Model Deepening with Large Language Models: Insights from Exploratory Studies with ChatGPT

Pierre Maier(✉) and Vicky Kadziolka

University of Duisburg-Essen, Essen, Germany
{pierre.maier,vicky.kadziolka}@uni-due.de

Abstract. Although multi-level modeling has long been argued to showcase benefits in various application domains, its adoption is still hindered not least because the construction of multi-level models may entail a cumbersome and error-prone re-engineering effort. While many studies have investigated the potential of using LLMs to support the automatic construction of two-level conceptual models, such as UML class diagrams, no research has yet been conducted on using LLMs to support the construction of multi-level conceptual models. In this paper, we report on experiments conducted with ChatGPT to support the re-engineering of flat two-level models into deep multi-level models – a process we refer to as model deepening – using the multi-level modeling language FMML^{x}. Our findings indicate that while ChatGPT can significantly aid in semantic tasks during model deepening – such as comparing attribute meanings or analyzing type-object patterns – it also presents challenges, sometimes generating erroneous models by removing and duplicating properties. Future research should aim to develop an overarching model-deepening method that integrates probabilistic information sources, such as ChatGPT, with rule-based algorithms, while clearly defining and leveraging the user's role in guiding and validating the process.

Keywords: multi-level modeling · large language models · model deepening · ChatGPT · FMML^{x}

1 Introduction

Conceptual modeling plays a vital role in the development and maintenance of digital artifacts. Modeling languages, such as the entity-relationship (ER) modeling language and the Unified Modeling Language (UML), as well as common domain-specific modeling languages (DSMLs), provide modelers with control over two levels of classification: a type level and an instance level. For this reason, they are sometimes labeled *two-level modeling languages* [1]. By contrast, *multi-level modeling languages* alleviate this restriction, allowing for the creation and manipulation of elements at any desired classification level. The additional

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2026
C. M. Fonseca et al. (Eds.): ER 2025 Workshops, LNCS 16190, pp. 119–136, 2026.
https://doi.org/10.1007/978-3-032-08620-4_8

abstraction offered by multi-level modeling languages can, among others, support improving the integration, reusability, integrity, and comprehensibility of digital artifacts [7]. An example multi-level model is discussed in Sect. 2.4.

Despite its supposed benefits, which have been known and discussed for more than ten years [1,6], multi-level modeling is sparsely adopted in industry. Multiple factors contribute to this circumstance. For one, the utility of a multi-level model, just like any conceptual model, depends on the competence of the modeler to construct high-quality abstractions with the means of the provided modeling language. However, since multi-level modeling differs from the two-level modeling languages most modelers are familiar with and is, in addition, sparsely taught in universities [10, p. 537], this competence will often be lacking, leading to solutions that are ill-suited to realize the prospects of multi-level modeling. Novices should therefore be assisted in constructing multi-level models. Furthermore, many artifacts within an organization have likely been created using a two-level modeling language. Switching to multi-level languages, then, implies a massive re-engineering effort which would be cumbersome, expensive, and error-prone. For this reason, support is needed to guide re-engineering efforts from two-level to multi-level models. We refer to the process of transforming flat two-level models into deep multi-level models as *model deepening*.

Different approaches may be suited to implement automatic support for model deepening, one of which is the use of large language models (LLMs). LLMs can generate textual output contingent on natural-language prompts given by a user. Because they are pre-trained on large text corpora, they may serve as general-purpose knowledge bases. The advent of LLMs has invited researchers to experiment with using them for all kinds of different problem areas, conceptual modeling being one of them. Publications such as [3,17] are just a few of many similar examples that investigate the potentials of LLMs to support conceptual modeling activities. The results are generally mixed; some areas seem to be well-supported by LLMs (e.g., constructing classes with attributes based on a natural-language description), while others pose problems as of now (e.g., modeling relationships between classes or writing expressions in the Object Constraint Language [OCL]). Since model deepening must consider semantic questions (see 3.1), LLMs may also be useful to support model deepening. To the best of our knowledge, no publication has yet addressed the potential of LLMs to support the automatic construction of multi-level models in general, or model deepening in particular.

In this paper, we report on experiments conducted with ChatGPT and the multi-level modeling language FMML^{x} for model deepening over the past year. The following research questions guided our efforts:

RQ1: Which abstraction mechanisms in multi-level modeling may be supported by LLMs? Multi-level modeling covers multiple abstraction mechanisms (see Sect. 2.2). An exploratory study on using LLMs for model deepening should investigate the potential of an LLM to support the construction of each kind of abstraction mechanism.

RQ2: What challenges are encountered when using an LLM for model deepening? Furthermore, we aim to report on challenges that occur when using LLMs for model deepening. For some challenges, we may provide alternative means

of interaction or specific training examples that seem to overcome a respective challenge. Others may showcase strict limitations of LLMs.

Our paper is structured as follows. We begin by clarifying our concept of model deepening, which includes a concise explanation of multi-level language features of the FMML^x. In Sect. 3, we elucidate the role of LLMs in model deepening and outline our process for adapting ChatGPT to model-deepening studies. The detailed design of our experiments, alongside gained insights and experiences, is reported on in Sect. 4. We close by discussing implications of our results in Sect. 5 and provide a conclusion in Sect. 6.

