JOT
JOURNAL OF
OBJECT TECHNOLOGY

EMF-Kaizen: an intelligent assistant for domain-specific modelling and meta-modelling

Lissette Almonte, Jefferson Iván Rengifo, Esther Guerra, and Juan de Lara
Universidad Autónoma de Madrid, Spain

ABSTRACT Model-Driven Engineering (MDE) fosters the active use of models throughout the software construction process. Models are typically built using domain-specific languages, which are themselves described via a meta-model. While MDE yields advantages due to its possibilities for automation, modelling and meta-modelling remain complex tasks, which would benefit from intelligent assistants.

In this paper, we propose an assistive approach based on Large Language Models (LLMs) to help in the construction of both meta-models and domain-specific models. The approach works at the abstract syntax level, hence being independent of the (graphical, textual) concrete syntax of the modelling language. We have realised the approach as an intelligent assistant for EMF, called EMF-Kaizen. The assistant suggests model fragments that satisfy the user requests and can be incorporated to the model under construction (e.g., in the default tree-based modelling editor) via drag&drop. It also maintains and persists the modelling assistance chat sessions, together with the suggested model fragments. We report on: (a) an ablation study showing the rationale of each prompt component and assistant design decision; and (b) an evaluation of EMF-Kaizen for 120 completion tasks, over five meta-models (including Ecore) at different modelling stages and usage scenarios, repeated atop 4 LLMs. The results suggest the usefulness of the proposal, showing high syntactic accuracy, high semantic fidelity, and low redundancy.

KEYWORDS Model-Driven Engineering, Domain-Specific Modelling, Meta-Modelling, Conversational Assistants, Large Language Models, EMF, Eclipse.

1. Introduction

Model-Driven Engineering (MDE) promotes models as the main assets of software projects. They are used to specify, test, verify, simulate, and generate code for the final application, among other activities (Brambilla et al. 2017). Often, models are domain-specific, and so require building and using Domain-Specific Languages (DSLs) (Wasowski & Berger 2023).

MDE aims at simplifying software development, enabling even users with no technical background to perform specific programming tasks via modelling. Still, modelling can be hard due to complexities in the DSL, in the domain, in the system being modelled, or because of the explicit or tacit knowledge that high-quality modelling requires. Hence, intelligent modelling assistance – akin to assistants available for programming like GitHub Copilot[1] or Gemini Code Assist[2] – would be beneficial to help building models with higher quality more efficiently, especially for novice modellers.

Many approaches for modelling assistance have emerged in the last few years, especially with the advent of Large Language Models (LLMs). However, they suffer from one or several of the following limitations: target a single language (e.g., UML or Ecore) (Ben Chaaben et al. 2026; Cibrián et al. 2025), offer fixed kinds of assistance (e.g., recommending attributes for classes) (Ben Chaaben et al. 2026), work either for models or for meta-models but not for both (Saini et al. 2022), are limited to DSLs with a textual concrete syntax (Durá et al. 2024; Lamas et al. 2026), are not integrated within standard modelling environments (Cámara et al. 2023; Fill et al. 2023), or lack a

JOT reference format:
Lissette Almonte, Jefferson Iván Rengifo, Esther Guerra, and Juan de Lara. *EMF-Kaizen: an intelligent assistant for domain-specific modelling and meta-modelling*. Journal of Object Technology. Vol. 25, No. 3, 2026.
 http://dx.doi.org/10.5381/jot.2026.25.3.a8

[1] https://github.com/features/copilot
[2] https://codeassist.google/

conversational interface that facilitates their use (Ben Chaaben et al. 2026; Prokop et al. 2024).

To alleviate these limitations, in this paper, we propose a novel LLM-based assistive approach that is *conversational*, to enable flexible assistance requests; *agnostic to the meta-model* of the model for which assistance is requested, and thus applicable to arbitrary DSLs; *agnostic to the meta-level*, as it can assist in the creation and understanding of both meta-models and models; and *concrete-syntax independent*, as it works at the abstract syntax level and thus can be integrated with DSLs featuring graphical, textual or other forms of concrete syntax.

To make the approach available to a wide number of practitioners, we have realised it as an intelligent assistant for the *de-facto* (meta-)modelling standard Eclipse Modeling Framework (EMF) (Steinberg et al. 2009). The assistant is called EMF-Kaizen, and is available as an Eclipse plugin at http://EMFKaizen.github.io. EMF-Kaizen is conversational, helps in creating and comprehending EMF models and meta-models, and integrates off-the-shelf with the standard EMF tree-based modelling editor via drag&drop. It supports storage and retrieval of modelling assistance chat sessions, and reuse of model recommendations across different models. Currently, it can be configured to work with Gemini[3] and OpenAI[4] LLMs.

We have evaluated EMF-Kaizen based on five case studies that consider different meta-models, types of requests (creation commands, model modification commands, recommendation requests, and domain descriptive requests), and level of completeness of the model under construction. The case studies consider both meta-modelling (Ecore), and domain-specific modelling using structural and behavioural DSLs. The results show that the assistant suggests model fragments that are generally correct (up to 90.83%), semantically meaningful (up to 92.5%), and with low redundancy (up to 96.67% of non-redundant fragments). Moreover, we compare the effectiveness of EMF-Kaizen for four LLMs, and discuss trade-offs between assistance completeness, response time, and other factors. We also report on a lightweight ablation study that helps understanding the role of each prompt component and design decision. A replication package with the experiments data can be found at https://github.com/EMFKaizen/emf-kaizen-experiments.

Overall, this paper makes the following contributions: (i) *A conceptual contribution:* a novel approach to LLM-based modelling assistance that is agnostic on the DSL, meta-level and concrete syntax; (ii) *A technical contribution:* an Eclipse plugin that realises the approach atop EMF (to our knowledge, the first practical assistant with these features); (iii) *An empirical contribution:* a data-set of 120 modelling tasks over 5 meta-models; its evaluation over the tool atop 4 LLMs; and a study of the effect of different prompting strategies.

Paper organisation. Section 2 positions our work w.r.t. state-of-the art. Then, Section 3 presents the approach, and Section 4 describes the architecture and functionalities of EMF-Kaizen. Section 5 reports on the evaluations and Section 6 concludes.

[3] https://deepmind.google/models/gemini/
[4] https://openai.com/

2. Related work

Over the years, researchers have proposed ways to assist in various modelling activities (e.g., model completion, model generation, model finding, model repair, model reuse) using different techniques (from classical recommendation methods such as content-based or collaborative filtering, to model search or static analysis) (Almonte et al. 2022). These solutions provided the assistance via buttons or menu commands, making the interaction less natural and adaptable to the user's needs.

Recent advances in generative AI and LLMs have enabled AI assistants that understand natural language (NL) and have conversational interfaces to carry out arbitrary user requests. Since modelling is cognitively demanding, it can also benefit from intelligent assistants that provide modelling knowledge. However, building such modelling assistants is complex and faces open challenges, including the need *"to make functionality for modelling assistance reusable across a wide range of domains"* (Mussbacher et al. 2020). To improve the modelling experience and the quality of models, Mussbacher et al. (2020) envision conversational modelling assistants that interact using a vocabulary tailored to the modellers expertise and reason across multiple domains. Our proposal aims to advance this vision.

Most approaches to conversational assistance within MDE target the construction of domain models (i.e., meta-models). Early efforts relied on NL processing techniques, as is the case of RACE (Ibrahim & Ahmad 2010), SOCIO (Pérez-Soler et al. 2017) and DoMoBOT (Saini et al. 2022). However, the latest approaches are based on LLMs. Initially, to explore the potential of LLMs for domain modelling, researchers conducted ad-hoc experiments using various prompt strategies, e.g., via the console of ChatGPT (Arulmohan et al. 2023; Bragilovski et al. 2025; Cámara et al. 2023; Fill et al. 2023) or the API of OpenAI (K. Chen et al. 2023; Yang et al. 2024). These studies lack human interaction and integration with a modelling tool, but highlight the possibilities that LLMs can bring into domain modelling due to their understanding capabilities.

Only lately LLM-based meta-modelling assistants have gradually started to be integrated within modelling tools. For instance, Silva (2024) presents a prototype conversational bot that supports iterative human-AI collaboration for domain modelling, where the bot engages in dialogue to understand the user needs, and applies a Tree of Thoughts approach to explore alternative solutions and suggest relevant domain elements. Subsequently, Silva et al. (2025) propose a rule-based agent that creates a draft domain model from NL requirements via an LLM, and then refines it through a Q&A dialogue with the user.

Some domain modelling assistants use LLMs under the hood to avoid extensive training and domain-specific datasets, but are not conversational. Examples include MAGDA (Ben Chaaben et al. 2026) and the assistant by Prokop et al. (2024), both graphical tools that employ few-shot prompting and N-shot prompting, respectively, to suggest relevant classes, attributes and relationships for the domain model being developed. Compared to these works, our approach exploits the fact that meta-models are models conformant to a meta-model, and so it is directly applicable to both models and meta-models. Moreover, our assistant is

interacted via NL, which provides flexibility on the supported modelling tasks – i.e., the assistant is not limited to recommend attributes for a class – and the way to access to them.

Proposals focused on conversational modelling (as opposed to meta-modelling) assistance are less common. Like in the case of meta-modelling, earlier works often exploited NL processing to understand the user intention in an NL request. For example, given the meta-model of a DSL, some approaches generated intent-based chatbots that enabled users to edit (Pérez-Soler et al. 2019) or query (Pérez-Soler et al. 2020) models of the DSL through conversational interactions. However, intent-based chatbots are overly sensitive to how users express their intents, and require large amounts of training sentences to ensure an accurate recognition of the user intents.

