Generative AI for OCL Constraint Generation: Dataset Collection and LLM Fine-tuning

Fengjunjie Pan, Vahid Zolfaghari, Long Wen, Nenad Petrovic, Jianjie Lin and Alois Knoll

***Abstract*—The Object Constraint Language (OCL) is a formal specification language in model-based systems and software engineering. It defines complex rules and constraints for model-based system design and verification. Constructing an OCL constraint requires expertise not only in OCL syntax but also in meta-model information, which can hinder its application in the practical industrial scenario despite its broad usage. Recently, generative artificial intelligence has demonstrated remarkable performance in code and text generation. This work discusses the generation of OCL constraints from natural language specifications using large language models (LLMs). Given that the automotive and aviation industries are major consumers of model-based engineering, the use of commercial LLMs raises concerns about data privacy. Therefore, we propose to employ open-source and locally deployed LLMs for OCL generation tasks. In this work, we collected a set of meta-models and OCL constraints, which were syntactically validated to ensure the quality of the OCL dataset. Synthetic natural language specifications were generated and used in the dataset for model fine-tuning. Additionally, we designed a retrieval-augmented approach to incorporate meta-model information during LLM fine-tuning and OCL generation. The proposed fine-tuning and OCL generation approach has been experimented with the state-of-the-art open-source LLM, Llama 3 8B. The locally fine-tuned and deployed language model achieved comparable syntactic accuracy and a higher semantic similarity score for OCL generation compared to the cutting-edge commercial models, GPT-4 Turbo and Gemini 1.5 Pro. The usability of the fine-tuned model has been demonstrated for OCL generation in the context of automotive resource allocation.**

I. Introduction

Model-Based Systems Engineering (MBSE) has been widely promoted in complex system design, particularly in the aviation and automotive industries. It leverages formal models for information presentation, providing systematic approaches for requirements management, system analysis, design, validation, and verification. In MBSE, formal models are typically created using modeling tools and languages such as the Eclipse Modeling Framework (EMF) [1], Unified Modeling Language (UML) [2], and Systems Modeling Language (SysML) [3]. Although models can express building blocks and basic rules for system design, complex constraints, especially logical ones, cannot be simply specified. Therefore, the Object Constraint Language (OCL) [4] has been introduced as a complement to models for constraint definition.

OCL is a declarative language that supports models conforming to the MetaObject Facility Specification (MOF) [5], including EMF, UML, and SysML models. The MOF standard describes a meta-modeling methodology where meta-models provide abstract information, including rules, meta-types, and properties, needed for creating a system. Semantic models of concrete systems are referred to as instance models, which are instantiated from meta-models. OCL constraints are typically constructed based on meta-models and applied to model instances. They can be employed for instance model verification [6], information queries [7], and the definition of design space exploration problems [8]. An OCL constraint typically consists of a context and an invariant. The context in OCL specifies to which type of model components the constraint applies, while the invariant defines the constraint rules, typically in first-order logic. Due to the close relationship between OCL and models, writing OCL constraints requires expertise in both system modeling and OCL grammar, which strongly burdens the application of OCL in MBSE for complex systems, despite its wide and reliable usage.

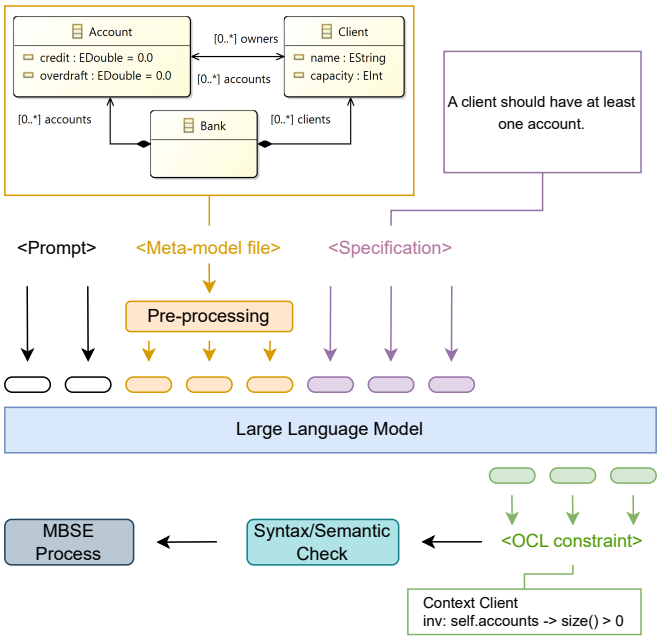

FIG. 1: OCL constraint generation via LLMs

Currently, artificial intelligence (AI) has been frequently discussed for generative tasks. In particular, with the development of transformer-based neural networks [9], large language models (LLMs) have demonstrated their capability to generate text and code based on user input. FIG. 1 illustrates a

F. Pan, V. Zolfaghari, L. Wen, N. Petrovic, J. Lin, A. Knoll are with Robotics, Artificial Intelligence and Real-Time Systems, School of Computation, Information and Technology, Technical University of Munich, Munich, Germany. {panf, zov, wenl, pne, jianjie.lin, knoll}@in.tum.de

potential process of LLM-enabled OCL constraint generation from natural language specification. The input of LLM consists of prompts, meta-model information and natural language specifications. OCL is not a standalone language. It only becomes meaningful when connected to model elements. Since standard text-based language models have limited performance in directly handling model files, a preprocessing step is necessary to parse and extract model information as textual descriptions. Using LLMs, the OCL constraint, based on the input meta-model and reflecting the input specification, can be generated automatically. We propose performing checks against the syntax and semantics of the generated OCL. Afterwards, the generated OCL can be integrated into MBSE processes for system analysis and design.