2 Model Deepening with FMML^x

2.1 Multi-level Modeling with the FMML^x and XModeler^{ML}

Many different multi-level modeling languages and tools currently co-exist [10], but no standard has emerged so far. The Flexible Multi-Level Modeling and Execution Language (FMML^x) is based on a meta-circular language architecture which defines classes as objects [6]. Consequently, all entities modeled with the FMML^x are objects, some of which possess a class facet and may thus be referred to as a class, too. All classes (i.e., objects which possess a class facet) therefore have a state and are executable. Classes may specify an arbitrary number of attributes, associations, constraints, and operations. These are collectively referred to as the properties of a class. Objects provide slots for attributes and links for associations.

The meta-circularity of FMML^x allows for the specification of an unbounded number of classification levels. In FMML^x, the classification level of an object is expressed by an integer value, which is shown to the left of the class name. The classification level of a class determines its instantiation depth. If an object possesses a class facet (i.e., if an object has a classification level higher than 0), it may not only instantiate properties defined at a higher level, but it may also inherit them. Unlike class properties in two-level languages, in FMML^x, each property must also specify an *instantiation level*. The instantiation level specifies the level at which a property shall be instantiated. In two-level languages, properties are always instantiated at the level immediately below; the instantiation level of all properties in UML classes would always be 0. If a property is not instantiated at the level immediately below, we say its instantiation is *deferred* to lower levels.

The example FMML^x diagram shown in Fig. 3 illustrates these core language features. The L1 (level 1) class `Customer` is associated with an L2 abstract class `Car`. Each property of `Car` specifies an instantiation level, which is expressed by a white number against a black background. The attribute `brandName`, for example, must be instantiated on level 1 while the instantiation of the attribute `hasDamages` is deferred to level 0. The two L2 classes `Cabrio` and `Transporter` are child classes of `Car`. Instances of these child classes serve to represent different car models a rental car company might possess. One such car model is represented by the L1 object `FordMustang`, which is an instance (more accurately:

concretion, see Sect. 2.2) of `Cabrio`. Here, we can provide a brand name, a model name, and a roof construction type, avoiding the need to specify these values redundantly for each particular car. Instances of `FordMustang` serve to represent particular cars that can be rented by customers. It introduces an additional attribute `hasActiveExhaust`, which will only be instantiated for L0 objects of `FordMustang`.

The FMML^{x} is an executable language. Therefore, it is also possible to specify operations and constraints – among other more advanced features. But since these specific language features have not been part of the exploratory studies with ChatGPT, they are not explained in more detail here. Further information on FMML^{x} is provided, e.g., in [9].

The XModeler^{ML} is the corresponding modeling and execution environment for the FMML^{x}. It enables the execution and instantiation of models at runtime. The tool can be downloaded at www.le4mm.org/xmodelerml. FMML^{x} models created with the XModeler^{ML} are saved as XML files, which can be opened again within the tool. This XML representation of FMML^{x} models serves as a starting point for representing models to ChatGPT.

2.2 A Note on Abstraction Mechanisms in Model Deepening

In traditional two-level object-oriented modeling, two abstraction mechanisms are distinguished by a strict dichotomy: *classification* and *generalization*. Classification refers to the inter-level relationship between classes and objects – a class serves as an abstraction over a set of objects. Each object is, in turn, a direct instance of exactly one class. Generalization refers to a specific intra-level relationship between classes in which a so-called parent class serves as a generalization to one or multiple child classes. Child classes inherit all properties of their parent class(es). These properties may also be overwritten.

These traditional abstraction mechanisms are also present in object-oriented multi-level modeling, using languages such as FMML^{x}. However, due to the specific characteristics of multi-level modeling, some differences need to be highlighted.

Intrinsic Classification. Classification in multi-level modeling, just like in two-level modeling, refers to constructing classes from objects. In multi-level modeling, however, all elements can be considered objects; no strict dichotomy between objects and classes exists. Therefore, classification in the sense presented above only occurs between levels 1 and 0, since pure objects (i.e., modeling elements which cannot possess a class facet) can be modeled only on level 0. This traditional sense of classification may be referred to as *pure classification*. Inter-level relationships between elements on levels higher than 1, however, are different from this traditional sense of classification. At least in FMML^{x}, an object on level n>0 may not only instantiate properties of its class on level m=n+1, but, as a class itself, may also inherit and overwrite them. Thus, the typical inter-level relationship between elements in FMML^{x} differs from traditional classification.

This abstraction mechanism has been coined *intrinsic classification*, or, referring to the reverse direction, concretization [8]. In the example model in Fig. 3, `Cabrio` serves as an intrinsic classification to `FordMustang` and `FordMustang` as a concretion to `Cabrio`.

Pure Generalization. Intrinsic classification can be considered a combination of classification in the traditional sense and generalization. Multi-level modeling languages like the FMML^x also allow for modeling generalization relationships in the traditional sense, i.e., as an intra-level relationship between elements with a class facet. This is also illustrated in Fig. 3 where the classes `Cabrio` and `Transporter` are modeled as specializations of the abstract class `Car`. Generalization in this traditional sense is referred to as *pure generalization* in multi-level modeling to clearly distinguish it from inter-level generalization that is part of intrinsic classification.

