*23rd Annual Conference on Systems Engineering Research*
*Intelligent Digital Twin-enabled Systems Engineering for 21st Century Sociotechnical Systems*

***Editors: Dr. Mathew Amissah[1], Dr. Abbas Zaidi[1], Dr. Ali K. Raz[1], Dr. John Shortle[1], Dr. Azad M. Madni[2], Dr. Dinesh Verma[3],***
*[1]George Mason University,[2]University of Southern California, [3]Stevens Institute of Technology,*

***Arlington, VA, USA April 6-9, 2026***

# Large Language Models for Systems and Ontological Modeling: Fine-tuning and Evaluation

***Ramón María García Alarcia[a,b], Ahmed Alhamadah[b], Maged Elaasar[c], Alessandro Golkar[a], Alejandro Salado[b]***
**[a]Department of Aerospace and Geodesy, Technical University of Munich, ramon.garcia-alarcia@tum.de**
**[b]Department of Systems and Industrial Engineering, The University of Arizona, alejandrosalado@arizona.edu**
**[c]Jet Propulsion Laboratory, National Aeronautics and Space Administration, maged.e.elaasar@jpl.nasa.gov**

***Abstract***

***An increased number of specialized Large Language Models (LLMs) are being released recently, supporting a variety of domain-specific tasks at a fraction of the inference cost of general-purpose LLMs. The Systems Engineering community, with a fragmented landscape of modeling languages, could benefit from specialized LLMs for system modeling tasks, bringing down barriers that prevent adopting Model Based Systems Engineering (MBSE). However, the community does not yet have a well-established corpus of specialized LLMs, nor fine-tuning methodologies tailored to the specificities of its languages. In this paper, we contribute a methodology to fine-tune LLMs for small, underrepresented modeling languages. Using the generic Ontological Modeling Language (OML) as a reference, we explore the balance between the fine-tuning an LLM and using a general-purpose one.***

***Keywords: ontology modeling, systems modeling, LLM, CoPilot, fine-tuning, OML***

## 1. Introduction

The current wave of development of Large Language Models (LLMs) has brought not only more performant general-purpose models, but increasingly, also smaller, domain-focused models [1]. These models can deliver more accurate responses in select, domain-specific tasks at a portion of the bigger models' cost, while being able to run on lower-resource computing environments. They have been successfully applied to coding for a variety of programming languages [2] [3] [4], chip design [5], and telecommunications engineering [6], among others.

Large Language Models (LLMs) can significantly lower the barrier for systems engineers to engage in model-based systems engineering or digital engineering by enabling them to create models simply by describing system concepts in natural language, rather than mastering a specific modeling language. This capability is particularly crucial given the constant evolution and diversity of modeling languages, as well as the reality that many practicing systems engineers have received little or no formal training in these specialized languages. By allowing engineers to communicate their ideas in everyday language, LLMs make modeling more accessible, freeing engineers from the cognitive demands of learning complex syntax and allowing them to focus on the substantive system aspects. Moreover, this approach has the potential

to enhance interoperability across different modeling languages, reducing the need for intensive retraining and facilitating collaboration within multidisciplinary teams.

However, the SE community lacks specialized LLMs (besides a couple of exceptions for SysML [7] and OCL [8] which have not yet shown a clear improvement of modeling accuracy), and general-purpose LLMs do not show a complete lexical and grammatical command of modeling languages. This is the case especially for modeling languages that are more niche and therefore less represented on the Internet, such as the Ontological Modeling Language (OML) [9].

While some have addressed this gap by trying to enhance general-purpose LLMs via Prompt Engineering (PE) or Retrieval Augmented Generation (RAG) pipelines [10], they often fall short of the required accuracy improvement and in all cases require the usage of a portion of the LLM's context window, which reduces the available space for the remainder of the LLM conversation. In addition, general-purpose LLMs are typically closed source, and available only via costly external API calls, implying a transfer of data as well to external servers. Smaller, specialized LLMs can run on-premises on more modest computing infrastructure, in some cases even on personal computing devices.

Despite their advantages, there are also challenges associated with the creation of domain-specific LLMs. Among others, the lack of big repositories of modeling code, the efforts linked to the extraction and annotation of data for the training process, and the costs associated with the training itself [11]. The accuracy of the specialized LLM is proportional to the amount of data and quality of the training data. As a result, building specialized LLMs for modeling languages with a small user community is especially challenging.