For this reason, recent proposals leverage LLMs for model-level assistance. Some focus on model generation, but are not thought to be used as conversational assistants (B. Chen et al. 2025; Neuberger et al. 2024). Among those enabling interactive assistance, most assume a textual concrete syntax for models. For example, ModelMate (Durá et al. 2024) assists in identifier suggestion, line completion, and block completion of textual DSLs by fine-tuning pre-trained language models with model examples in their textual syntax; and DSL-Xpert (Lamas et al. 2026) translates NL instructions into the vocabulary of a textual DSL by prompting a general-purpose LLM using the DSL grammar and usage examples as context (i.e., it employs grammar prompting (Wang et al. 2023) and few-shot learning). Instead, we work at the abstract syntax level, and thus can integrate our assistant with both textual and graphical editors.

Other LLM-based assistants target a particular modelling language, like Text2VQL (López et al. 2024), which translates NL specifications into the VIATRA query language using fine-tuned models, or MCeT (Ahmed et al. 2025), which detects errors in sequence diagrams with respect to NL requirements via LLMs. Joel et al. (2025) survey techniques used for applying LLMs to specific (textual) low-resource languages and DSLs. Instead, our proposal is not constrained to a fixed language but is reusable across domains.

In summary, there is a lack of intelligent modelling assistants that can seamlessly handle both models and meta-models, operate at the abstract syntax level to support DSLs with any concrete syntax (graphical, textual, or tree-based), and can be used with different DSLs or domains. For flexibility, such assistants should have a conversational interface, and be available within the modelling environments practitioners use daily.

3. Intelligent assistance approach

To address the identified gap in the state-of-the-art, we propose an LLM-based conversational assistant for modelling and meta-modelling that is integrated off-the-shelf in the default EMF editors of Eclipse. EMF is an implementation of the OMG's Essential MOF (OMG 2016) widely used in practice. Other richer meta-metamodels, like Complete MOF or Semantic MOF, are not covered by our approach.

Fig. 1 shows the working scheme of our assistant. First, to customise the assistance for a (meta-)modelling language, there

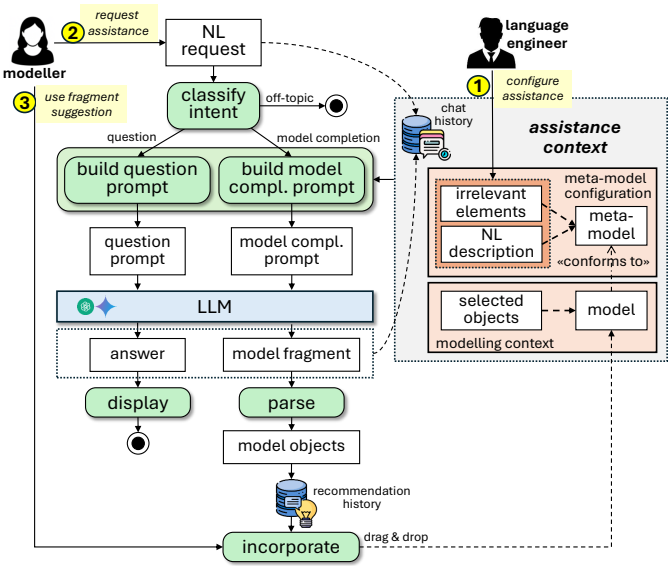


Figure 1 Working scheme of our assistance approach.

is an optional configuration phase where a modelling language engineer can filter irrelevant elements for the assistance from the meta-model, and provide a NL description of the meta-model, if so desired (label ①).

After configuration, modellers can request assistance in NL, e.g., employing utterances such as "*complete the traffic light state machine to consider pedestrian crossing*", "*translate the model into Spanish*", "*add support for complex tasks following the composite pattern*", or "*does the meta-model support multiple inheritance?*" (label ②). At this point, the assistant prompts an LLM to identify the modeller's intent, classifying the utterance either: (i) as a question about the current model, its meta-model, or modelling in general; (ii) a request to complete the current model; or (iii) an off-topic request otherwise.

In the first two cases, the assistant assembles a structured prompt, which depends on the identified intent. In any case, this prompt always contains the modeller's request and contextual information (*assistance context* in the figure), including the current model, its selected objects, its meta-model and configuration, and the conversation history. The assistant sends the prompt to an LLM, which produces a response. In the case of a question, the answer is simply presented to the modeller, while for model completion, the answer is parsed into a set of EMF model objects, and the modeller can integrate the parsed results into the model under construction via drag-and-drop (label ③). Assistance suggestions are persisted and can be reused in other models. Moreover, since meta-models are models conformant to a meta-model (e.g., Ecore), the same approach is used to assist in the creation of both models and meta-models.

The next subsections detail this process: initial assistance configuration (Section 3.1), user request classification (Section 3.2), prompt construction (Section 3.3), model serialisation and parsing (Section 3.4), and incorporation of model fragment suggestions into the model under construction (Section 3.5).

3.1. Assistance configuration

The language engineer can optionally adjust the meta-model information sent to the LLM in two ways. First, the engineer can filter out elements deemed irrelevant for the assistance. These often include volatile, derived and read-only (non-changeable) attributes and references, as they are not set explicitly but computed automatically from other elements. This filter can also be used to prune large meta-models to keep only the elements of interest, making prompts concise and focused. For example, in the case of UML, the engineer could filter its meta-model to leave just the elements corresponding to class diagrams, if this is the only diagram type subject of assistance. By this pruning, both the meta-model and the model under construction are sent to the LLM with the unwanted elements filtered out.

Second, the engineer can provide an NL description of the meta-model. This can help the LLM to understand better the purpose of the meta-model and its different parts, to produce better suggestions. To facilitate this step, our implementation has a facility that uses an LLM to automatically generate this description, which can be refined later.

Section 5.2 analyses the effect of both configuration options.

3.2. User intent classification

When a modeller requests assistance in NL, the assistant recognises the intent by prompting an LLM to classify the request in one of two categories: a *question*, if it demands information about the current model, its meta-model, or modelling topics (e.g., EMF); or *model completion*, if the aim is to obtain suggestions of model fragments that can be incorporated into the model under construction. Other kinds of utterances are considered off-topic, and the assistant replies by explaining its capabilities.

As we will explain in Section 3.3, all model completion requests are treated in the same way. However, they can take distinct forms, including:

- *Command-like* requests that explicitly describe the model elements to be created, like: "*create 10 states with names S1...S10*". These can be seen as NL creation *macros*.
- *Model-based command-like* requests asking to adapt the current model, like: "*translate the current model into Spanish*", or "*create an initial state for the selected composite state*". As the last example illustrates, such requests often require selecting objects from the current model, which serve as context to correctly interpret the request.
- *Recommendation* requests about missing or complementary details for existing model elements, like: "*recommend three more attributes for class Person*", or "*suggest more descriptive names for the selected tasks*".
- *Domain descriptive* requests of model fragments that capture high-level domain descriptions or requirements, like: "*create a subsystem for handling the human resources of the company*". These typically involve knowledge of the domain (e.g., human resources) or the modelling language (e.g., design patterns if using class diagrams).

3.3. Prompt construction

After identifying the user intent, the assistant assembles and sends a structured prompt to the LLM. The prompt contains a *common* part used for any request, and a *task-specific* part whose content and structure depend on the detected intent, either model completion or question.

Common part of prompt. Every prompt includes:

- *System prompt*: a brief description of the assistant's role and general interaction guidelines.
- *User input*: the user's NL request.
- *Active context*: the objects currently selected in the model, if any, to narrow down the request.
- *Current model*: the current model encoded in JSON, discarding elements of types filtered out in the configuration phase (details in Section 3.4).
- *Current model's meta-model*: the Ecore meta-model of the current model (pruned, if some elements were filtered).
- *Meta-model description*: NL description of the model's meta-model, if given in the configuration phase.
- *Conversation history*: previous assistant-user interactions, to preserve continuity in multi-turn interactions.

Specific data for model completion intent. For model completion, the following information is also added to the prompt:

- *Model completion prompt*: an instruction for the LLM to generate only valid model fragments that conform to the meta-model.
- *Example model*: one or several synthetic models that jointly cover all meta-model elements but the filtered ones, avoiding multiple objects of the same type for conciseness. The example model is represented in JSON (details in Section 3.4), and illustrates the correct formatting and conformance with the meta-model schema. It is particularly useful when the current model has few elements, as the example helps the LLM understand the expected structure of models for all meta-model elements. In most cases, a single synthetic model suffices, but multiple ones are needed when the meta-model contains unconnected classes or lacks a root class. More in detail, the method for example model generation receives a meta-model and the list of user-selected (i.e., non-filtered) classes and features. This method first orders the user-selected classes: those not contained in others come first, followed by the rest of classes in descending order of the number of classes they contain. This is a heuristic to reduce the number of generated example models. Then, the method iterates over the ordered classes, creating a new object of each class not sampled yet, and populating its features. Each such object will be the root of a new example model. To populate an attribute, a default value is assigned if the attribute is user-selected, changeable, and not derived. To populate a containment relation, the method checks that it is changeable, not derived, not yet sampled, and both the reference and its target type are user-selected; only then is an object of the containment target type created, and its attributes and containments recursively populated. To populate a non-containment reference, the method checks the

same conditions as for containments, and if they hold, an existing object of the proper type is assigned to the reference.
– *Response guidelines*: an instruction prompting the LLM to reply with only a JSON fragment (i.e., a model fragment) conformant to the meta-model, without comments, explanations, or markdown formatting. Thus, the LLM's response should be a (partial) model in JSON format, which will be parsed into a set of XMI objects.

Specific data for question intent. For answering a question, the common prompt is added the following:

– *Question prompt*: an instruction for the LLM to behave purely as an explanatory assistant and produce NL explanations only.
– *Response guidelines*: explicit constraints forbidding JSON or model generation. This ensures that the response remains descriptive and analytical, focused on explaining modelling concepts or interpreting the current model, and can be presented as-is to the modeller.