2.3 Outline of Model-Deepening Process

Given the breadth and variety of two-level languages, it is hardly possible to account for all particularities of each language. Therefore, it is reasonable to focus on a set of commonly available core language constructs that any other two-level language may be transformed into. For this purpose, we rely on two-level modeling with UML++, a two-level dialect of the FMML^x [12]. Two-level modeling with UML++ has the benefit that classes and objects can be modeled at their actual classification levels (classes at M1, objects at M0) as opposed to UML, where classes and objects must both, as instances of respective M2 elements, be modeled at M1 [12]. Additionally, since UML++ is a dialect of FMML^x, this allows us to use the same language for representing flat input and deep output models.

In our conception of model deepening, we focus on the identification of model-deepening patterns present in a flat model. That is, we try to identify snippets of a flat model that should be re-engineered as a multi-level solution. The identification of a model-deepening pattern can be conceived as a two-step process. First, we need to identify *smells* in a flat model, which, by relying on certain heuristics, may indicate the presence of a model-deepening pattern. Second, each identified smell, or candidate pattern, may be approved or denied as a model-deepening pattern. For each approved model-deepening pattern, a tailored deepening transformation must be executed which results in a deep modeling solution. This general process of model deepening is outlined in Fig. 1.

Model-deepening transformations consist of two main types of operations: promotion transformations and extension operations. Promotion transformations, or simply "pull-up" transformations, are frequently discussed in conventional meta-modeling research. A class promotion increases the classification level of a class. The promotion of a class property moves a property to a higher-level class.

Extension operations add at least one element to the current model. They can be distinguished between primary and secondary extensions. Primary exten-

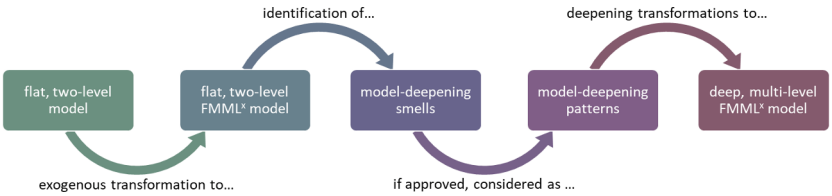

Fig. 1. Outline of model-deepening process

sions refer to the addition of classes required for subsequent promotions of class properties. In this case, an extension is only executed to enable property promotions. A secondary extension adds any element to the model regardless of any potential promotions.

2.4 Model Deepening: An Example

To illustrate our conception of model deepening, as well as features and prospects of multi-level modeling with the FMML^x, consider the flat example model in Fig. 2. It implements the *type-object pattern* to model cars and car models. A type-object pattern is characterized by two classes that are associated with one another, where the association on one end has a multiplicity of 1..1 and on the other end has an unbounded multiplicity. Such an association may be intended to replicate a classification relationship that cannot be modeled in a regular UML class diagram and, as such, is a potential candidate for model deepening.

The type-object pattern can be re-engineered as a multi-level model by promoting the 'type class' (in this case `CarModel` as well as its specializations `Cabrio` and `Transporter`) to level 2. All properties of the type class are promoted as well, and the instantiation level is increased to 1 (in a regular UML++ diagram, the instantiation level of all properties must be 0 because no classes can be added above level 1). Consequently, we can now model 'type objects' such as Ford Mustangs at the level they conceptually belong to – on level 1. Particular cars are instances of car models. Properties of the 'instance class' (in this case: `Car`) are promoted to the L2 class as well, but their instantiation level is kept at 0 since they must be assigned to each particular car individually. The association `is_of` between `Car` and `CarModel` can be removed. Modeling type-object patterns as multi-level hierarchies avoids accidental complexity because fewer elements are required to model the same scenario [1].

The attribute `hasActiveExhaust: Boolean` demands special attention here. As shown in the envisioned multi-level solution, the attribute applies only to the car model `FordMustang`. However, in the two-level solution, this attribute must also be instantiated for all other car models. Here, the two-level solution threatens the integrity of the system. Tedious workarounds (e.g., constraint in class `Car` or adding a specialization `FordMustang` to `Car`) would be required to avoid this integrity threat. Such integrity threats may also be overcome through model deepening, e.g., by detecting that `hasActiveExhaust` is only assigned to particular cars which are linked to the Ford Mustang car model.