In this paper, we explore the creation of a domain-specific LLM for the generation of Ontological Modeling Language (OML) code. OML is an interesting example of a language due to its relatively small community, resulting in few data available, but also its higher level of abstraction, appealing to a broader community by allowing the modeling of systems and of system modeling languages alike. In the paper, we explain our methodology for the creation of an OML training dataset and process, the selection of a fine-tuning strategy and parameters, and the resulting model's accuracy.

## 2. Background

The application of Large Language Models (LLMs) to systems modeling is a nascent area of research and development within Systems Engineering (SE) [12]. It is encouraged by the success of LLMs at coding in mainstream programming languages [13], and by the promises of lower access barriers to SE modeling languages and a refocus of the engineer's efforts to the content rather than the continent of the modeling task.

The off-the-shelf usage of general-purpose LLMs is limited by their lack of deep knowledge of low-resource or domain-specific languages [14]. This is the case of SE modeling languages, which are underrepresented on the Internet. To overcome this limitation, the most extended approach in the SE community is to leverage the LLMs capabilities of in-context learning [15] by employing Prompt Engineering (PE) or Retrieval Augmented Generation (RAG) [16]. By using PE, practitioners encode fundamental information about the modeling language, such as its syntax, and a series of examples of task instructions and expected responses, in the initial prompt sent to the LLM [17] [18]. When using RAG, the LLM prompt is matched against a database that contains information such as the modeling language's reference or model examples, and the most relevant examples are pulled and added as context to the initial prompt [19] [20] [21].

However, the LLM responses with PE or RAG still depict mixed results (e.g., more than 91% grammar accuracy but as low as 44% semantic consistency [18], syntactic validities ranging from 20 to 55% [19], or functional correctness from 42% to 95% [20]). We argue that this might be due to the nature of the in-context learning techniques: they do not modify the LLM parameters and thus do not fully achieve domain-specific language understanding. The effect of in-context learning might just be a patch to LLM's lack of understanding. It might also be due to the sensitivity of the LLM to the way that the prompt is formulated due to the model's stochasticity, resulting in some prompts being suboptimal yet others appearing to work. In addition, employing PE and RAG leads to a higher token usage -increasing both cost and latency- and is limited by the size of the context window of the LLM -which is tight in small LLMs-.

Fine-tuning general-purpose LLMs to create specialized models for modeling might resolve some of the

drawbacks: fine-tuning acts directly on the model parameters and thus affects its fundamental understanding, and LLM usage thereafter might not require the inclusion of additional context tokens. In the SE community, only two papers tackle the creation of specialized LLMs for modeling, one for the Systems Modeling Language (SysML) [7], and one for the Object Constraint Language (OCL) [8], but these papers have fundamental limitations on the fine-tuning strategy and a limited evaluation and discussion of the resulting model.

The fine-tuned model for SysML v2 [7] still underperforms the generic-purpose model it is compared to (OpenAI's GPT-4). We argue this is a flaw of both the fine-tuning approach and the selected base model. First, the approach is not granular enough, as full SysML v2 examples are used for training. These examples are long and contain a variety of syntactical elements of disparate complexity. This might prevent progressive learning of the different language elements and features of SysML v2 that might be attained when separating language elements into different packs of increasing complexity. Second, the 200-million parameter Code-T5 model employed might not be big enough. Current state-of-the-art models are typically an order of magnitude bigger. The fine-tuned model for OCL [8] is based on the larger, 8-billion-parameter Llama 3. However, the results are still ambivalent, reaching higher syntactic accuracy than the Gemini 1.5 Pro model, but lower than the GPT-4 Turbo. The dataset is still composed of full examples of OCL code, and a limited number of them (less than 300 samples), which limits the learning process. On both papers, the evaluation of the results is very limited, as the fine-tuned model's accuracy is compared only with one or two closed-source LLMs whose size or characteristics are not well known due to their proprietary nature.

With our paper, we aim to fill the gaps in the early R&D efforts on fine-tuning LLMs for systems modeling by proposing a novel approach with a granular, progressive fine-tuning on state-of-the-art base LLM; and a comprehensive evaluation strategy with multiple LLMs of diverse sizes and providers with lexical and semantic metrics.

