projects. In *Proceedings of the 26th International Conference on Extending Database Technology (EDBT)*. 27–39. doi:10.48786/edbt.2023.03