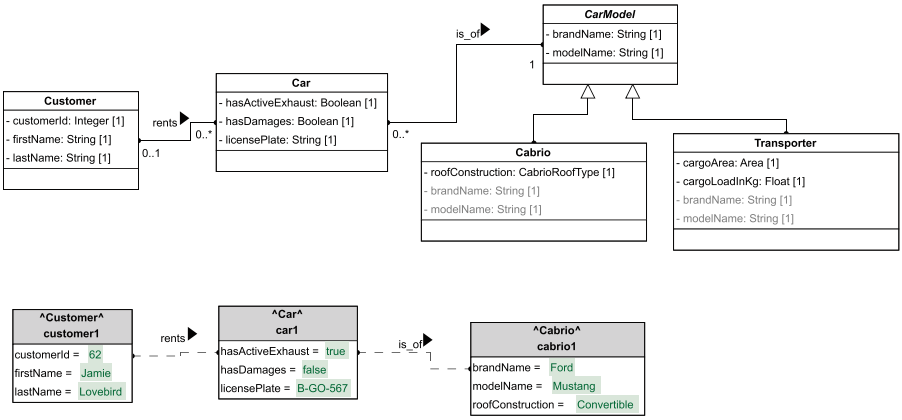

Fig. 2. Two-level model of rental cars in UML++

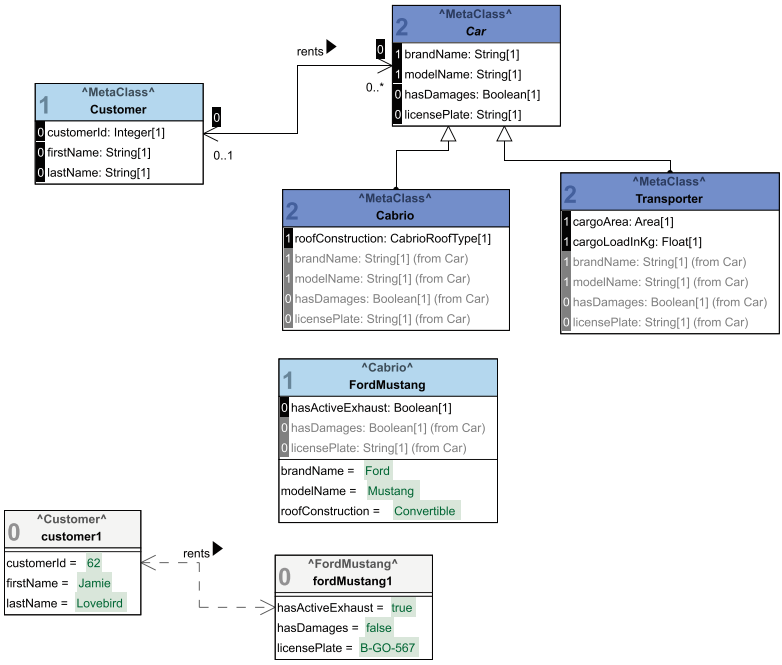

Fig. 3. Multi-level model of rental cars in FMML^x

3 Outline of Model Deepening with FMML^x and ChatGPT

3.1 Role of Large Language Models for Model Deepening

LLMs, in general, are characterized by extensive pre-training, which allows them to serve as general-purpose knowledge bases. Language models function as "string prediction" models [2, p. 5185] that can provide semantic information through the analysis of string co-occurrence, which some refer to as distributional-semantic information (e.g., [13, pp. 134–136]). Manually crafted semantic resources, such as WordNet [4], often struggle to deal with the contingencies and dynamics of human language, as people may invent new terms, use them in a slightly different sense, or simply miswrite them. LLMs, by being trained on a mass of publicly available texts, may help overcome some of these issues.

LLMs may support the automatic construction of multi-level models because effective model deepening must consider semantic issues. In intrinsic classification, such as in the example outlined in Sect. 2.4, semantic information plays a crucial part. Some researchers propose re-engineering flat models into multi-level models by transforming the type-object pattern into a multi-level hierarchy [11,14]. Within these publications, type-object patterns are assumed to be easily detectable by purely syntactical measures such as the presence of aggregation relationships [14] or predefined, static naming patterns [11, p. 63]. But whether a class indeed serves to instantiate 'type objects' for another class can hardly be deduced from linguistic conventions (such as assuming the type class is suffixed with the string "Type") alone. Modelers may always adopt different conventions or avoid them completely. The multiplicity indicator of the type-object pattern mentioned in Sect. 2.4 is also insufficient in this regard since these multiplicities may also be used in cases where the type-object pattern is not implemented (e.g., a customer lives at exactly one address, but each address may host multiple customers). By re-engineering two associated classes as a multi-level hierarchy, we only improve the overall model quality if this tightened coupling represents a time-invariant abstraction. If that is, however, not the case – if, for example, we at some time want to allow customers to be linked to multiple address objects – a multi-level solution may also threaten the integrity of the model.

While such syntactic measures are reasonable heuristics for identifying pattern candidates, relying on them alone will lead to ill-suited model-deepening transformations. Whether objects of one class serve as types to objects of another class is, at its core, a semantic issue.

In pure generalization, one needs to account for the semantic similarities of class properties that may be promoted to a new parent class. Often, the automatic generation of pure generalizations is based solely on a formal comparison of strings. This, however, ignores that one attribute may be written slightly differently or that one attribute in one class (e.g., `birthDay: Date`) may be mapped to multiple attributes in another class (e.g., `birthYear: Integer`, `birthMonth: Integer`, and `birthDay: Integer`), among many similar semantic issues that should be considered.

Large language models may help address these semantic issues in order to support identifying smells in flat models and approving/declining them as model-deepening patterns But while LLMs may provide relevant semantic information, they also need to be adapted in order support multi-level modeling languages in general and model deepening in particular.

3.2 Adapting ChatGPT for Model Deepening via GPT Builder

We used ChatGPT mainly for two reasons. First, many studies in LLM-assisted conceptual modeling focus on ChatGPT as well (e.g., [3,5]). By using ChatGPT, too, our experiences can be compared more directly to those of existing studies. Second, ChatGPT is easy to use and adapt. After conducting some initial experiments, we developed two custom GPTs: one for pure generalization and one for intrinsic classification, using OpenAI's GPT builder tool [15]. Custom GPTs can be adapted through natural-language instructions, and they include a "knowledge base," which consists of user-provided documents. The GPT may access these documents at any point to generate more accurate responses.