## 3. Methodology

We structure our theoretical contributions around two research questions (ref. Table 1), for which we formulate hypotheses, and we discuss, in this section, the associated methodological elements.

| RQ ID | Research Question | Hypothesis | Methodological elements | Results discussion |
|---|---|---|---|---|
| **#1** | Can the gap between smaller LLMs and bigger LLMs be bridged by fine-tuning? | By fine-tuning, a smaller LLM can achieve or exceed the accuracy of a bigger LLM in OML code generation tasks. | Gathering of OML projects (3.1)<br>Extraction of OML snippets (3.2)<br>Construction of an OML training and testing dataset (3.3)<br>Fine-tuning procedure (3.4)<br>Selection of benchmarking models (3.5)<br>Selection of evaluation metrics (3.6) | Results of the fine-tuning process (4.1)<br>Comparisons of benchmarking models and OMLlama (4.2)<br>Comparison of OML snippet generation (4.3) |
| **#2** | How does the captured OML modeling accuracy vary across levels of OML construct's complexity? | There is an inversely proportional relation between the complexity of the requested OML construct and the LLM's response accuracy. | Construction of an OML training dataset (3.3)<br>Selection of benchmarking models (3.5)<br>Selection of evaluation metrics (3.6) | Similarity evolution with increasing complexity (4.4) |

Table 1: Research questions and associated methodological elements and results discussion.

### 3.1. Gathering of OML projects

In the data gathering phase, we collected the OML repositories available in GitHub (coming mostly from the openCAESAR initiative [22]) and the OML projects of the University of Arizona, and created a unified combined project containing these source OML files. A total of 316 OML files from 48 projects were collected.

### 3.2. Extraction of OML snippets

We structured the fine-tuning process similarly to the way we would teach OML to a student: by teaching the different elements of the language in a sequential way, starting from the elements with the lowest level of complexity and following onto the most complex, step by step. To reproduce this teaching scheme, we divided the OML language reference [23] into 11 packs of growing complexity (ref. Table 2). Complexity order was established by the authors based on their experience teaching OML to students.

We did not code an extraction algorithm for each of the packs because we considered that the variability of the forms in which the OML elements can appear (e.g., in conjunction with comments, labels, etc.) would have made algorithm design convoluted and lengthy. Instead, we employed an LLM to extract appearances of elements of each of the OML packs for each of the gathered OML files. We used prompt engineering, progressively refining LLM prompts for pack extraction instructions. The prompts included a description of the pack's elements, what to include and what not to, the pack elements' Backus-Naur Form (BNF) [24], and a few examples of elements' instances. We used the open-source, 120B parameter LLM *gpt-oss-120b* from OpenAI [25], running on top of an NVIDIA H100 GPU to perform the snippet extraction.

| **Pack Num.** | **Content** | **Example** | **Num. of extracted snippets** |
|---|---|---|---|
| **#1** | Ontology declarations (vocabulary, vocabulary bundle, description, and description bundle), and imports (extends, uses, includes) | vocabulary <http://purl.org/dc/elements/1.1/> as dc {} | 307 |
| **#2** | Scalar declarations | scalar SSN | 89 |
| **#3** | Aspect and concept declarations | aspect IdentifiedElement | 80 |
| **#4** | Unreified (plain) relations | relation drives [<br>from Vehicle<br>to Person<br>functional<br>reverse drivenBy<br>] | 652 |
| **#5** | Relation entities | relation entity Ownership [<br>from Person<br>to Vehicle<br>forward owns<br>reverse ownedBy<br>functional<br>key ex:id<br>] | 141 |
| **#6** | Scalar properties | scalar property mass [<br>domain Vehicle<br>trange Mass<br>functional<br>] | 220 |
| **#7** | Key and instance enumeration axioms | concept Component [<br>key hasId<br>] | 13 |
| **#8** | Specialization and equivalence relations | concept Car < Vehicle | 771 |
| **#9** | Property restrictions | concept Component [<br>restricts performs to function1 [ hasPriority 3 ]<br>] | 258 |
| **#10** | Rules declarations | rule heavyVehicle [<br>Vehicle(x) & hasMass(x, m) -> differentFrom(m, "0"^^ex:Kilogram)<br>] | 15 |
| **#11** | Concept and relation instances | instance Car1 : Car [<br>tex:hasMass "1200"^^ex:Kilogram<br>] | 2061 |

Table 2: Categories of extracted OML snippets (packs), examples, and number of extractions per pack.