3.4. (Meta-)model serialisation and fragment parsing

Our approach uses JSON as the exchange format for the model fragments passed to and produced by the LLM. The rationale is to increase the robustness of the assistance. Even if using precise prompt instructions and structured outputs, LLMs can still provide invalid outputs (Lu et al. 2025), and so, a direct use of XMI may result in model suggestions with syntactic errors (e.g., non-existing attributes or references, attribute values of incorrect types) and semantic errors (e.g., dangling references, incorrect XMI:id identifiers, instances of abstract classes). Using JSON as an intermediate exchange format enables a flexible, error-tolerant parsing process that fixes incoherent information and discards erroneous data from the JSON, like non-existing field names. In contrast, asking the LLM to produce models directly in XMI hinders this fault-tolerance, being more rigid.

However, while our approach relies on JSON to represent models for flexible parsing, it uses XMI to represent meta-models within prompts. The reason for this is that meta-models are read-only, solely needed to inform the LLM of the available types and their properties.

Using JSON as the models' format also permits representing containment references naturally (nested dictionaries). For non-containment references, objects in our approach need to have a unique identifier. Technically, references in XMI can be stored in several ways, including *positional paths* (e.g., #//@items.0/@subItems.3 is a reference to the fourth object in reference subItems of the first object in reference items). However, positional paths may be error-prone for LLMs due to the need to count object positions, and preclude referencing objects in the current model from a recommended model fragment, or uniquely identifying the objects selected in the current model. Instead, our approach relies on *unique object identifiers*: it automatically serialises all models using XMI:id unique identifiers, and includes them in the JSON serialisation.

Fig. 2 depicts the model serialisation and parsing processes. For serialisation, the meta-model is first pruned from the elements marked as irrelevant in the configuration phase. Each

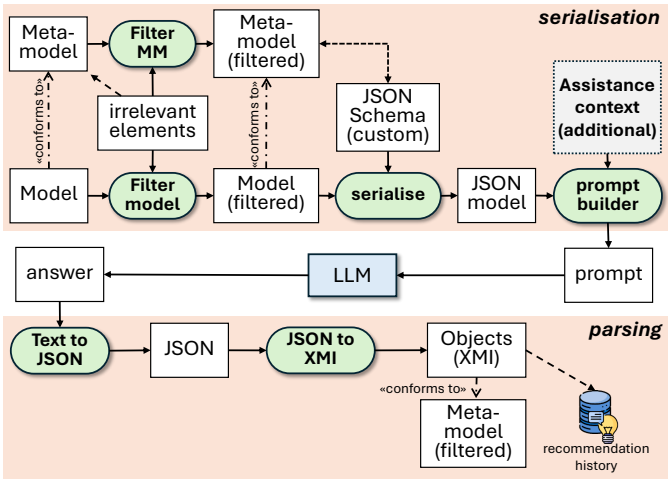


Figure 2 Schema of model serialisation and parsing.

pruned meta-model defines an implicit custom JSON schema, by which objects are serialised as dictionaries with a key-value pair per attribute, a key-value pair to store their type, and another to store their XMI:id. Moreover, containment references are represented as nested dictionaries, and non-containment references use the XMI:id of the target objects. The current model, as well as the example models in case of model completion requests, are persisted using this JSON-based format. If the meta-model includes OCL constraints (in EMF given as annotations), these are serialised in the prompt. However, we currently do not validate the response model fragments against the OCL constraints, which is left as future work.

Parsing the models included in LLM's responses proceeds in two steps (cf. bottom part of Fig. 2). Since the LLM's response is text, the first step is to locate and extract the JSON part. This might not be a fully-compliant JSON document; for example, if a set of objects is requested, they may appear as comma-separated JSON dictionaries, which is not a proper JSON document. Although the LLM typically wraps the requested objects in a container object (e.g., if the modeller requests EClasses, these are placed into an EPackage), this might not be possible if the pruned meta-model lacks the appropriate containment references. Hence, the first step flexibly parses the JSON text within the LLM's response into a collection of JSON dictionaries. Our approach does not apply a Schema validation to these extracted JSON dictionaries, but a resilient parsing process. This way, a second step parses the JSON dictionaries into EMF objects, instances of the meta-classes in the meta-model. This second parsing ignores extraneous key-value pairs that do not correspond to features in the meta-model, or types that do not exist in the meta-model. Still, the result may be a partial model not fully conformant to its meta-model. For example, the parsed objects may lack a container (to be assigned when dropped into the model under construction) or miss mandatory references. Anyhow, these may be valuable recommendations that the modeller can use and complete as needed. In Section 5, we will see that parsing frequently yields complete model objects in practice.

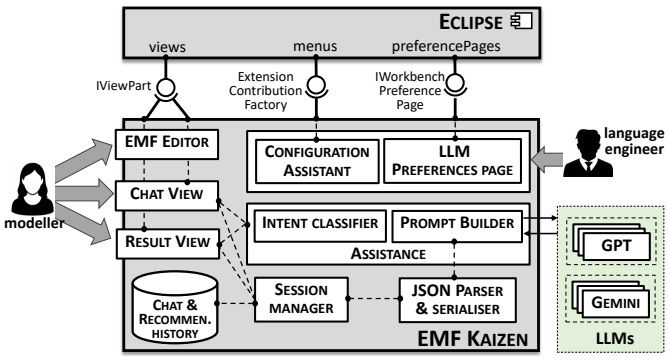


Figure 3 Architecture of EMF-Kaizen.

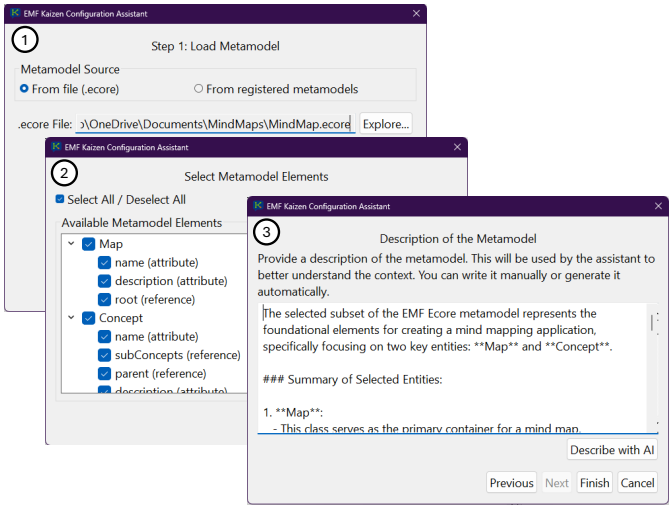


Figure 4 Configuration Assistant. (1) Meta-model selection. (2) Selection of relevant meta-model elements. (3) Configuration of NL meta-model description.

3.5. Model fragment incorporation

Once parsed into objects, the recommended model fragments are stored in a *recommendation history* where they can be inspected and, if appropriate, incorporated into the current model. Recommendations can be incorporated either fully or partially; in the latter case, the incorporated objects may need to be filtered to avoid issues, such as dangling references to objects that are not incorporated. All objects incorporated from recommendations are properly cloned, allowing the same recommendation to be reused multiple times in the same or different models.

As we will see in Section 4, the recommendation history is persisted so that it can be reloaded in future modelling sessions. The history includes not only the suggested model fragments, but also the chat sessions originating them. .

4. Tool support

We have realised our proposal in a conversational modelling assistant called EMF-Kaizen. It is an Eclipse plugin that integrates seamlessly into the EMF modelling workflow. The update-site and installation instructions are available at https://emfkaizen.github.io/.

EMF-Kaizen helps in completing (meta-)models, producing (meta-)model fragments, and explaining existing EMF artefacts in response to NL requests. It supports multiple LLM providers (currently, any model from OpenAI GPT and Google Gemini) and the configuration of their parameters (e.g., temperature, top-p sampling, maximum output tokens, reasoning level). This way, users can shape the style, determinism, and depth of the generated responses.

Fig. 3 shows the architecture of EMF-Kaizen and its integration within Eclipse. It is made of three subsystems:

1. *Configuration* (Section 4.1): It is contributed through Eclipse menus. It comprises a *Configuration Assistant* and an *LLM Preferences Page*, which the language engineer can use to configure the meta-model information sent to the LLM, as well as the used LLM itself.
2. *Assistance* (Section 4.2): This subsystem is exposed to the modeller through the *Chat View* and the *Result View*. It processes the modeller's requests by classifying their intent, constructing structured prompts, and sending them to the selected LLM. The model fragments returned by the LLM are parsed, transformed into EMF objects, and displayed in the *Result View*, where the modeller can incorporate them (fully or partially) into the current model via drag&drop in the EMF tree-based modelling editor.
3. *Session management* (Section 4.3): It manages the persistence and loading of the modelling assistance sessions.

4.1. Configuration subsystem

EMF-Kaizen can be customised by means of two complementary components: the *Configuration Assistant* and the *LLM Preferences Page*.

The *Configuration Assistant* allows the language designer to configure the meta-model information sent to the LLM. This enables EMF-Kaizen to scale from small domain-specific meta-models to large ones such as UML (cf. Section 5.2), ensuring that the LLM receives concise and meaningful contextual information. Fig. 4 illustrates the configuration process, which comprises three steps (labelled in the figure):

- *Step* ①*:* The engineer selects the meta-model, which can be loaded either from a local *ecore* file or from the meta-model registry of Eclipse.
- *Step* ②*:* The assistant displays all concrete classes and all changeable, non-derived attributes and references from the selected meta-model. The engineer can then choose the relevant ones (all by default).
- *Step* ③*:* The engineer can provide a NL description of the meta-model, or generate one automatically and then modify it if so desired. This description will be added to the LLM prompts to provide context.