Apart from adapting a GPT for specific model-deepening tasks (described for each GPT in Sect. 4), our initial focus was on having it interpret and generate syntactically correct XML files in FMML^x notation. This also includes explaining core concepts of the FMML^x modeling language, such as classes, objects, or data types, alongside their corresponding XML representation. For the custom GPT to support intrinsic classification, this also includes specific multi-level modeling concepts such as stateful classes and deferred instantiation (see Sect. 2.1). We verified each GPT's capability to interpret and generate FMML^x documents by prompting it to analyze provided FMML^x files and generate example FMML^x models of arbitrary domains. The FMML^x examples we provided as a "knowledge base" to the custom GPTs were domain-specific at first. Because we suspected that the custom GPT sometimes simply replicates parts of the provided examples, we used domain-agnostic examples for the final training.

We conducted preliminary studies in isolated conversations with each custom GPT, using different prompting strategies. Because these pre-studies did not yield any conclusive answers about preferred prompting techniques, we decided to perform experiments on the same example models with different prompting techniques. In our experiments, we varied the level of detail of the task description, sometimes dissecting one model-deepening task into multiple, sequential prompts.

4 Exploratory Studies on Model Deepening with ChatGPT and FMML^x

The following discussion of experiences and results focuses only on selected examples and synthesized impressions. The complete documentation of all ChatGPT interactions, along with the used input models and generated results, is provided at www.le4mm.org/AutoMLM. The example IDs used in this paper can be looked up in the documentation for more detailed information.

4.1 Pure Generalization Studies with ChatGPT and FMML^x

Detailed Outline of Pure Generalization Studies. To conduct pure generalization studies, we designed three GPT interaction processes, which we refer to as *high-granularity process* (HGP), *mid-granularity process* (MGP), and *low-granularity process* (LGP). The three prompt interaction processes resemble different implementations of the chain-of-thought prompting technique. Each process was initiated by providing the GPT a two-level FMML^x model in XML format. In the LGP, the GPT is only prompted to add a generalization to a given model snippet. In the MGP, the GPT should perform required primary extensions first, i.e., adding potential new parent classes, and only afterwards promote class properties as required. At last, it is prompted to extend the new abstractions with domain-specific elements. The HGP consists of seven successive prompting steps:

1. Perform primary extensions (i.e., new, at first empty, parent classes)
2. Suggest names for the newly added classes and choose one
3. Identify promotion candidates by comparing all class properties with one another
4. Promote all user-approved promotion candidates to the new parent class
5. Remove all promoted properties from the child classes
6. Suggest extensions to the solution model
7. Extend the solution model with all user-approved extensions

Each of the three prompting processes was executed by relying on a consistent protocol: each experimental run used exactly the same prompts in the same order. We designed eleven example model snippets well suited for pure generalization, referred to as "positive examples," and five model snippets ill-suited for pure generalization, referred to as "negative examples." We executed each of the three interaction processes for all of the example models, resulting in a total of 48 experimental runs. We designed reference solutions based on semantic assumptions. One example model snippet for pure generalization consists of two classes, `Lecture` and `Tutorial`. Some attributes in both classes are identical in name and type (e.g., `shortName: String`). Others differ in name (e.g., `end: Date` vs. `terminationDate: Date`) or type (e.g., `term: TermEnum` vs. `term: String`) but are intended to represent the same property. Most example models only require promoting attributes. Two positive and two negative example models require promoting associations, too. All example models for the pure generalization studies model different facets of a university domain. Identifiers of experimental runs consist of the chosen prompt interaction process (LGP, MGP, or HGP), a letter indicating whether it is a positive ("P") or negative ("N") example, and its number.

Synthesized Results. The subsequent report on gathered insights distinguishes between the primary extension, promotion, and secondary extension operations outlined in Sect. 2.3, at first only discussing the results of positive-example runs.

Primary Extension. In all positive-example runs, ChatGPT generated one or more new parent classes, each of which was abstract. Overall, 76% of these parent classes were given adequate names. In the remaining 24%, ChatGPT generated generic names, such as `BaseClass` (MGP.P-02) or `SuperClass` (MGP.P-03 s). This issue was particularly prominent in the MGP runs, where 73% of the generated parent classes were given inadequate names.

Promotion of Attributes. Attribute promotion worked reasonably well in most cases. 94% of positive-example runs contain appropriate promotion of some attributes, although that does not mean that ChatGPT performed all reasonable promotions. In these runs, ChatGPT promoted semantically similar attributes even if they had different names or types. The promoted attributes also consisted of an appropriate name with a well-suited, specific data type. For instance, consider example model P-04, which consists of three classes representing different kinds of curriculum modules. One class contains the attributes `ss: Boolean` and `ws: Boolean` (intended as abbreviations for summer semester and winter semester, respectively), one class the attribute `cycle: CycleEnum`, and one class the attribute `cycle: String`. The generated ChatGPT parent class in the LGP.P-02 run merges these attributes into `cycle: cycleEnum`. In 15% of positive-example runs, some promoted attributes were not removed from the child classes, resulting in redundancy in the model. In a few exceptions, ChatGPT – for unknown reasons – resorted to generic attribute names such as `newAttribute1`. It also occurred quite frequently (in 39% of positive-example runs) that, when generating a solution model, ChatGPT removed attributes or associations present in the original input model.

Promotion of Associations. Promotion of associations proved generally more difficult than the promotion of attributes. Association promotion was performed in neither of the two LGP runs and in only one MGP run. Only in HGP runs did association promotion work well for both example models.