This study investigate on LLM-based OCL constraint generation. While LLMs are natively powerful due to their training on massive data, they often face compatibility issues with specific downstream tasks, such as OCL generation. Given that the automotive and aviation industries are major consumers of model-based engineering, the use of commercial LLMs raises concerns about data privacy. To address these challenges, we propose to utilize and fine-tune an open-source and locally deployable pre-trained LLM for OCL generation. In this work, the state-of-the-art Llama 3 8B [10] has been selected as proof-of-concept. We gathered a set of meta-models and OCL constraints, ensuring their quality through syntactic validation. To create a complete dataset for LLM fine-tuning, we generated synthetic natural language specifications using the cutting-edge commercial LLM, Generative Pre-trained Transformer-4 (GPT-4) Turbo [11]. During LLM fine-tuning and OCL generation, information in model files was presented as textual description in the domain-specific language, PlantUML [12]. We further designed a retrieval-augmented approach to incorporate meta-model information, preventing context length issues and out-of-memory errors due to limited local hardware. OCL generation is a non-deterministic task. Therefore, for model evaluation, we implemented a syntax checker to validate grammatical correctness and utilized embedding-level Euclidean distance and cosine similarity for semantic evaluation of the generated OCL constraints. Our locally fine-tuned and deployed language model shows comparable syntactic accuracy and achieves a higher semantic similarity score for OCL generation compared to GPT-4 Turbo and Gemini 1.5 Pro [13]. We further showcased the fine-tuned model for generating OCL constraints in a practical automotive scenario.

II. Related Work

Our research topics draws on different directions ranging from traditional natural language processing method for OCL generation to LLM-based approaches. Dataset collection for OCL generation has also been discussed as another important aspect. In addition, we also mention the relation of this paper to our previous works in order to emphasize the practical importance of its outcome.

Before the advent of machine learning models, OCL generation was primarily achieved through model transformations. Bajwa et al. [14] introduced a framework for OCL generation based on natural language specifications. They utilized the Semantic Business Vocabulary and Rules (SBVR) to express natural language text into a structured description. Model-to-model transformation has been used for the generation of OCL from structured text. Salemi et al. [15] followed a similar approach, extending the transformation from SBVR to OCL with additional mapping rules. These traditional methods rely heavily on the implemented transformation rules, making them effective in predefined domains. However, they require thorough implementation of the transformation process.

As generative AI demonstrates its competence in code generation tasks [16], researchers are exploring its potential for OCL generation. Abukhalaf et al. [17] investigated different prompt designs for OCL generation for UML models using OpenAI's Codex (based on GPT-3). They found that including model information enhances the validity score of generated OCL constraints. Similarly, Cámara et al. [18] presented an approach for LLM-based OCL constraint generation starting from textual descriptions and UML models using ChatGPT. The authors highlighted the potential of this method in practical applications across various domains, including airline systems, education, banking, and automotive industries. In our recent work [19], we utilized ChatGPT 4 for OCL rule extraction as part of an automated workflow for designing future-proof vehicular systems. These approaches using commercial LLMs have demonstrated the feasibility of OCL generation. However, the demonstrations involved relatively small numbers and sizes of components. Additionally, the use of commercial LLMs raises data privacy concerns, particularly in domains such as automotive and aviation, which are key players in MBSE. Therefore, we aim to discuss the fine-tuning of open-source and locally deployable LLMs for OCL generation tasks.

For fine-tuning LLMs for OCL generation, a dataset containing meta-models, OCL constraints, and natural language specifications is essential. Existing OCL datasets, such as those from Cabot et al. [20] and Noten et al. [21], include ecore models and OCL constraints. However, these datasets lack natural language specifications. The UML dataset collected by Abukhalaf et al. [17] comprises 15 UML models and 114 pairs of OCL constraints and specifications. For effective training, a significantly larger and more complex dataset is required.

In this work, we will collect a dataset containing diverse meta-models, OCL constraints, and natural language specifications for fine-tuning and experimenting with large language models (LLMs). We will discuss how we utilize a locally deployed open-source LLM to generate OCL constraints.