The *Preferences Page*, shown in Fig. 5, allows configuring the behaviour of the LLM by selecting: the LLM provider to be used for the assistance, either OpenAI GPT or Google Gemini (① in the figure); the API key, which must be validated before it can be used (②); and a predefined LLM configuration profile that sets a default value for the LLM parameters (③). Exam-

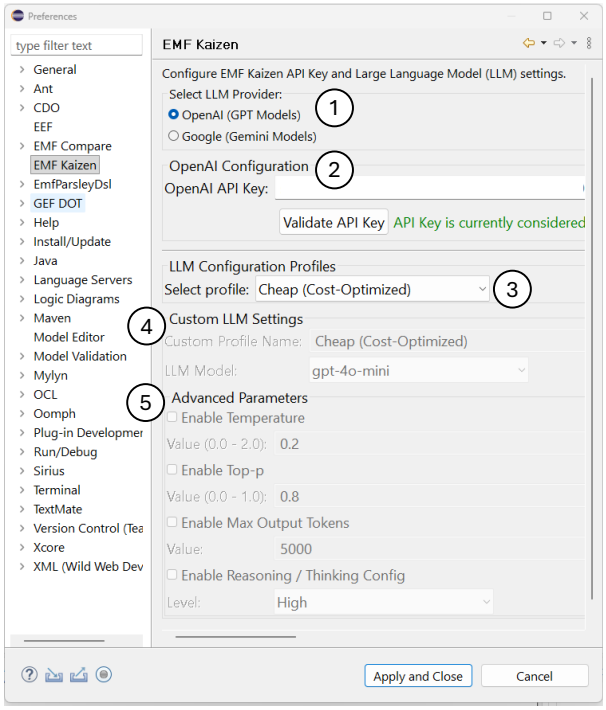


Figure 5 LLM Preferences Page.

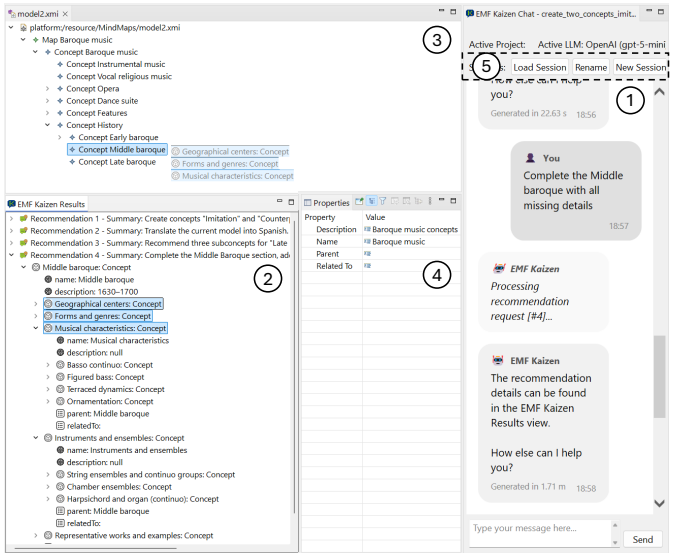


Figure 6 EMF-Kaizen perspective. (1) Chat View. (2) Result View. (3) EMF modelling editor with a Mind map model of Baroque music. (4) Properties View. (5) Session handling.

ples of predefined LLM configurations include cost-optimised, deterministic, and optimised for reasoning. Alternatively, the engineer can define a custom profile (④) by providing a name, the LLM version to be used, and the value of the LLM parameters, which may depend on the language model (⑤). We support the most common parameters: temperature, top-p value, maximum number of tokens, and level of reasoning, when supported by the model.

4.2. Assistance subsystem

Once the meta-model and the LLM are configured, modellers can start using the assistant. EMF-Kaizen offers an Eclipse perspective that organises the workspace into four areas, displayed in Fig. 6: the *Chat View* (label ①), the *Result View* (②), the EMF tree-based modelling editor (③), and the *Properties View* (④). Then, the typical assistance workflow proceeds as follows:

- *Step 1:* The modeller uses the *Chat View* to issue a modelling request in NL, such as "*complete the Middle baroque concept with all missing details*". EMF-Kaizen processes this request using the model opened in the modelling editor and its selected elements as context.
- *Step 2:* Once the request is processed, the *Chat View* informs the modeller. If the request is a query, the *Chat View* shows the answer; if it is a model-completion request, the *Result View* is updated to display the recommended model fragment. Model recommendations are presented in the *Result View* using a tree-based format, and include new objects, attribute values, and references.
- *Step 3:* The modeller can add the recommended model fragments, or parts of them, to the current EMF model via drag&drop. Selecting a recommendation highlights the corresponding request in the *Chat View* and vice versa.
- *Step 4:* If necessary, the modeller can modify the properties of the incorporated model objects in the *Properties View*.

EMF-Kaizen also assists in creating Ecore meta-models. Fig. 7 shows its use to complete a meta-model for Petri nets. Upon the request "*finish the meta-model for Petri nets, considering time in transitions*", the assistant creates three classes (Place, Transition, Arc) with their attributes and references.

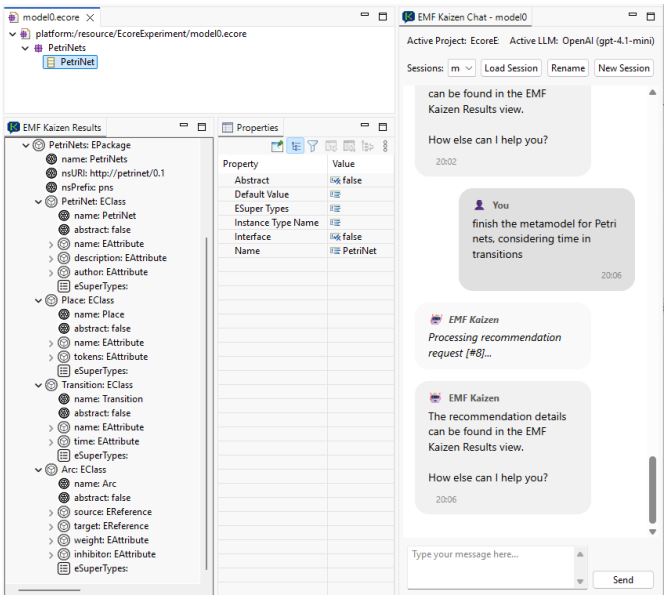


Figure 7 Building an Ecore meta-model with EMF-Kaizen.

4.3. Session management subsystem

The session manager of EMF-Kaizen persists the assistance sessions including both the assistant-modeller conversations

and the recommendation results. Modellers can rename and reload the persisted sessions, and create new ones (cf. label ⑤ in Fig. 6). The modelling sessions are persisted in JSON, and so, their loading requires parsing the fragment recommendations into EMF objects for their display in the *Result View*.

This functionality enables long-running modelling workflows and supports the traceability and retrospective analysis of design decisions. Additionally, each request is timestamped and stored as part of the session, which further contributes to the transparency and traceability of the assistance.

5. Evaluation

Next, we aim to answer the following research questions (RQs):

RQ1 How effective is EMF-Kaizen in providing (meta-)modelling assistance?
RQ1.1 What is the effect of different LLMs?
RQ2 What is the effect of the different parts of the prompt?

These RQs are important to assess the value of the assistant (RQ1), to understand the trade-offs of LLMs with different capabilities and costs when used by EMF-Kaizen (RQ1.1), and to assess the effect of the different parts of the prompt (e.g., the synthetic examples) and configuration options (NL meta-model explanation, meta-model pruning) (RQ2). Even if future LLMs may yield other results, RQ1.1 is necessary because evaluating the approach with a single LLM would make the results difficult to separate from the characteristics of that model. By considering multiple LLMs, we aim to show that the approach generalises across model families rather than being tied to a particular LLM.

In the following, Sections 5.1 and 5.2 present the design and result of the experiments, and answer the RQs; Section 5.3 discusses the results; and Section 5.4 analyses threats to validity.

5.1. RQ1, RQ1.1: EMF-Kaizen effectiveness, LLM effect

We first conducted an offline experiment to analyse the effectiveness of EMF-Kaizen across different domains, modelling stages, types of NL requests, and LLMs.

5.1.1. Experiment design We considered the following dataset in our offline experiment:

- As case studies, we selected 5 meta-models in different domains that jointly cover structural and behavioural aspects, and work at the meta-model and model levels (cf. Table 1).
- We created 3 models per meta-model, each on a degree of completeness, from almost empty models to detailed ones. These represent different stages at which the assistant may be used.
- For each model completion request category in Section 3.2 (command-like, model-based, recommendation, domain-descriptive), we designed 2 NL requests per model. This amounts to 120 requests.
- We selected 4 LLMs for the experiment: *gemini-2.5-flash-lite* (from Google), *gpt-4o-mini*, *gpt-4.1-mini* and *gpt-5-mini* (from OpenAI). This covers different vendors and includes a reasoning model (*gpt-5-mini*).

We used EMF-Kaizen to obtain model fragments out of the NL requests, and evaluated the fragments as follows:

1. *Syntactical correctness*: EMF-Kaizen supports flexible parsing, able to fix or recover from incorrect or malformed JSON responses from the LLM. Here, we measure the degree in which the parsed fragment – once converted into XMI – is syntactically correct, it misses mandatory attributes or references, the answer is not a model fragment, or the assistant does not respond.
2. *Semantical meaningfulness*: we evaluate whether the provided fragment is semantically coherent with the request.
3. *Fragment redundancy*: we assess the degree of redundancy of the provided fragment with the current model (i.e., whether the fragment contains elements repeated from the model that are unnecessary).
4. *Fragment size*: we record the size of the fragment, as a measure of completeness.
5. *Time*: we register the time taken by the LLM to respond the assistance request.