Secondary Extension. Secondary extensions were only added when explicitly prompted by a user, as was done in all MGP and HGP runs. All extended attributes were typically adequate for the domain, such as the extension of the attributes `maxAttempts: Integer` and `evaluationMethod: String` to the parent class `GeneralExam` (HGP.P-03). Sometimes, however, ChatGPT also proposed redundant attributes such as `duration: Integer` when the attributes `durationInMinutes: Integer` and `lengthInMinutes: Integer` were already modeled (HGP.P-04).

Pure Generalization in Negative-Example Runs. ChatGPT generated pure generalizations in all negative-example runs. This led to some strange solution models. The example model N-03 contained the two classes `AcademicStaff` and `Exam`, which superficially seemed to share some common attributes such as `roomNr: String` or `id: Integer`. Although a pure generalization over these two classes

clearly represents no time-invariant abstraction, ChatGPT still generated a parent class `StaffAndExam` and promoted some attributes of the provided classes (LGP.N-03). It seems that ChatGPT has a tendency to approve any user request, resulting in ill-suited solutions for all negative-example runs.

Validity of Generated FMML^x Code. Valid FMML^x code was provided in most example runs (83%). In some cases, the XML elements were arranged in the wrong order (MGD.P-8) or an unspecified enumeration was used as a data type (HGP.P-2).

Duplication of Knowledge Base. As described in Sect. 3.2, we suspected that sometimes ChatGPT replicates parts of the models we provided our custom GPT as a knowledge base, although we explicitly prompted it not to. For this reason, we switched to domain-agnostic examples containing classes named `A` with attributes named `attributeA1` and so on. One of our domain-agnostic examples was replicated completely in run MGP.P-04. Here, the ChatGPT-generated solution included three classes `A`, `B`, and `C` with respective generic attributes. All information of the input model was merged into a single class unrelated to these three classes. This proves that custom GPTs do sometimes replicate the provided knowledge base. This may also be the cause why sometimes generic attributes and class names have been proposed by ChatGPT.

Comparison of Prompting Techniques. Using the HGP generally resulted in slightly improved results: in all positive-example HGP runs, attributes have been correctly promoted with an adequate name and often also a fitting type. Information from the original model was removed in 27% of positive-example HGP runs, as opposed to 36% in LGP and 55% in MGP positive-example runs. However, HGP runs more often include syntactical mistakes. In 74% of positive-example HGP runs, the generalization relationship was not provided in the final XML code – an error that only appeared in 14% of all other positive-example runs. Additionally, 45% of generated FMML^x files were invalid, whereas in LGP and MGP, only 9% of generated files were invalid.

4.2 Intrinsic Classification Studies with ChatGPT and FMML^x

Initial Struggle with Generating Multi-level FMML^x Models. At the initial start of our experiments with ChatGPT for intrinsic classification, we were surprised that ChatGPT no longer generated valid FMML^x files. Since we relied on concepts familiar from standard two-level modeling before, we assume the cause to be the newly introduced multi-level modeling concepts.

In attempting to overcome this issue, we developed a new FMML^x XML format, which we refer to as *Simple FMML^x*. Simple FMML^x XML files only include the bare minimum of required information. For example, we removed the possibility to specify advanced language features such as contingent-level classes [8], which have not caused any trouble in our pure generalization studies. Additionally, we tidied up the names of all XML elements and attributes. We

implemented a small application, called *Simple FMML^x Validator*[1] that transforms Simple FMML^x to standard FMML^x and vice versa. It can also validate a Simple FMML^x file against its defined schema.

With Simple FMML^x we were able to generate valid multi-level models at first sight, but soon noticed that it contained errors not caught by our validator. For example, it would generate multi-level models with direct instantiations of abstract classes. We were able to correct this issue by specifying integrity constraints that are also checked within our Simple FMML^x Validator tool.

Detailed Outline of Intrinsic Classification Studies. Our first exploratory studies on intrinsic classification focused on the type-object pattern also described in Sect. 2.4. In these studies, we wanted to explore using ChatGPT to detect type-object pattern candidates and determine, given a pattern candidate as input, whether it should be approved or declined as a model-deepening pattern. Additionally, ChatGPT was prompted to extend deep models with exemplary concretions and instantiations.

We constructed six flat models, each consisting of four to six classes with associations and attributes. Each example model represents a different domain and contains multiple type-object pattern candidates; we conducted experiments with six of these candidate patterns. Four of these patterns serve as negative examples, where the candidate pattern should be declined, and two as positive examples, where it should be approved. One positive example covers cars and their various models. The other positive example of a type-object pattern is included in an example model representing flight bookings. Here, a class `FlightConnection` serves as the 'type class' for particular flights, instances of the class `Flight`.

We performed four successive rounds of experiments with our custom GPT. In Round A, we provided all complete models to our custom GPT and asked whether the model contains any type-object pattern at all. In Round B, we provided the GPT a complete model but prompted it only to analyze user-defined pattern candidates. We did the same in round C, this time using a chain-of-thought prompting technique by asking successive questions about the pattern candidate. In Round D, we provided our GPT with multi-level solutions of the example models and prompted it to extend them with types and instances.

Note that in these intrinsic classification experiments, we do not use ChatGPT for executing promotion operations, as was previously investigated in the pure generalization studies. Promotion in the simple type-object patterns considered here can better be performed by a rule-based algorithm as described in 2.4.

Synthesized Results

[1] https://github.com/map-research/SimpleFmmlxValidator.

Round A: Detection of Type-Object Pattern. In all example models, ChatGPT detected 75% (12 of 16) of type-object pattern candidates. It correctly approved only the two positive examples.