III. Dataset Collection

Current LLMs have presented decent capability in general tasks such as text summarization and chat generation. However, their performance, particularly in open-source LLMs, for specific tasks including OCL constraint generation, requires further improvement. Fine-tuning is a supervised learning process applied to a pre-trained LLM to enhance its performance

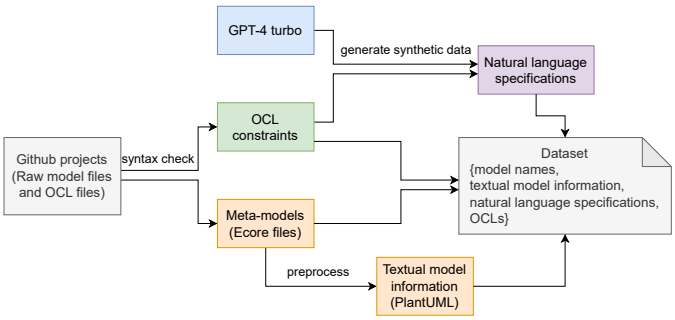

FIG. 2: Collection of meta-models and OCL constraints, generation of synthetic natural language specification

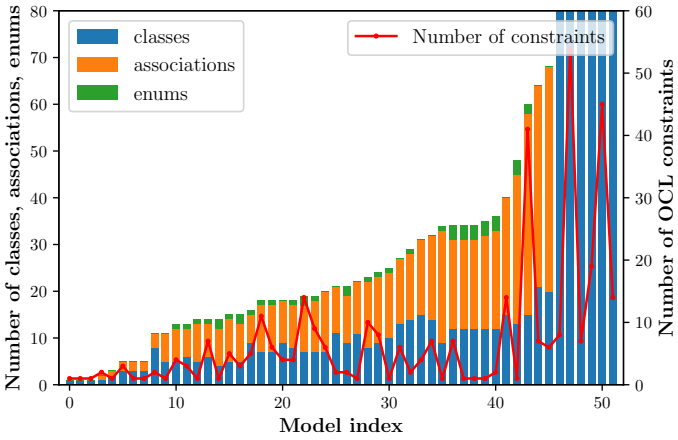

FIG. 3: Distribution of model elements and constraints in the Dataset

in specific domains. In any machine learning procedure, the dataset is the most fundamental element.

Several open-source modeling datasets and projects are available. However, they often lack natural language specifications or OCL constraints. In this work, we aim to create a dataset containing models, natural language specifications, and OCL constraints. The procedure for data collection and processing is presented in FIG. 2. We utilized the searching approach described in [21] to collect open-source EMF projects containing both meta-models (.ecore) and OCL files (.ocl) on GitHub via the GitHub-REST-API. EMF is a tool provided by Eclipse that primarily works with Ecore meta-models but can also support UML and SysML models. The Eclipse tools for UML and SysML are both based on EMF, allowing for the possibility of extending this dataset with UML and SysML models using the same data collection and processing approach. We consider that the corresponding Ecore file of an OCL file should either have the same file name or be specified in the *import* declaration. Currently, we mainly focus on OCL constraints presented as invariants because the gathered GitHub projects do not have enough samples of other constraint types, such as preconditions and postconditions. We have also removed all invariant names to maintain data uniformity. The invariant names are used as labels and do not affect the validity of OCL constraints. A valid OCL invariant constraint in our dataset consists of a context and a constraint expression body.

After the initial search and filtering, we collected 276 unique meta-models and 1054 OCL constraints. As mentioned by [22], the quality of data is more important than quantity for the LLM fine-tuning. Thus, we further utilized the Eclipse EMF[1] and OCL[2] plugins to perform syntactic check on the collected models and constraints. We consider each model and constraint that can be correctly loaded in Eclipse as a valid sample for the dataset. This resulted in 52 meta-models and 369 OCL constraints of different sizes and contexts, which build up our dataset FIG. 3 presents the complexity of the models in terms of the number of elements, including classes, associations, enums, and OCL constraints that apply to these models. The most of collected models have between 1 and 68 elements, providing a total of 222 constraints. Six models are more complex, with 314 to 885 elements, and they have a total of 147 constraints.

Synthetic data for language models have been discussed and proven to be a successful approach for fine-tuning LLMs [23]. We recognize that state-of-the-art LLMs can generate reasonable natural language specifications based on OCL constraints, as OCL, being a declarative language, is easier to understand than to write. Therefore, we utilized GPT-4 Turbo to generate natural language specifications based on OCL constraints through a dedicated prompt involving one-shot learning (Listing 1).