As seen in Table 2, some elements of the OML language reference are rarely captured in our available

dataset, while others are very prevalent. This may lead to imbalances in the learning process. To avoid an excessive learning on the best represented OML packs, we limited the number of snippets per pack to 1000. This is an arbitrary limit that was set to constrain the number of examples of the biggest pack within two orders of magnitude of the smallest. No sensitivity analyses were conducted to assess the impact of this limit on the fine-tuning due to the computational cost.

### 3.3. Construction of an OML training and testing dataset

The specific use case that we aim to cover with the fine-tuned LLM is the generation of OML code (ranging from small snippets to complete OML files) prompted by the user through a conversation in a chat window. The user will type instructions in natural language into the chat window, such as “Define a vocabulary for a cinema” or “Declare a concept named Thriller that specializes Movie”, and the LLM will respond with syntactically correct OML code that can be copied, after inspection by the user, into the code editor window.

To maximize the performance of the fine-tuned LLM, we fine-tuned it on a training dataset resembling our use case of interest. The dataset contains an instruction for the LLM to generate OML code given the requests of the user, and individual prompts starting with "Write a...", "Define a..", "Add a...", and "Include a...", followed by descriptions of the previously extracted OML snippets.

Due to the limited workforce and time availability, we created the dataset using an LLM. We asked an LLM to produce, for each of the pack’s OML snippets, a question that a user would likely ask for an LLM to generate such OML code snippet, following the previously specified format. The model we used was again the *gpt-oss-120b* from OpenAI. 80% of the samples produced were randomly selected to create the training set, and 20% for the test set.

### 3.4. Fine-tuning procedure

We selected the open-source Llama-3.1-8B-Instruct model from Meta to serve as a base model because of its well-established trajectory (released in 2024), broad community adoption (with more than 1600 fine-tuned versions of it produced), and open weights published in Hugging Face, which can be easily accessed for training.

We employed the *unsloth* version of the model, which can run and be fine-tuned using the Python library of the same name. Compared to the standard HuggingFace’s *transformers* library [26], *unsloth* is 2.2 times faster and uses 70% less of VRAM [27]. In addition, we set up a Low-Rank Adaptation (LoRA) fine-tuning [28]. We opted for LoRA instead of a full model fine-tuning approach because it achieves strong accuracy while reducing computational requirements. This also allowed us to establish an initial fine-tuning baseline faster and at lower cost.

LoRA generates a series of low-rank matrices with random weights and biases that are trained instead of the full base model. Once the training is finalized, the low-rank matrices’ values are transferred into the base model. This allows training only a small portion of the number of parameters of the original model, as low as around 5% depending on the rank of the LoRA matrices, while still achieving a high accuracy.

With LoRA, we fine-tuned 25% of the model’s weights and biases. The main parameters of the fine-tuning process are available in Table 3. The produced fine-tuned name is hereafter referred to as OMLlama.

| **Fine-tuning parameter** | **Value** |
|---|---|
| **Base model** | Llama-3.1-8B-Instruct |
| **Training targets** | q_proj, k_proj, v_proj, up_proj, down_proj, o_proj, gate_proj, embed_tokens, lm_head (only weights trained, no biases) |
| **LoRA matrices’ rank** | 1024 |
| **LoRA matrices’ scaling factor** | 64 |
| **Dropout** | 0.05 |
| **Percentage of base model trained** | 25% |
| **Decimal precision** | 16 bits |

| | |
|---|---|
| **Number of training epochs** | 2 |
| **General learning rate** | 0.0002 |
| **Embedding layers' learning rate** | 0.000005 |
| **Learning rate scheduler** | Cosine with restarts [29] |
| **Optimizer** | *adamw_torch* [30] (16 bits) |

*Table 3: Main parameters of the fine-tuning process.*

### 3.5. Selection of benchmarking models

We selected a series of open-source, general-purpose LLMs (covering major ones such as OpenAI, Meta, or Google) and sizes (ranging from 7 B to 120 billion parameters) (ref. Table 4) as models against which we compared the fine-tuned model accuracy. Our selection was based on the model's availability in Ollama [31], and a model size limit set to 120 billion parameters to accommodate the model on a single NVIDIA H100 GPU.

| Model name | Provider | Number of parameters |
|---|---|---|
| **GPT OSS** [25] | OpenAI | 120 B |
| **Llama 3.3** [32] | Meta | 70 B |
| **Qwen 3** [33] | Alibaba | 32 B |
| **Gemma 3** [34] | Google | 27 B |
| **Phi 4** [35] | Microsoft | 14 B |
| **Llama 3.1** [36] | Meta | 8 B |
| **Mistral 7B** [37] | Mistral AI | 7 B |
| ***OMLlama*** | *Authors of this paper* | *8 B* |

Table 4: List of evaluated Large Language Models.