Round B: Low-Granularity Analysis of Candidate Type-Object Pattern. ChatGPT correctly approved or rejected all explicitly provided type-object pattern candidates. In the case of a candidate type-object pattern between a class `Customer` and `Address`, it noted that "the association lives_at represents a data relationship (a customer resides at an address), not a classification or typing relation."

Round C: High-Granularity Analysis of Candidate Type-Object Pattern. In Round C, we prompted ChatGPT with the following three successive questions before asking for its final verdict on approving or rejecting a pattern candidate, here stated with the example classes `Car` and `CarModel`:

Q1 Could there be scenarios where, over time, an instance of `Car` might need to be linked to different instances of `CarModel`?
Q2 Could it ever be possible for a `Car` object to be associated with more than one `CarModel` object?
Q3 Could a `Car` object ever be associated with no `CarModel` object at all?

Affirmative answers to these questions indicate that re-engineering a pattern candidate as a multi-level solution should be avoided. For all negative examples, ChatGPT answered affirmatively to each question and, therefore, also correctly concluded that no model-deepening transformation should be executed.

For the two positive examples (`Car` and `CarModel` as well as `Flight` and `FlightConnection`), ChatGPT also confirmed Q1, providing plausible exceptions. For cars, it mentioned cases where vehicles are reclassified for local branding; for flights, it cited rescheduling or code-sharing for why a flight might change its connection.

In response to Q2, ChatGPT listed several realistic scenarios in which an instance of `Car` could belong to multiple instances of `CarModel` – such as different regional brandings – while also pointing out that each car typically has a single canonical model based on its vehicle identification number, making multiple car models for a particular car "practically unlikely." A similar line of argumentation is presented for question three for cars and car models, leading ChatGPT to suggest that cars and car models should be re-engineered as a multi-level model.

ChatGPT was more ambivalent about the candidate pattern between `Flight` and `FlightConnection`. It acknowledged that a flight may cover multiple flight connections (e.g., in multi-leg or code-share scenarios), or also no typical flight connection at all (mentioning one-off and charter flights as examples). Interestingly, although ChatGPT affirms all three questions, it still recommends re-engineering the pattern as a multi-level hierarchy and even proposes refining the model with additional classes, such as `FlightSegment`, to better handle such edge cases.

Round D: Secondary Extension of Deep Model. Our custom GPT was able to generate convincing example instantiations on L1 and L0 for car models and

flight connections. The generated multi-level model of our custom GPT for car models is displayed in Fig. 4.

5 Discussion: Prospects, Challenges, and Limitations of Using ChatGPT for Model Deepening

Using a Non-standard Software Language with ChatGPT. Since the FMML^x is a non-standard modeling language, we at first faced the challenge to enable ChatGPT to process and generate valid FMML^x files. During the pure generalization studies, this seemed fairly simple and unproblematic. Using standard XML notation, ChatGPT was able to generate valid FMML^x files most of the time.

However, as soon as we required the generation of FMML^x files which rely on ideas different from standard object-oriented modeling techniques, we encountered severe struggles – even though the XML syntax was exactly the same as before. This was a blunt reminder that LLMs like ChatGPT prioritize probability over normativity. Given that, in pre-training, ChatGPT probably encountered many UML and XML examples, it posed no problem to use a specific XML notation to represent flat, two-level models. However, this pre-training may also bias LLMs in the wrong direction when, e.g., principles from two-level modeling are considered superior (since more probable) to those from multi-level modeling.

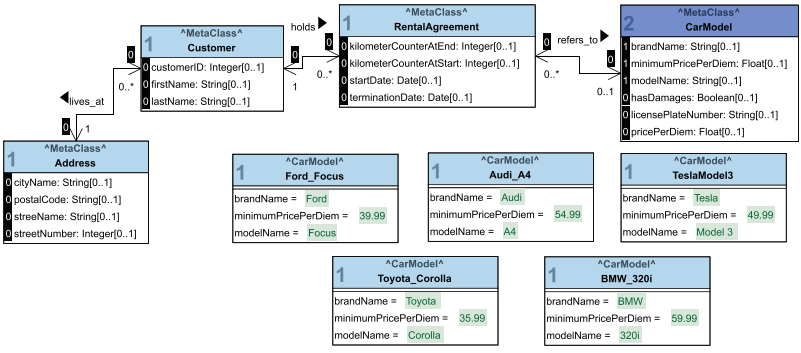

Fig. 4. ChatGPT-extended multi-level model for representing rental cars

A prospective solution may be to simply provide an LLM with sufficient example multi-level models upon which it can be trained or fine-tuned. But such a repository of high-quality multi-level models does not exist. A recent publication in Nature [16] discusses a comparable issue for geometric theorem solving, where the lack of training data historically prevented statistical learning approaches from being successfully applied. The authors use an LLM to generate the training data for another LLM. But since such an approach relies on a formal, verifiable evaluation of the synthetic training data generated by the LLM, it can hardly be applied to multi-level modeling.

Provision of Semantic Information. In many cases, ChatGPT was able to provide relevant semantic information that may assist in automatically analyzing flat models and, consequently, re-engineer them as deep models. In our pure generalization studies, ChatGPT was often able to correctly assess the semantic similarity of class properties. Comparisons which may seem simple to a human (e.g., to recognize that an attribute `ws: Boolean` in the context of universities may be an abbreviation of winter semester and, as such, similar to an attribute `semester: String`) are extremely difficult to detect by rule-based algorithms. Pre-trained language models can provide such semantic information without much effort required by a user.