LISTING 1: Prompt for natural language specification generation via one-shot learning

```
You are given a json file contains OCL (Object
     Constraint Language) constraint.
Your task is to generate natural language
    instruction that corresponds to this
    constraint.
You should use information from both model and
     OCL during generation.
You should return a JSON file with OCL and
    specification.
The resulting JSON should look like:
{
  OCL: // description of OCL constraint how it
       was in the initial JSON,
  specification: // natural language
      instruction (should be generated by you
      and correspond to OCL)
}
Example:
Input:
{
  OCL: context Human inv: age > 0
}
Output:
{
  OCL: context Human inv: age > 0
  specification: Age of a human should be a
      positive value.
```

[1]https://eclipse.dev/modeling/emf/

[2]https://projects.eclipse.org/projects/modeling.mdt.ocl

TABLE I: Summary of the character length and approximate token length of dataset entries

Content	Length of Characters			Length of Tokens (approx.)		
	min.	max.	avg.	min.	max.	avg.
puml	26	44100	10279	7	11025	2570
ocl	23	711	99	6	178	25
specification	29	550	99	7	125	25

}

The generated OCL dataset contains a total of 52 meta-models and 369 natural language specifications along with their corresponding OCL constraints. As discussed in FIG. 1, preprocessing of model files is required for leveraging the model information as textual information to LLMs. We convert the Ecore models into PlantUML [12] using the plantumlgen tool[3]. PlantUML is a domain-specific language originally designed for generating graphical UML diagrams. It specifies classes, their attributes, and relationships in the meta-model. In the dataset, we provide both the PlantUML texts and the original Ecore files, allowing for the generation of textual model descriptions in other formats through customized preprocessing steps. Introducing such preprocessing steps for model information makes the training and inference procedures independent of specific model formats. The generated OCL dataset is made publicly available on Huggingface Huggingface[4]. Listing 2 presents one data entry sample in our dataset, where *model_name* represents the name of the meta-model, with which the Ecore file can be located. *puml* contains the model information in PlantUML format, and *ocl* and *specification* present the OCL constraint and its natural language meaning, respectively.

LISTING 2: Sample of dataset entry

```
{
  "model_name": "catalogue.ecore",
  "puml": "class Catalogue {nom : EString}
  association Catalogue \"algorithmes 0..*\"
       *--> Algorithme",
  "ocl": "context Catalogue inv: nom <> null
      ",
  "specification": "The name of any
      catalogue must not be null."
}
```

Table I presents a summary of the character length and approximate token length of dataset samples consisting of PlantUML model descriptions, OCL constraints and natural language specifications. The character length of PlantUML descriptions ranges from 26 to 44100 with an average length of 10279. The character length for OCLs ranges from 23 to 711 with an average length of 99. The character length for natural language specifications ranges from 29 to 550 with an average length of 99. Tokens are the fundamental units of language model inputs and outputs. A token can be part of a word and is assigned a unique identifier, typically a numerical value, by tokenizers. As different LLMs utilize different tokenizers, the length of generated tokens can vary. For the approximation of token length, we roughly consider that one token represents four characters. In our dataset, the token length of OCL and specification samples both average around 25, with a minimal length of less than 10 and a maximal length of less than 200. For the model information in PlantUML, the token size ranges from 7 to 11025 with an average value of 2570.

[3]https://gist.github.com/aranega/eca9c8fbbd87b2f9c70317da53676ac6

[4]https://huggingface.co/datasets/fpan/text-to-ocl-from-ecore

IV. LLM Fine-Tuning and OCL Generation

This work discusses the OCL constraints generation based on meta-model information and natural language specifications. We aim to fine-tune an open-source, locally deployed LLM, Llama 3 8B, for OCL generation to ensure data privacy and maintain control over sensitive information This approach is particularly beneficial for industries like aviation and automotive, where confidentiality is crucial for system development. The fine-tuning described in this paper is based on the dataset presented in Section III. We utilize the Low-Rank Adaptation (LoRA) [24] method for fine-tuning the target LLM in our experiments. LoRA enables model fine-tuning on resources with limited computational power by reducing the number of parameters that need to be adjusted. In this section, we will discuss our approach to handling long contexts and introduce the prompts used during model fine-tuning and OCL generation.

A. Bi-encoder-based Information Retrival for Long Context

LLMs typically have limitation on the input and output context length. Although the context length is increasing with the development of the latest generative AI solutions, this problem persists, especially for locally deployed LLMs with limited computational power and memory. In MBSE, complex systems have extensive meta-models, which are challenging to feed into an LLM for fine-tuning and inference. As presented in Section III, the maximum length of an input model reaches around 11k tokens, exceeding the 8k context length supported by Llama 3. To address this, we propose adopting a Retrieval-Augmented Generation (RAG) concept for retrieving model information relevant to LLM fine-tuning and OCL generation. RAG has been designed for knowledge-intensive NLP tasks [25]. It involves filtering external knowledge from a large data source and using it as input for text generation, aiming to improve the quality of the generated response. In the context of OCL generation, it is essential to consider that the meta-model itself provides crucial information, such as class definitions, attributes, and their associations, which are relevant for OCL constraint generation.