We assessed categories 1, 2, and 3 in the previous list by inspection. We set a protocol, where three authors evaluated each request and resolved disagreements. For semantical meaningfulness, we followed the protocol proposed in (Cibrián et al. 2025) to evaluate a SysML modelling assistant.

Table 1 shows the size of the 5 case studies. We used Ecore to work at the meta-model level and assess EMF-Kaizen as a meta-modelling assistant. Then, we selected meta-models to represent structural models (mind-maps, wizards) and behavioural ones (state machines, process models). The meta-model for mind-maps supports hierarchies of concepts. The meta-model for wizards permits defining application wizards made of pages with widgets, and navigation between pages. The meta-model for state machines considers hierarchical states, events, timed events, guards and actions, and features multiple inheritance. The meta-model for process models supports 3 gateway types (parallel split, multiple choice, exclusive choice) and 2 synchronisation types (parallel synchronisation, simple merge).

Table 1 Size of meta-models, and configuration complexity.

	Original			Configuration			
Meta-model	**#Class**	**#Atts**	**#Refs**	**#Class**	**#Atts**	**#Refs**	**#Desc.Words**
Ecore	20	33	48	13	15	9	371
Mind-maps	2	5	3	2	5	3	266
Wizards	9	12	7	9	12	7	367
State machines	8	8	7	8	8	7	366
Process models	11	5	4	11	5	4	316

Table 1 also shows the initial assistance configuration. We only pruned the Ecore meta-model, selecting 13 classes (out of 20), 15 attributes (out of 33), and 9 references (out of 48). We used the LLM-based facility of the configuration wizard to automatically generate a NL description for the meta-models, which we adjusted manually. The table shows the size of such descriptions, ranging from 266 to 371 words. Except for Ecore, we created the meta-models from scratch to avoid data contamination (i.e., to ensure that the training sets of the LLMs did not contain the meta-models of the case studies).

Table 2 describes the 15 models used for the evaluation. For each case study, they are at three levels of completion: initial (almost empty), medium, and complete (or with few missing details). The table shows the number of objects of each model. They cover different modelling styles: structural (Ecore), knowledge-based (mind-maps), data collection (wizards), and behavioural (state machines, process models). Again, to avoid data contamination, we built the models from scratch.

Table 2 Description and size of models used.

Model	Description	Size
Ecore-0	Meta-model of a Petri net (initial)	2
Ecore-1	Meta-model for goal models (incomplete)	10
Ecore-2	A meta-model for a tourism application (complete)	55
Mind-map-0	Mind-map for AI and LLMs for education (initial)	2
Mind-map-1	Mind-map for healthy living (mid-completion)	12
Mind-map-2	Mind-map for baroque music (some details missing)	29
Wizard-0	Wizard to customise LLM requests (initial)	2
Wizard-1	Wizard to set up a user account (mid-completion)	10
Wizard-2	Onboarding data collection (some details missing)	21
State-machine-0	State machine for traffic light (empty)	1
State-machine-1	State machine for media player (mid-completion)	10
State-machine-2	State machine for an ATM (some details missing)	22
Process-model-0	Paper publication process (initial)	2
Process-model-1	CI/CD pipeline (mid-completion)	10
Process-model-2	Order fulfilment (some details missing)	20

Table 3 shows some representative requests used in the experiment. For brevity, we show 1 request per category for each case study. Some requests involved the selection of elements in the model, which we indicate in italics between brackets. Most requests are not trivial and require the assistant to have good "understanding" of the meta-models and models involved.

Overall, this dataset contains a mixture of meta-models and models covering domain modelling (Ecore, mind-maps), behavioural modelling (state machines, process models), and domain-specific modelling (wizards). Moreover, the requests check syntactical and semantical meta-model understanding, understanding of the current model, and domain-specific knowledge (e.g., Baroque music, CI/CD pipelines). The meta-models, initial models, requests, and assistance sessions are available at https://github.com/EMFKaizen/emf-kaizen-experiments.

5.1.2. RQ1: Effectiveness of EMF-Kaizen

Table 4 shows the results, aggregated by LLM and case study. The columns display three of the evaluated dimensions: syntactical correctness, fragment redundancy, and semantical meaningfulness. For syntactical correctness, 1 indicates syntactically correct, 2 indicates missing mandatory attributes, 3 missing mandatory references, and 4 incorrect intent (the assistant discarded the request as non-relevant or did not provide a model fragment). For redundancy, 1 means no redundancy, 2 means that the provided fragment includes repeated objects from the model unnecessarily, and 3 is for the case where all the model is repeated inside the fragment unnecessarily. For semantic meaningfulness, 1 means low fidelity, 2 partial fidelity, and 3 high fidelity, as in (Cibrián et al. 2025). All numbers in the table indicate percentages. The best values for each LLM are highlighted with a green background.

Most fragment recommendations were syntactically correct (between 82.5% for *gemini-2.5-flash-lite* and 90.83% for *gpt-5-mini*). The syntactical errors were minor, mostly missing attributes or references. Of these latter errors, most were due to a limitation of EMF-Kaizen: fragments that directly referenced objects in the model, which are filtered out as this is disallowed[5]. Only *gpt-4o-mini* had intent classification errors (11.67%), mainly when handling the request "*Translate the current model into Spanish*", classifying it as off-topic.

Table 3 Sample of NL requests. Labels for Category (Cat.): c=command-like, m=model-based, r=recommendation, d=domain-descriptive.

	Cat.	Request
Ecore	c	create two classes with names Country and City, where Country has a containment reference "cities" to its Cities
	m	translate the current model into Spanish
	r	recommend more classes for the current metamodel
	d	finish the metamodel for Petri nets, considering time in transitions
Mind-maps	c	create a concept with 3 children concepts, and each children concept should have 2 children related to each other
	m	create a copy of the selected concepts, but in lower case [*selecting concepts timbre, texture, harmony*]
	r	recommend three subconcepts for the concept "Exercise"
	d	complete the mind-map with concepts that summarise the history of use of AI in education
Wizards	c	create a page with a button to display the text "setup completed!"
	m	add a summary field, and a close button to the selected page [*selecting the page "Summary and finish"*]
	r	recommend fields for each selected page [*selecting the four bottom pages*]
	d	complete the wizard to model an account setup for an on-line shop
State machines	c	create two states connected by a time-triggered transition
	m	create a composite state that contains the selected elements [*selecting all states and transitions*]
	r	recommend more substates for Operational
	d	create the states and transitions for a traffic light that has both lights for cars and for pedestrians, and that has time-triggered transitions
Process models	c	create three tasks, one of the tasks should have as output a ParallelSplit. The ParallelSplit should have as output the other two tasks.
	m	create a replica of the current process model, but where the three parallel tasks (inventory check, payment processing, ship preparation) are sequential
	r	recommend conditions for a multichoice gateway that goes after task "unit test"
	d	complete the process model with a typical CI/CD pipeline for a cyberphysical system that has a digital twin

Redundancy was very low, as most recommended fragments included no unnecessary objects repeated from the model under construction (between 84.17% for *gpt-5-mini* and 96.67% for *gemini-2.5-flash-lite*).

Regarding semantics, a substantial number of fragments had high fidelity, though the percentage varies depending on the LLM (from 59.17% to 92.5%). *gpt-5-mini* and *gpt-4.1-mini* produced no fragment with low fidelity, whereas *gpt-4o-mini* and *gemini-2.5-flash-lite* produced 13.33% and 6.67%, respectively. Looking at domains, the lowest percentages of high fidelity were obtained for Ecore (*gpt-4.1-mini* and *gemini-2.5-flash-lite*) and wizards (*gpt-5-mini* and *gpt-4o-mini*). This may be due to the higher complexity of their meta-models.

Fig. 8 breaks down semantic meaningfulness by LLM and request category. Generally, the easiest requests for all LLMs were those on the command-like category. These requests were concise and clear, but required a good understanding of the meta-model. Conversely, the most difficult requests were those

[5] This would lead to a cross-reference from the fragment to the model, which could become broken after modifying the model, a situation we want to avoid.

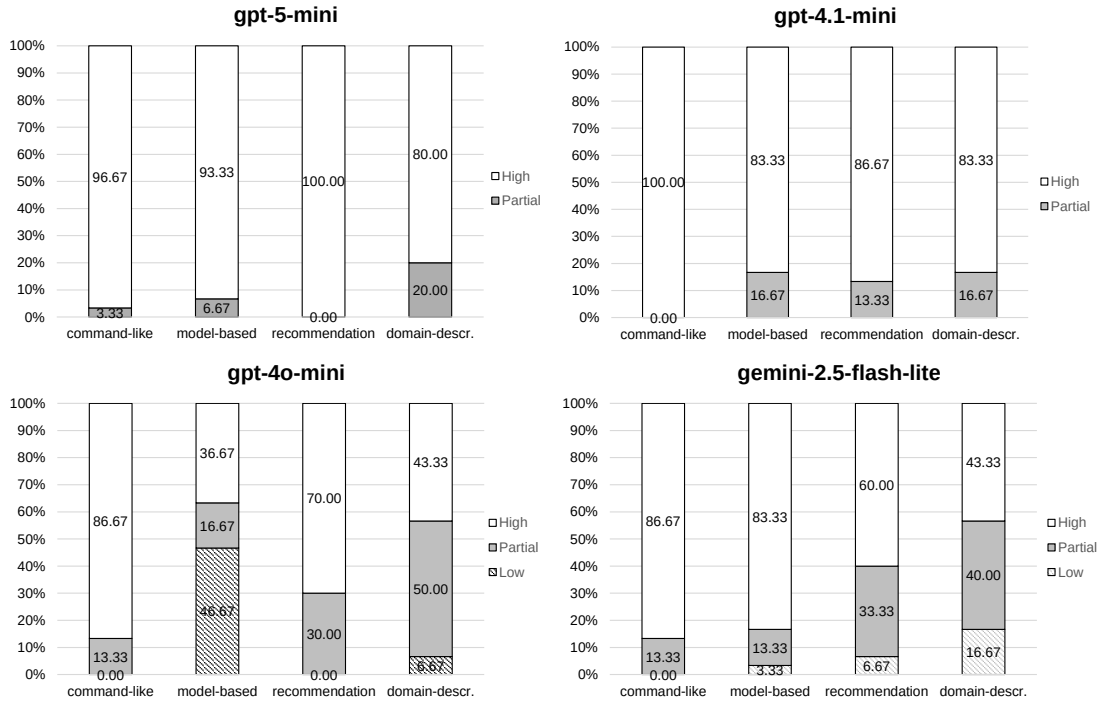


Figure 8 Semantic meaningfulness by LLM and request category.