### 3.6. Selection of evaluation metrics

To evaluate the results, we selected both lexical similarity and semantic similarity metrics that compare the LLM-produced output with the expected, ground truth output. These input-output pairs, available in the test dataset (a random selection of the 20% of the OML dataset), were not part of the training and thus, have not been seen by the fine-tuned model before.

For semantic similarity, we measure cosine similarity, which captures the distance between two text embeddings $\overrightarrow{E_1}$ and $\overrightarrow{E_2}$ as depicted by Equation 1. One embedding corresponds to the LLM-generated text, and the other to the ground truth text from the test dataset. For the embeddings, we have selected the *all-MiniLM-L6-v2* sentence transformer due to its popularity [38], and for the cosine similarity calculation, scikit-learn's function [39].

$$\text{cosine_similarity}(\overrightarrow{E_1}, \overrightarrow{E_2}) = \cos\left(\theta_{\overrightarrow{E_1},\overrightarrow{E_2}}\right) = \frac{\overrightarrow{E_1}\cdot\overrightarrow{E_2}}{\|\overrightarrow{E_1}\|\|\overrightarrow{E_2}\|} = \frac{\sum_{i=0}^{n} E_{1,i}E_{2,i}}{\sqrt{\sum_{i=0}^{n} {E_{1,i}}^2}\sqrt{\sum_{i=0}^{n} {E_{2,i}}^2}} \qquad (1)$$

For lexical similarity, we use the well-established BLEU [40] and ROUGE [41] metrics. These metrics consider the frequency with which *n* adjacent symbols (named "n-grams") appear in two different texts.

BLEU measures how many *n*-grams in the generated text appear in the reference text as well (i.e., the precision), as depicted by Equation 2. The weighed sum of *n*-gram's precision $p_n$ is smoothed by a factor that penalizes the brevity of the text, $BP$. While the typical usage of BLEU is in its BLEU-4 variant, we have focused on BLUE-1, which only looks at unigram precision, because of the compared texts being short, sometimes containing only 2 words. In this case, $\omega_n = [1,0,0,0]$. We used the *NLTK* library's *bleu_score* function with its default smoothing method [42].

$$BLEU = BP \exp\left(\sum_{n=1}^{N} \omega_n \ln p_n\right) \qquad (2)$$

ROUGE measures the number of *n*-grams from the reference text that also exist in the generated text

(i.e. the recall), as shown in Equation 3. Due to the texts compared being short, we have focused on unigram recall, calculating ROUGE-1. We used Python's *rouge_score* function [43].

$$ROUGE = \frac{\sum_{Ref}\sum_{Output} Count_Matches(gram_n)}{\sum_{Ref}\sum_{Output} Count(gram_n)} \quad (3)$$

To understand how different numerical scores capture the accuracy of the evaluated models, we also look at the generated OML code snippets for the least-performing LLM, an intermediate one, and the best-performing.

## 4. Results and discussion

### 4.1. Results of the fine-tuning process

During the fine-tuning and creation of the OMLlama model, a total of 2,836 interleaved OML samples were used. The training's two epochs were completed in 14.4 minutes, at a pace of 10 training samples per second on top of the NVIDIA H100 GPU.

The loss was significantly reduced during the fine-tuning process, early within the initial epochs and training steps, as shown by Fig. 1.