The proposed method for meta-model information retrieval has been presented in FIG. 4. The process begins with the preprocessing of meta-models into textual representations. We utilize PlantUML format as model textual representation (Section III). The generated PlantUML texts of each meta-model are then split into smaller chunks, with each chunk

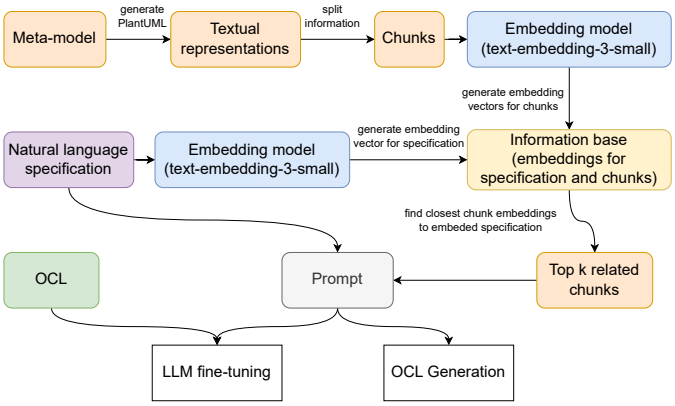

FIG. 4: Retrieval-augmented fine-tuning and generation

representing either one class with its properties, one association, or one enumeration. These chunks are compared with specifications to filter out those with related meanings for further fine-tuning and generation. This comparison is performed using a bi-encoder architecture. We employ embedding models to generate embeddings for both chunks and specifications. Embeddings are vectors that represent the meaning of words and sentences. For this purpose, we use the text-embedding-3-small model [11] from OpenAI as the bi-encoder. Each generated embedding vector has a length of 1536. During the information retrieval process, each specification embedding is compared against the chunk embeddings of the corresponding meta-model by calculating their cosine similarity values. The top k related chunks are then selected as the filtered meta-model information. Considering the hardware limitation in our experiment, we set k to 40, meaning the top 40 classes, associations, and enumerations will be considered related meta-model input. The retrieved meta-model information, together with the natural language specification and OCL constraints, forms the final prompt for LLM fine-tuning. During the OCL generation, only the model information and natrual language specification are provided as LLM inputs.

B. Prompt design for fine-tuning and OCL generation

Listing 3 presents the prompt that we craft for fine-tuning (with OCL constraints inserted at the end of prompt) and OCL generation (without OCL constraints in the last line). The design of the prompts for fine-tuning and generation follows a similar structure to the prompt used for dataset generation (Listing 1). The prompt starts with an instruction expressing the goal of generation. This is followed by the retrieved meta-model information, natural language specification, and OCL constraints (included only for model fine-tuning).

LISTING 3: Prompt example for LLM fine-tuning and OCL generation

```
You are given an meta-model with information
    about classes, associations and their
    attributes.
You are also given a natural language
    specification.
Your task is to generate an OCL (Object
    Constraint Language) constraint for this
     specification and based on the meta-
    model.
The meta-model information is:
{{model}}
The natural language specification is:
{{specification}}
The generated OCL constraint is:
{{ocl}}
```

V. EXPERIMENT

In the experiment, we fine-tuned the pre-trained Llama 3 8B with the collected dataset in Section III. The dataset was split using an 80-20 rule, resulting in a training set containing 295 samples and a test set containing 74 samples. We utilized the Transformers API[5] to perform Parameter-Efficient Fine-Tuning using the LoRA method on a single Nvidia A5000 GPU with 24GB of VRAM. The LoRA rank, which denotes the rank of the low-rank matrix used for adapting the original weight matrix, was set to 16. This configuration resulted in around 6 million trainable parameters out of a total of around 8 billion parameters. The LoRA alpha, a scaling weight for merging LoRA weights with the original model weight, was set to 32. The LoRA dropout rate, applied during the fine-tuning process, was set to 0.1. Due to the VRAM limitations of the GPU, we set the context length to 3000. The model was trained for 3 epochs.

A. LLM evaluation

The fine-tuned model was compared with state-of-the-art commercial LLMs, GPT-4 Turbo and Gemini 1.5 Pro, by assessing the syntactic correctness and semantic similarity of the generated OCL constraints. Despite the small parameter size of Llama 3 8B, we successfully enhanced it for OCL generation through fine-tuning. This enhancement brought the performance of the fine-tuned model, referred to as Llama 3-OCL in this paper, to a similar level as the commercial LLMs, GPT-4 Turbo and Gemini 1.5 Pro, in our test set. The evaluation metrics are presented in Table II and will be analyzed in the following part.

TABLE II: Model comparison result

LLM	Syntactic Correctness	Cosine Similarity	Euclidean Distance
GPT-4 Turbo	76%	0.86	0.47
Gemini 1.5 Pro	70%	0.86	0.45
Llama 3-8B	39%	0.80	0.58
Llama 3-OCL	73%	0.91	0.32

[5]https://huggingface.co/docs/transformers/en/index

1) Syntactic correctness: The syntactic correctness is a key evaluation metric for OCL generation, as it directly determines whether the generated OCL constraints can be read by a machine and used in further automated MBSE processes, such as model verification and design space exploration. Validation of syntactic correctness should be based on the meta-model. A syntactically correct OCL must not only adhere to the OCL grammar structure but also use accurate variable names from the meta-model. We employed the same strategy for syntactic checking of generated OCL constraints as we did for filtering valid data in the dataset (Section III). The generated OCL constraints were loaded together with their corresponding meta-model using Eclipse EMF and OCL tools. Any syntactically incorrect OCL constraint would result in a failure during this process.