Table 4 Correctness percentages by LLM and case study. Labels for Syntax: 1=OK, 2=Missing attribute, 3=Missing reference, 4=Incorrect intent. Labels for Redundancy: 1=OK, 2=Unnecessarily repeated elements, 3=All model repeated. Labels for Semantic Fidelity: 1=Low, 2=Medium, 3=High.

	Syntax				Redundancy			Semantic		
	1	2	3	4	1	2	3	1	2	3
	gpt-5-mini									
Ecore	75.00	0.00	25.00	0.00	95.80	0.00	4.20	0.00	4.17	95.80
Mind-maps	100.00	0.00	0.00	0.00	70.83	12.50	16.67	0.00	4.17	95.80
Wizards	83.33	8.33	8.33	0.00	87.50	8.33	4.17	0.00	12.50	87.50
State machines	100.00	0.00	0.00	0.00	79.16	0.00	20.83	0.00	8.33	91.67
Process models	95.83	0.00	4.17	0.00	87.50	0.00	12.50	0.00	8.33	91.67
Total	90.83	1.67	7.50	0.00	84.17	4.17	11.67	0.00	7.50	92.5
	gpt-4.1-mini									
Ecore	95.83	0.00	4.17	0.00	95.83	0.00	4.17	0.00	33.33	66.67
Mind-maps	83.33	0.00	16.67	0.00	100.00	0.00	0.00	0.00	8.33	91.67
Wizards	79.17	0.00	20.83	0.00	100.00	0.00	0.00	0.00	4.17	95.83
State machines	100.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	4.17	95.83
Process models	91.67	4.17	4.17	0.00	87.50	0.00	12.50	0.00	8.33	91.67
Total	90.00	0.83	9.17	0.00	96.67	0.00	3.33	0.00	11.67	88.33
	gpt-4o-mini									
Ecore	79.17	0.00	8.33	12.50	100.00	0.00	0.00	12.50	25.00	62.50
Mind-maps	95.83	0.00	0.00	4.17	91.30	4.34	4.34	4.17	20.83	75.00
Wizards	87.50	0.00	0.00	12.50	90.47	9.52	0.00	12.50	45.83	41.67
State machines	87.50	0.00	0.00	12.50	95.24	4.76	0.00	16.67	20.83	62.50
Process models	70.83	0.00	12.50	16.67	90.00	10.00	0.00	20.83	25.00	54.17
Total	84.17	0.00	4.17	11.67	93.40	5.66	0.95	13.33	27.50	59.17
	gemini-2.5-flash-lite									
Ecore	58.33	0.00	41.67	0.00	100.00	0.00	0.00	4.17	37.50	58.33
Mind-maps	100.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	12.50	87.50
Wizards	91.67	4.17	4.17	0.00	100.00	0.00	0.00	8.33	25.00	66.67
State machines	91.67	0.00	8.33	0.00	100.00	0.00	0.00	16.67	25.00	58.33
Process models	70.83	0.00	29.17	0.00	83.33	0.00	16.67	4.17	25.00	70.83
Total	82.50	0.83	16.67	0.00	96.67	0.00	3.33	6.67	25.00	68.33

on the domain-descriptive category, where all LLMs (but *gpt-4o-mini*) produced the lowest percentage of high-fidelity fragments.

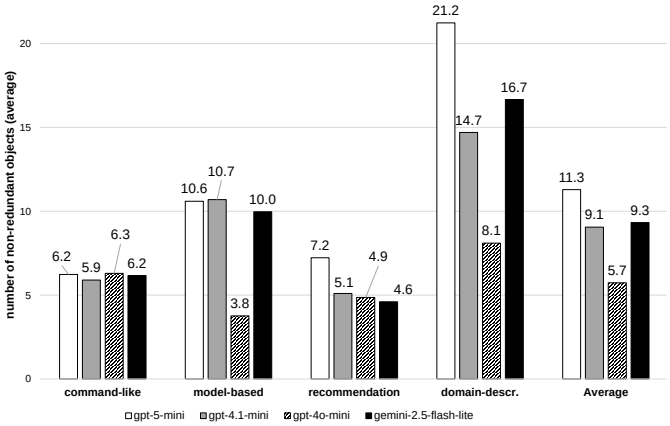
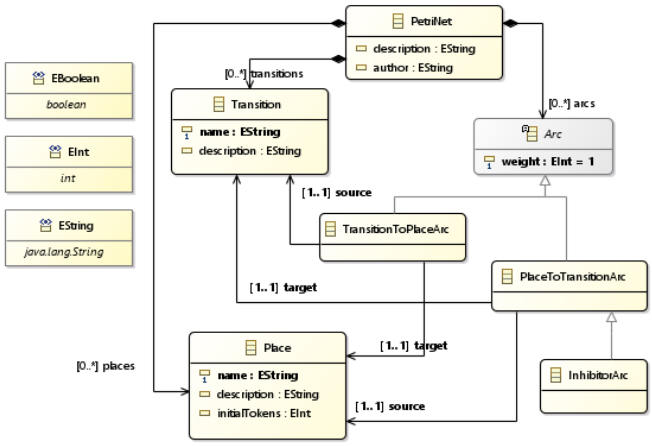


Figure 9 Fragment size by LLM and request category.

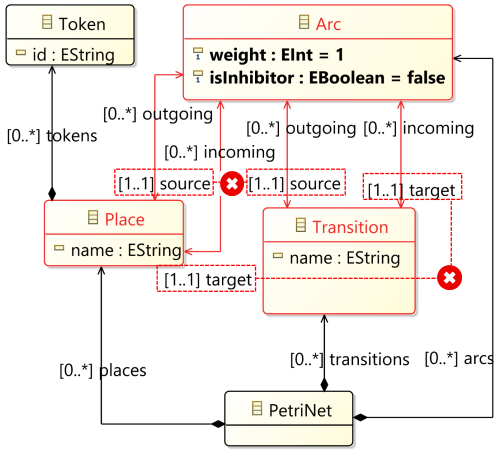
gpt-4o-mini obtained the lowest percentage of high-fidelity fragments for model-based requests, due to its inability to classify the translation of models into Spanish as a model completion request. Domain-descriptive requests resulted in the largest fragments, and therefore had more room for errors.

Fig. 9 displays the average fragment size by LLM and request category (redundant elements are filtered). Domain-descriptive requests yielded the biggest fragments, and command-like and recommendation requests the smallest ones. Still, the request leading to the biggest fragment was a model-based one asking to translate an Ecore model with 91 elements (EClasses, EAttributes, EReferences) into Spanish.

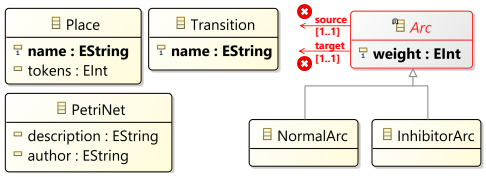
Answering RQ1. For the considered dataset, EMF-Kaizen achieved good syntactic correctness (82.5%–90.83% depending on the LLM), low redundancy (84.17%–96.67% of frag-



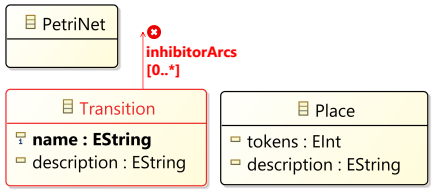
(a) gpt-5-mini [26 elements]



(b) gemini-2.5-flash-lite [21 elements]



(c) gpt-4.1-mini [15 elements]



(d) gpt-4o-mini [9 elements]

Figure 10 Fragments recommended by the LLMs for request "*complete this metamodel of Petri nets, with all missing details, and to consider inhibitor arcs*" (Ecore case study). (a) was classified as "high fidelity"; (b), (c) and (d) as "partial fidelity".

ments had no redundancy), and high semantic fidelity (59.17%–92.5%). Fragments with low semantic fidelity were seldomly recommended (0%–13.33%). Recommendations were generally of substantial size, especially for domain-descriptive requests. Overall, this suggests effectiveness of EMF-Kaizen for modelling and meta-modelling assistance.

5.1.3. RQ1.1: Effect of LLMs Table 4 showed that the LLM used has an impact on the quality of the generated fragments. Overall, *gpt-5-mini* achieved the best syntactical (90.83%) and semantical (92.5%) correctness, while *gemini-2.5-flash-lite* and *gpt-4.1-mini* produced the largest number of fragments without redundancy (96.67%). Likewise, Fig. 9 showed that, overall, *gpt-5-mini* produced the biggest fragments. To gain insight on the recommendations, next, we qualitatively compare the fragments produced by all LLMs for 2 requests.