Similarly, ChatGPT was able to provide valuable assistance in analyzing model-deepening pattern candidates. The example scenarios described by ChatGPT (e.g., car models with regional-specific names or multi-leg and charter flights) were even unknown to us when we designed the example models. This, to us, indicates a great prospect of using LLMs for model deepening. Whether or not these exceptions need to be accounted for in a specific information system must ultimately be decided by the system's developers.

Contingency of ChatGPT Interactions. While it is certain that the generated output of an LLM depends on the provided user input, no deterministic relationship between the two can be determined. Minor changes in the formulation of a prompt may yield different responses, and – even more drastically – the identical prompt may yield different responses when performed by a different user or on a different date. This causes two challenges when investigating LLMs. First, even if an LLM generates convincing results once, it may fail to do so again, leading to potentially inconsistent findings. Second, any results obtained may not be reproducible in the same manner by other researchers. To address this issue, we found it useful to reduce the scope of contingency by using more specific prompting techniques, as exemplified by the HGP prompting sequence in the pure generalization studies or the high-granularity analysis of pattern candidates in the intrinsic classification studies.

6 Conclusion and Future Research

In this paper, we report on a number of exploratory studies we conducted with ChatGPT to support the re-engineering of flat models into multi-level models. Our findings suggest that ChatGPT may provide powerful assistance to this re-engineering effort in some regards, but faces challenges and limitations in others. Semantic tasks that arise in model deepening, like the comparison of attribute meanings or the semantic analysis of type-object patterns, are greatly aided by ChatGPT. However, the use of ChatGPT may also result in syntactic errors, such as duplicate properties or invalid output models.

The utility of LLMs for model deepening is greatly influenced by the overarching re-engineering method employed. Our studies highlight that it is unwise to provide an LLM with a flat model and prompt it to generate a complete,

multi-level model, as this yields more inadequate solutions and an increased risk of certain syntactic errors, such as the arbitrary removal of model elements. A model-deepening process must account for intermediate steps where rule-based algorithms are preferable or human validation is required.

Much more research is required to provide effective and efficient support for LLM-assisted model deepening. For one, more model-deepening patterns need to be defined and embedded. This also includes investigating more complex cases of the type-object pattern. While the present studies only considered attributes and associations, future experiments should also account for multi-level constraints and operations. Ultimately, we envision the development of a hybrid model-deepening method that integrates rule-based and statistical approaches.

References

1. Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. Softw. Syst. Model. **7**, 345–359 (2008)
2. Bender, E.M., Koller, A.: Climbing towards NLU: on meaning, form, and understanding in the age of data. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 5185–5198 (2020)
3. Cámara, J., Troya, J., Burgueño, L., Vallecillo, A.: On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML. Softw. Syst. Model. **22**(3), 781–793 (2023)
4. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. The MIT Press, Cambridge (1998)
5. Fill, H.G., Fettke, P., Köpke, J.: Conceptual modeling and large language models: impressions from first experiments with ChatGPT. Enterp. Model. Inform. Syst. Architect. **18**(3), 1–15 (2023)
6. Frank, U.: Multilevel modeling: toward a new paradigm of conceptual modeling and information systems design. Bus. Inf. Syst. Eng. **6**(6), 319–337 (2014)
7. Frank, U.: Multi-level modeling: cornerstones of a rationale. Softw. Syst. Model. **21**, 451–480 (2022)
8. Frank, U., Töpel, D.: Contingent level classes: motivation, conceptualization, modeling guidelines, and implications for model management. In: Guerra, E., Iovino, L. (eds.) MODELS 2020: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (2020)
9. Frank, U., Töpel, D.: Association types: motivation, specification and implementation with the xmodelerml. In: MODELS Companion 2024: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, pp. 780–789 (2024)
10. Jeusfeld, M.A., Frank, U.: Unifying multi-level modeling: a position paper. In: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), pp. 536–540 (2021)
11. Macías, F., Guerra, E., de Lara, J.: Towards rearchitecting meta-models into multi-level models. In: Mayr, H.C., Guizzardi, G., Ma, H., Pastor, O. (eds.) Conceptual Modeling: 36th International Conference, ER 2017, Valencia, Spain, November 6–9, 2017, Proceedings, pp. 59–68. Springer, Heidelberg (2017)

12. Maier, P., Schwarz, T.: Uml++: enhancing student learning of object-oriented modeling through executable objects. In: MODELS Companion 2024: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, pp. 107–114 (2024)
13. Manning, C.D.: Human language understanding & reasoning. Daedalus **151**(2), 127–138 (2022)
14. Neumayr, B., Schrefl, M.: Domain object hierarchies inducing multi-level models. Softw. Syst. Model. **21**(2), 587–621 (2022). https://doi.org/10.1007/s10270-022-00973-7
15. OpenAI: Gpt builder (2024). https://help.openai.com/en/articles/8770868-gpt-builder
16. Trinh, T.H., Wu, Y., Le, Q.V., Luong, T.: Solving olympiad geometry without human demonstrations. Nature **625**, 476–482 (2024)
17. Wang, B., Wang, C., Liang, P., li, B., Zeng, C.: How LLMs aid in UML modeling: an exploratory study with novice analysts. In: IEEE International Conference on Software Services Engineering (SSE), pp. 249–257 (2024)