As shown in Table II, GPT-4 Turbo achieved the highest syntactic correctness rate, with 76% of the generated content being directly loadable and usable in Eclipse. Gemini 1.5 Pro achieved a syntax accuracy of 70%. In contrast to these two commercial models, the pre-trained Llama 3-8B showed unsatisfactory performance, with only 39% syntactic accuracy. However, after fine-tuning, we successfully increased the syntactic correctness of Llama 3-8B to 73%, which outperforms Gemini 1.5 Pro and is comparable to GPT-4 Turbo.

2) Semantic similarity: OCL generation can be recognized as a variant of the machine translation task. The goal is to translate the constraints during system design into a machine-processable format, which in our case is OCL, for automated analysis. Semantic correctness plays an important role in any translation task. Therefore, we use the semantic similarity between generated and ground truth OCL constraints as another indicator of the performance of the LLM for OCL generation.

We calculated the cosine similarity and Euclidean distance of embeddings of generated and ground truth OCL. Cosine similarity is measured by the cosine of the angle between embedding vectors, determining whether these two embeddings point in roughly the same direction [26]. A higher cosine value indicates greater semantic closeness of the texts. Euclidean distance, a straightforward similarity metric, reflects the distances between the endpoints of vectors. A lower Euclidean distance indicates higher similarity.

According to Table II, the OCL constraints generated by the fine-tuned model have the highest similarity to the ground truth, followed by GPT-4 Turbo and Gemini 1.5 Pro. An obvious semantic improvement can be observed during the fine-tuning of Llama 3-8B.

B. A practical application in automotive system design

We further demonstrated the effectiveness of the fine-tuned model by applying it to a practical automotive resource allocation scenario described in our previous work [8]. We used the Llama 3-OCL for generating OCL constraints based on a meta-model of a centralized vehicle architecture. The natural language specifications were derived from the non-functional requirements of the system design. In this part, we compare the generated OCL constraints with those manually crafted in [8]. Additionally, we created a simple instance model to illustrate how system validation can be performed using OCL constraints. The demonstration was conducted using Eclipse, with EMF and OCL plugins.

The meta-model of the target vehicle system is presented in FIG. 5. It describes an abstract automotive resource allocation scenario on a centralized vehicle platform. This platform includes resources such as CPU, RAM, and other related devices necessary for executing vehicle applications. Isolated virtual machines (VMs) should be created to enable the coexistence of applications with different safety levels on the same physical platform.

Designing a concrete automotive system involves various requirements and design specifications, such as resource availability, application coexistence/conflict, safety aspects, and platform-related configurations. We selected five representative specifications and generated the corresponding OCL constraints using the fine-tuned Llama 3-OCL, following the proposed workflow presented in FIG. 4. The generated OCL constraints were then compared with those written by human experts. The natural language specifications, generated, and human-written OCL constraints are presented in Table III. All OCL constraints were found to be both syntactically and semantically correct.

Notably, the generated and human-written OCLs exhibit different coding styles. The generated constraints omitted the names for invariants, as these were removed from our dataset. This omission is purely a labeling issue and does not affect the OCL constraints themselves. Additionally, the generated constraints provide detailed information in the body of the OCL expressions, while the human-written ones are more compact. For example, in the OCL constraints from the third specification, both OCLs use the same context and operators. However, the generated OCL explicitly presents all hidden references to elements (with *self, x* representing *App* instances), whereas the human-written one omits such details. Since OCL constraints are non-deterministic for a specification, the generated OCLs may have different contexts and operators compared to the human-crafted ones. Nonetheless, they all strictly adhere to the class, attribute, and association names from the meta-model.

The OCL constraints can be used for instance model validation and design space exploration. In this demonstration, we created a customized instance model and introduced a conflict with the fifth specification (FIG. 6). The concrete automotive system is instantiated with two VMs, two applications, and one CPU with four cores. The CPU has the turbo boost feature activated. *Vm0* has a safety level of 1. However, *Core0* is allocated to *Vm0*, which violates specification 5. We used the same labels in the generated OCL as in the human-written OCL. The generated constraints were saved in a separate text-based .ocl file and used for validating the instance model. The conflict in specification 5 was successfully detected and reported in a separate window in Eclipse.

TABLE III: OCL constraints (generated and human written) for automotive resource allocation-related specifications.