First, we consider the request "*complete this metamodel of Petri nets, with all missing details, and to consider inhibitor arcs*" of the Ecore case study, where the initial model is *Ecore-0* in Table 2. Fig. 10 depicts graphically (with EcoreDiag) the result of incorporating the recommended fragment to the model via drag&drop, for each LLM. The models show their syntactic errors in red: in model (c), class Arc has two references (source and target) without target type; model (b) contains references without target, and incorrectly defined opposite references (Arc.source cannot be the opposite of both Place.outgoing and Transition.outgoing, and similar for Arc.target); and model (d) includes references without target. Regarding semantic correctness, model (d) lacks any form of inhibitor arcs, as requested. Hence, for this request, only *gpt-5-mini* produces a fragment without errors, and the most detailed. Notably, it also identified that inhibitor arcs are a special case of PlaceToTransitionArc (i.e., connecting places to transitions), while inhibitor arcs in the recommendations of *gemini-2.5-flash-lite* and *gpt-4.1-mini* can connect transitions to places as well.

Fig. 11 shows another example for state machines, where the assistant was requested to "*create the states and transitions for a traffic light that has both lights for cars and for pedestrians, and that has time-triggered transitions*" starting from an empty state machine. The solution of *gpt-5-mini* is the most complete, using composite states, time triggers and actions. The solution of *gpt-4.1-mini* is similar but lacks actions. Since the meta-model has no support for parallel regions, both solutions used composite states, but would need some kind of synchronisation to adjust times. Instead, the solutions (c) and (d) sequentialise the car and pedestrian states, and do not need any synchronisation.

Regarding times, Table 5 shows the seconds taken to serve the requests. For each request category and LLM, it shows the median (columns 2–5), overall median, minimum, and maximum time. The fastest model was *gemini-2.5-flash-lite*, and *gpt-5-mini* the slowest one. The latter was expected, as *gpt-5-mini* is a reasoning model. Looking at request categories, LLMs were slower for domain-descriptive requests, which produced larger fragments. These numbers provide an intuition of the median completion times of the LLMs in our context, but note that this is not a comprehensive evaluation of time performance, which would require a more detailed experiment with repetitions to account for the non-deterministic nature of LLMs.

Answering RQ1.1. The experiment shows a substantial effect

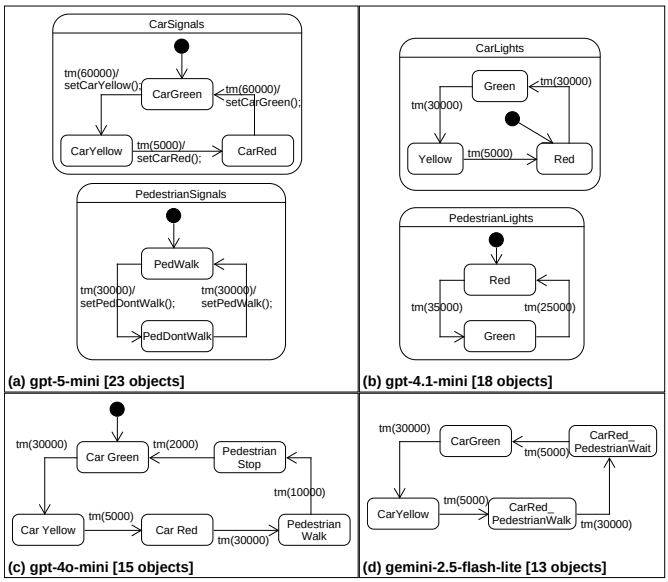


Figure 11 Fragments recommended by the LLMs for request "*create the states and transitions for a traffic light that has both lights for cars and for pedestrians, and that has time-triggered transitions*" (state machines case study). All fragments were classified as "high fidelity".

Table 5 Performance (in secs.) by LLM and request category.

Model	Comm.	Model Recomm.	Domain	Overall	Min	Max
gpt-5-mini	24.47	25.02	28.50	61.10	31.10	11.57	182
gpt-4.1-mini	5.42	5.83	4.93	12.75	6.29	1.87	82
gpt-4o-mini	10.03	2.97	7.46	11.39	8.31	0.55	52.43
gemini-2.5-flash-lite	5.52	5.73	6.39	7.82	6.11	1.83	27.71

on the LLMs used. We found that *gpt-5-mini* achieved the best results w.r.t. semantic accuracy and recommendation detail but, since it is a reasoning model, was the slowest one (by a factor of 5×). The second performing LLM was *gpt-4.1-mini*, with much faster times than *gpt-5-mini*.

5.2. RQ2: Effect of prompt structure and configuration

To understand the effect of the parts of the prompt (example models) and configuration options (NL meta-model description, meta-model pruning), we conducted an ablation study.

5.2.1. Experiment design

For this experiment, we used *gpt-4o-mini*, and considered 2 meta-models: adapters and UML2.

The meta-model for adapters was taken from (de Lara & Guerra 2025). An adapter is a rule-based component to transform models between variants of a language. Adapters may reference rules by name, but the rules themselves are defined separately. Language variants are defined via a feature model, and each adapter captures the feature changes (activation, deactivation) required for its application. We selected this very specialised, domain-specific meta-model to study the effect of including a NL meta-model description in the prompt.

The meta-model of UML2 has 256 classes (>17 700 XML LoC). We selected this large meta-model to assess the effect of meta-model pruning in the prompt. In this study, we were only interested in the class diagram part of UML2.

Similar to the experiment for RQ1 and RQ1.1, we created 3 models at different levels of completion (with one object, medium and complete), and 8 prompts per model.

5.2.2. Results

Table 6 shows the results for 4 prompt configurations: complete, without the NL meta-model description, without example models, and without pruning the meta-model.

Table 6 Correctness percentages by ablation option. Labels for Syntax: 1=OK, 2=Missing attribute, 3=Missing reference, 4=Incorrect intent. Labels for Redundancy: 1=OK, 2=Unnecessarily repeated elements, 3=All model repeated. Labels for Semantic Fidelity: 1=Low, 2=Medium, 3=High.

	Syntax				Redundancy			Semantic		
	1	2	3	4	1	2	3	1	2	3
	Complete prompt									
Adapters	100.00	0.00	0.00	0.00	100.0	0.00	0.00	8.33	50.00	41.67
UML2	33.33	0.00	66.67	0.00	100.0	0.00	0.00	4.16	33.33	62.5
	No NL meta-model description									
Adapters	100.00	0.00	0.00	0.00	100.0	0.00	0.00	33.33	41.67	25
UML2	33.33	0.00	66.67	0.00	100.0	0.00	0.00	8.33	29.17	62.5
	No example models									
Adapters	100.00	0.00	0.00	0.00	100.0	0.00	0.00	8.33	62.50	29.17
UML2	33.33	0.00	41.67	25	77.78	5.55	16.67	37.5	16.67	45.83
	No meta-model pruning									
Adapters	100.00	0.00	0.00	0.00	100.0	0.00	0.00	8.33	50.00	41.67
UML2	–	–	–	–	–	–	–	–	–	–

No NL meta-model description. NL descriptions are useful for meta-models in specialised domains or with non-obvious primitives. As Table 6 shows, removing the description decreases the percentage of suggestions with high semantic fidelity for adapters, compared to using the complete prompt. Without the description, the assistant failed to produce meaningful suggestions for requests like "*create an adapter named InhByDelegation applicable when 'Simple' is activated*", whereas including it, the assistant suggested a correct fragment. This effect was not observed for UML2, whose results were similar with or without the description, likely as it is a widely known meta-model.

No example models. The synthetic examples in the prompt illustrate how to encode model fragments in JSON. For adapters, when these examples are omitted and the current model lacks references between adapters, the assistant fails to encode references between adapters correctly: they should be encoded as an XMI:id string, but the assistant uses a dictionary or the adapter name. Although our flexible parsing fixes this issue, it results in an empty reference and a semantically incorrect model. This leads to a lower percentage of high-fidelity suggestions in both meta-models compared to the complete prompt. In UML2, in addition, some prompts were misclassified as non-relevant, and the suggestions were more redundant.

No meta-model pruning. Pruning reduces the meta-model to the parts of interest. Without it, the UML2 meta-model exceeds the context window of *gpt-4o-mini* (though it would fit in other LLMs, like *gpt-4.1-mini*). Since our interest was on class diagrams, pruning UML2 in the configuration enables the use of *gpt-4o-mini* for just this diagram type.

Answering RQ2. NL meta-model descriptions clarify domain-specific terms, example models illustrate the JSON encoding,

and pruning reduces large meta-models, which otherwise may not fit in the LLM's context window. Together, they contribute to improving suggestion fidelity while reducing the prompt size.

5.3. Discussion and lessons learnt

We next provide some discussion, useful for both practitioners using modelling assistants, and researchers working on LLM-based intelligent assistance.

Which LLM? We have seen that a reasoning LLM (*gpt-5-mini*) gives the best results but is the slowest. Hence, for simple requests, one can resort to faster LLMs, leaving reasoning LLMs for larger, complex requests. To automate this decision, it is worth exploring a dynamic selection of LLM given a request.

Prompt matters. In some cases, requests like "*recommend three tasks for the model"* led to uncontextualised, generic proposals of tasks with names *Task1*, *Task2*, *Task3* (e.g., when using *gpt-4o-mini*). Instead, the response improves by adding more context to the request, like "*recommend three tasks for the process model representing a publication process"*. For fairness, our experiment did not include such clarifications, but it is a lesson learnt for the future use of the assistant.

Identification of context. The latter issue occurs due to difficulties of the LLM in inferring the context domain from the current model. In addition, the elements selected in the current model are important for requests like "*recommend more fields for the selected page*". Our prompt explicitly mentions the selected elements, identified by their unique XMI:id. However, some LLMs still struggled to locate the selected objects in large models.

Fragment-model cross-references. Fragment recommendations may need to refer to objects in the current model. This happens in requests like "*complete the process model with a typical CI/CD pipeline...*", where (a) existing model objects need to reference objects in the provided fragment, and (b) fragment objects need to reference existing model objects. EMF-Kaizen currently has limitations in this respect: we only allow references between fragment objects, but fragment objects can be dropped in containment relations of model objects. We will work on this aspect in future versions of the assistant. Our idea is identifying *proxy* objects in the fragment (representing objects in the model), which would be automatically merged with the *real* object upon a drag&drop operation.