Specification	Generated OCL constraints by Llama 3-OCL	Human written OCL constraints from [8]
1. VM should provide sufficient memory for applications	`context Vm inv: self.app -> forAll(a:App| a.ramSize <= self.ramSize)`	`context Vm inv VmRamSize: app->collect(ramSize)->sum() <= ramSize`
2. Application should be mapped to VMs with the respective safety levels.	`context App inv: self.vm->forAll(vm| vm.safetyLevel = self.safetyLevel)`	`context App inv AppAndVmSafety: safetyLevel = vm.safetyLevel`
3. Co-location applications should be mapped on the same VMs.	`context App inv: self.coLocation-> forAll(x| x.vm = self.vm)`	`context App inv CoLocation: coLocation->forAll(vm = self.vm)`
4. Separate-location applications should be mapped on different VMs.	`context App inv: self.sepLocation->forAll(p| p.vm->excludes(self.vm))`	`context App inv SeparateLocation: sepLocation->forAll(vm <> self.vm)`
5. Cores assigned to safety-critical VMs (safety >0) should have disabled the turbo boost on its CPU.	`context Cpu inv: self.core->forAll(c:Core | c.vm-> exists(v:Vm | v.safetyLevel > 0) implies not self.turboBoost)`	`context App inv VmSafetyConfig: safetyLevel > 0 implies core.cpu.turboBoost = false`

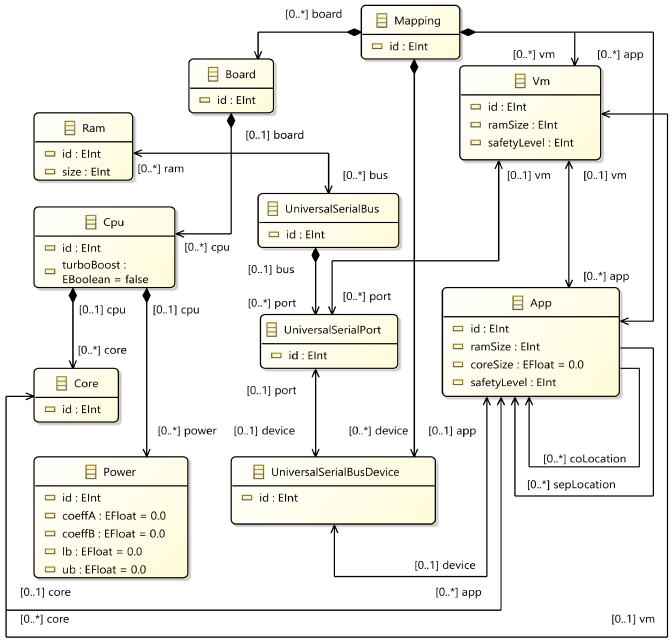

FIG. 5: Meta-model for an automotive resource allocation scenario on a centralize vehicle platform [8]

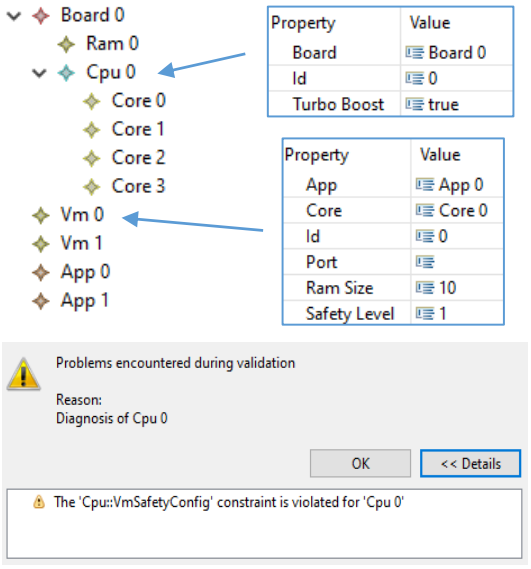

FIG. 6: Validation of instance model against generated OCL constraints

VI. Conclusion

In this paper, we present a novel data collection, LLM fine-tuning and OCL generation approach on an open-source and locally deployable LLM. We successfully fine-tuned the Llama 3 to achieve performance comparable to the state-of-the-art commercial GPT-4 Turbo for OCL generation. The usability of the fine-tuned model has been demonstrated for generating OCL constraints related to automotive resource allocation.

The generated dataset consists of ecore meta-models, OCL constraints as invariants, and natural language specifications. All meta-models and OCLs in the dataset were collected from open-source projects on GitHub. Additionally, we designed a grammar checker using Eclipse EMF and OCL plugins to ensure the syntactic correctness of the collected data. Natural language specifications for OCL constraints represent synthetically generated data resulting from a state-of-the-art commercial LLM (GPT-4 Turbo in our case). During LLM fine-tuning and OCL generation, we converted the meta-model files into textual PlantUML format. We also introduced a retrieval-augmented approach to reduce the input context length while preserving necessary information for OCL generation. This approach enables LLM fine-tuning and OCL generation in-

volving large system meta-models on hardware with limited computational resources. The fine-tuned LLM was compared with the state-of-the-art GPT-4 Turbo and Gemini 1.5 Pro, which rank highly on many LLM leaderboards, for OCL generation. The evaluation considered both the syntax and semantics of the generated OCL constraints. Our findings indicate that the fine-tuned Llama 3-OCL outperforms the pre-trained Llama 3-8B model in both syntax and semantics. It achieves a syntactic correctness rate that outperforms Gemini 1.5 Pro and is comparable to GPT-4 Turbo, while having the closest semantic similarity to the ground truth. We further utilized the fine-tuned model for OCL generation in the context of automotive design. The generated OCLs have been successfully used to detect conflicts in a customized instance model against the design constraints.

For future work, we plan to extend the dataset with a higher number of constraints and enrich the types of OCL expressions, including pre- and post-conditions. Additionally, we will further investigate the semantic evaluation. Currently, we use similarity as a performance indicator, but we plan to explore the automated generation of instance models to create positive and negative test cases for OCL constraint compliance. Furthermore, as models are often represented as block diagrams in industry-related scenarios, another aspect to investigate is OCL rule generation from multi-modal inputs (visual and language).

ACKNOWLEDGMENTS

This work was conducted as part of the CeCaS-Mannheim project funded by Bundesministerium für Bildung und Forschung (BMBF).

REFERENCES

[1] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: eclipse modeling framework*. Pearson Education, 2008.

[2] D. Pilone and N. Pitman, *UML 2.0 in a Nutshell*. " O'Reilly Media, Inc.", 2005.

[3] S. Friedenthal, A. Moore, and R. Steiner, *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[4] OMG, *Object Constraint Language Version 2.4*, Feb. 2014.

[5] OMG, *OMG Meta Object Facility (MOF) Core Specification Version 2.5.1*, Oct. 2019.

[6] B. Demuth and C. Wilke, "Model and object verification by using dresden ocl," in *Proceedings of the Russian-German Workshop Innovation Information Technologies: Theory and Practice, Ufa, Russia*, 2009, pp. 687–690.

[7] D. H. Akehurst and B. Bordbar, "On querying uml data models with ocl," in *International Conference on the Unified Modeling Language*. Springer, 2001, pp. 91–103.

[8] F. Pan, J. Lin, M. Rickert, and A. Knoll, "Automated design space exploration for resource allocation in software-defined vehicles," in *2023 IEEE Intelligent Vehicles Symposium (IV)*, 2023, pp. 1–8.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[10] Meta, "Meta-llama-3-8b," https://huggingface.co/meta-llama/Meta-Llama-3-8B, accessed: 2024-06-15.

[11] OpenAI, "Models," https://platform.openai.com/docs/models, accessed: 2024-06-15.

[12] PlantUML, "Plantuml at a glance," https://plantuml.com/, accessed: 2024-06-21.

[13] Google, "Gemini pro," https://deepmind.google/technologies/gemini/pro/, accessed: 2024-06-26.

[14] I. S. Bajwa, B. Bordbar, and M. G. Lee, "Ocl constraints generation from natural language specification," in *2010 14th IEEE International Enterprise Distributed Object Computing Conference*, 2010, pp. 204–213.

[15] S. Salemi, A. Selamat, and M. Penhaker, "A model transformation framework to increase ocl usability," *Journal of King Saud University - Computer and Information Sciences*, vol. 28, no. 1, pp. 13–26, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157815000853

[16] A. Svyatkovskiy, S. K. Deng, S. Fu, and N. Sundaresan, "Intellicode compose: Code generation using transformer," in *Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2020, pp. 1433–1443.

[17] S. Abukhalaf, M. Hamdaqa, and F. Khomh, "On codex prompt engineering for ocl generation: An empirical study," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, 2023, pp. 148–157.

[18] J. Cámara, J. Troya, L. Burgueño, and A. Vallecillo, "On the assessment of generative ai in modeling tasks: an experience report with chatgpt and uml," *Softw Syst Model 22*, vol. 22, pp. 781–793, 2023.

[19] N. Petrovic, F. Pan, K. Lebioda, V. Zolfaghari, S. Kirchner, N. Purschke, M. A. Khan, V. Vorobev, and A. Knoll, "Synergy of large language model and model driven engineering for automated development of centralized vehicular systems," Technical University of Munich, Tech. Rep., 2024. [Online]. Available: https://mediatum.ub.tum.de/doc/1738811/1738811.pdf

[20] J. Cabot, D. Delgado, and L. Burgueño, "Combining ocl and natural language: a call for a community effort," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 908–912. [Online]. Available: https://doi.org/10.1145/3550356.3561542

[21] J. Noten, J. G. Mengerink, and A. Serebrenik, "A data set of ocl expressions on github," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 531–534.

[22] C. Zhou, P. Liu, P. Xu, S. Iyer, J. Sun, Y. Mao, X. Ma, A. Efrat, P. Yu, L. Yu, S. Zhang, G. Ghosh, M. Lewis, L. Zettlemoyer, and O. Levy, "Lima: Less is more for alignment," 2023.

[23] R. Liu, J. Wei, F. Liu, C. Si, Y. Zhang, J. Rao, S. Zheng, D. Peng, D. Yang, D. Zhou, and A. M. Dai, "Best practices and lessons learned on synthetic data for language models," 2024.

[24] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021.

[25] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[26] J. Han, M. Kamber, and J. Pei, "2 - getting to know your data," in *Data Mining (Third Edition)*, third edition ed., ser. The Morgan Kaufmann Series in Data Management Systems, J. Han, M. Kamber, and J. Pei, Eds. Boston: Morgan Kaufmann, 2012, pp. 39–82.