Multi-languages. EMF-Kaizen can handle one current model at a time, but not multiple ones simultaneously. However, if a meta-model includes several sublanguages (like UML), and all are instantiated within a model, EMF-Kaizen can produce suggestions for them. For instance, if a model combines a class diagram and state machines, a suggested fragment may include both a class method and a transition trigger. The user could incorporate these two objects into the current model (or into other models) using two interactions, and automated change propagation may also occur if natively implemented in the model. Still, incorporating complex object structures into different parts of a model under construction has limitations. To address this, we plan to add *proxy* objects in the suggested fragments to represent objects in the current model. These proxies would act as glueing points when merging the fragment into the model.

5.4. Threats to validity

Internal validity. RQ1 aimed to assess the effectiveness of the assistant, for which we manually created meta-models, models, 120 requests, and evaluated the results. While three authors did this to mitigate bias or errors, we acknowledge that the best measure for effectiveness needs a user study, which we plan to perform in future work.

External validity. To ensure diversity of scenarios, RQ1 considered 5 meta-models that permit modelling structure and behaviour of systems, including Ecore to work at the meta-model level. Still, stronger evidence of generalisability of results can be gained with more meta-models and models. Similarly, we evaluated four LLMs, but in the future, we may experiment with other LLMs.

Construct validity. When designing experiments with LLMs, there may be risk of *data contamination* (Sainz et al. 2023). This occurs if the training data of the LLMs have seen, even partially, the task to perform, resulting on artificially good results. To mitigate this risk, we manually built all meta-models (except Ecore), models, and NL requests.

6. Conclusions

This paper has presented an assistance approach that is: (i) domain-independent (can be used with different modelling languages); (ii) meta-level independent (can help create models and meta-models); and (iii) notation-independent (works at the abstract syntax level). We have realised the approach as an intelligent assistant for EMF-based (meta-)modelling, and shown its usefulness on several domains and usage scenarios.

Technically, we are working on tracing the recommended model fragments into the model under construction, based on an annotation model. This may serve to understand the parts of the model created by the assistant. Moreover, EMF-Kaizen currently adds the OCL constraints in meta-models to the LLM, but does not validate if the LLM response satisfies the constraints. We aim to tackle this issue, though it might be hard to check as the recommendations are partial models.

In the future, we would like to explore mechanisms for proactive assistance. We plan to support other assistance tasks, like explaining validation errors (e.g., OCL constraint violations) and model repair. In the long term, we may consider extending the set of supported tasks to other MDE activities like model transformation or code generation, and moving our architecture towards an agentic AI framework. We would also like to conduct user studies to better identify strengths and weaknesses of EMF-Kaizen in practice. Finally, an important aspect of intelligent modelling assistance is *benchmarking*. To our knowledge, there is no set of tasks available that can be used to uniformly assess and compare modelling assistants. To fill this gap, researchers can use our experiment replication package for benchmarking. With the same purpose, we encourage the community to make public their experiments and tools.

Acknowledgments

Work funded by MICINN with project PID2024-155231OB-I00.

References

Ahmed, K., Song, J., Chen, B., Wei, O., & Zheng, B. (2025). MCeT: Behavioral model correctness evaluation using large language models. In *MoDELS.* IEEE.

Almonte, L., Guerra, E., Cantador, I., & de Lara, J. (2022). Recommender systems in model-driven engineering. *SoSyM*, *21*(1), 249–280.

Arulmohan, S., Meurs, M., & Mosser, S. (2023). Extracting domain models from textual requirements in the era of large language models. In *MoDELS Companion.* IEEE.

Ben Chaaben, M., Burgueño, L., David, I., & Sahraoui, H. (2026). On the utility of domain modeling assistance with large language models. *ACM ToSEM*, *35*(4), 112:1–112:38.

Bragilovski, M., et al. (2025). Leveraging machines to derive domain models from user stories. *Req. Eng.*, *30*, 241–262.

Brambilla, M., Cabot, J., & Wimmer, M. (2017). *Model-driven software engineering in practice, 2nd ed.* Morgan&Claypool.

Cámara, J., Troya, J., Burgueño, L., & Vallecillo, A. (2023). On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML. *SoSyM*, *22*(3), 781–793.

Chen, B., Wei, O., Zheng, B., & Mussbacher, G. (2025). Accurate and consistent graph model generation from text with large language models. In *MoDELS.* IEEE.

Chen, K., Yang, Y., Chen, B., López, J. A. H., Mussbacher, G., & Varró, D. (2023). Automated domain modeling with large language models: A comparative study. In *MoDELS.* IEEE.

Cibrián, E., Olivert-Iserte, J., et al. (2025). An agent-based approach for the automatic generation of valid SysMLv2 models in industrial contexts. *Comput. Ind.*, *172*, 104350.

de Lara, J., & Guerra, E. (2025). Adaptive modelling languages: Abstract syntax and model migration. *ACM ToSEM*, *34*(3), 66:1–66:54.

Durá, C., López, J. A. H., & Cuadrado, J. S. (2024). ModelMate: A recommender for textual modeling languages based on pre-trained language models. In *MoDELS* (pp. 183–194). ACM.

Fill, H., Fettke, P., & Köpke, J. (2023). Conceptual modeling and large language models: Impressions from first experiments with ChatGPT. *EMISA Journal*, *18*, 3.

Ibrahim, M., & Ahmad, R. (2010). Class diagram extraction from textual requirements using natural language processing (NLP) techniques. In *ICCRD* (pp. 200–204).

Joel, S., Wu, J., & Fard, F. (2025). A survey on LLM-based code generation for low-resource and domain-specific programming languages. *ACM ToSEM*. (Just Accepted)

Lamas, V., García-González, D., Sala, L., & Luaces, M. R. (2026). DSL-Xpert 2.0: enhancing llm-driven code generation for domain-specific languages. *IST*, *190*, 107954.

López, J. A. H., Földiák, M., & Varró, D. (2024). Text2VQL: Teaching a model query language to open-source language models with ChatGPT. In *MoDELS* (pp. 13–24). ACM.

Lu, Y., Li, H., Cong, X., Zhang, Z., Wu, Y., Lin, Y., … Sun, M. (2025). Learning to generate structured output with schema reinforcement learning. In *ACL (1)* (pp. 4905–4918). ACL.

Mussbacher, G., Combemale, B., Kienzle, J., Abrahão, S., Ali, H., Bencomo, N., … Weyssow, M. (2020). Opportunities in intelligent modeling assistance. *SoSyM*, *19*(5), 1045–1053.

Neuberger, J., Ackermann, L., van der Aa, H., & Jablonski, S. (2024). A universal prompting strategy for extracting process model information from natural language text using large language models. In *ER* (Vol. 15238, pp. 38–55). Springer.

OMG. (2016). *MOF: meta-object facility.* Retrieved from https://www.omg.org/spec/MOF

Pérez-Soler, S., Daniel, G., Cabot, J., Guerra, E., et al. (2020). Towards automating the synthesis of chatbots for conversational model query. In *EMMSAD* (Vol. 387, pp. 257–265).

Pérez-Soler, S., González-Jiménez, M., Guerra, E., & de Lara, J. (2019). Towards conversational syntax for domain-specific languages using chatbots. *J. Object Technol.*, *18*(2), 5:1–21.

Pérez-Soler, S., Guerra, E., de Lara, J., & Jurado, F. (2017). The rise of the (modelling) bots: towards assisted modelling via social networks. In *ASE* (pp. 723–728). IEEE.

Prokop, D., Stenchlák, S., Skoda, P., Klímek, J., & Necaský, M. (2024). Enhancing domain modeling with pre-trained large language models: An automated assistant for domain modelers. In *ER* (Vol. 15238, pp. 235–253). Springer.

Saini, R., Mussbacher, G., Guo, J. L. C., & Kienzle, J. (2022). Automated, interactive, and traceable domain modelling empowered by artificial intelligence. *SoSyM*, *21*(3), 1015–1045.

Sainz, O., Campos, J. A., et al. (2023). NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark. In *EMNLP (findings)* (pp. 10776–10787).

Silva, J. (2024). AI assisted domain modeling explainability and traceability. In *MoDELS Companion* (pp. 130–135). ACM.

Silva, J., Ma, Q., Cabot, J., Kelsen, P., & Proper, H. A. (2025). Towards human-in-the-loop LLM-enabled domain modeling. In *ER* (Vol. 16189, pp. 127–145). Springer.

Steinberg, D., Budinsky, F., et al. (2009). *EMF: Eclipse modeling framework 2.0*. Addison-Wesley Professional.

Wang, B., Wang, Z., Wang, X., Cao, Y., Saurous, R. A., & Kim, Y. (2023). Grammar prompting for domain-specific language generation with large language models. In *NeurIPS.*

Wasowski, A., & Berger, T. (2023). *Domain-specific languages - effective modeling, automation, and reuse.* Springer.

Yang, Y., Chen, B., Chen, K., Mussbacher, G., & Varró, D. (2024). Multi-step iterative automated domain modeling with large language models. In *MoDELS Companion.* ACM.

About the authors

Lissette Almonte is assistant professor at the Universidad Autónoma de Madrid (Spain), and member of the miso group. You can contact the author at lissette.almonte@uam.es.

Jefferson Iván Rengifo is member of the miso group. You can contact the author at jefferson.rengifo@uam.es.

Esther Guerra is full professor at the Universidad Autónoma de Madrid (Spain), and co-directs the miso group (http://miso.es). You can contact the author at esther.guerra@uam.es.

Juan de Lara is full professor at the Universidad Autónoma de Madrid (Spain), and co-directs the miso group (http://miso.es). You can contact the author at juan.delara@uam.es.