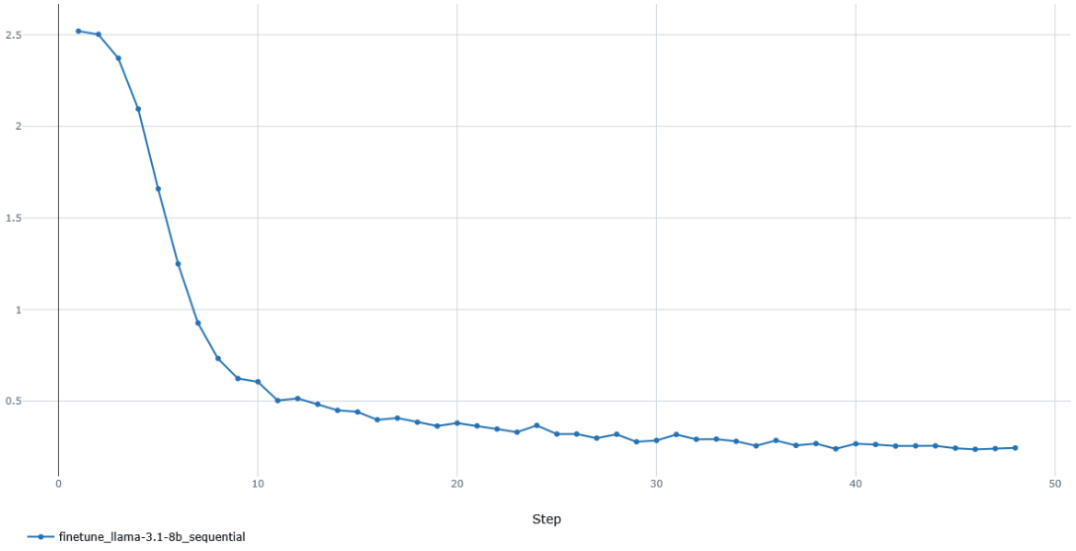



Fig. 1: Evolution of the loss function's value during the fine-tuning process.

### 4.2. Comparisons of benchmarking models and OMLlama

In Table 5, we show the results of running the test dataset with the pre-selected general-purpose LLMs and our OMLlama and measuring the outputs through the lexical and semantical metrics. The metrics' values are shown averaged over the 11 packs.

| Model name | Number of parameters | BLEU-1 score | ROUGE-1 score | Cosine similarity |
|---|---|---|---|---|
| **GPT OSS** | 120 B | 0.606 | 0.878 | 0.913 |
| **Llama 3.3** | 70 B | 0.586 | 0.868 | 0.914 |
| **Qwen 3** | 32 B | 0.557 | 0.881 | 0.916 |
| **Gemma 3** | 27 B | 0.560 | 0.852 | 0.884 |
| **Phi 4** | 14 B | 0.377 | 0.800 | 0.855 |
| **Llama 3.1** | 8 B | 0.452 | 0.825 | 0.894 |

| | | | | |
|---|---|---|---|---|
| **Mistral 7B** | 7 B | 0.284 | 0.705 | 0.825 |
| ***OMLlama*** | 8 B | **0.864** | **0.943** | **0.953** |

Table 5: Results of the lexical and semantical evaluation metrics for the generic-purpose LLMs and fine-tuned OMLlama.

One of the first observations is that OMLlama, our fine-tuned model, outperforms all general-purpose models across all metrics. This responds to RQ1: the gap between smaller LLMs and bigger LLMs can effectively be bridged by fine-tuning. This opens a way for domain-specialized models that can deliver, at a fraction of the inference cost, results that are more accurate than those of the bigger models.

### 4.3. Comparison of OML snippet generation

We have looked at the generated OML code snippets for the least-performing LLM (Mistral 7B), an intermediate one (Qwen 3, 32B), and our fine-tuned OMLlama. *Table 6Table 6*Tables 6-9 show example prompts pertaining to Packs 2, 4, 6, and 8, covering scalars, relations, scalar properties, and concept specialization definitions, respectively. Each table shows the expected response to the prompt (ground truth), the retrieved response from the three selected LLMs, the associated metrics' values, and the difference between the response and the ground truth.

| **Write a scalar named nonNegativeInteger that extends integer and include the annotation @dc:source with the URL "https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/datatypes.html#nonNegativeInteger".** | | | |
|---|---|---|---|
| **Expected** | @dc:source "https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/datatypes.html#nonNegativeInteger"<br>scalar nonNegativeInteger < integer | | |
| **Mistral 7B** | import owl:Ontology;<br>import rdf:RDF;<br>import xsd:xsd;<br>Class nonNegativeInteger extends xsd:integer;<br>annotation ( rdf:type rdfs:Resource ;<br>dc:source <https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/datatypes.html#nonNegativeInteger > ) | BLEU-1: 0.05<br>ROUGE-1: 0.65<br>Cosine sim.: 0.68 | Uses incorrect syntax (e.g. "import" beyond ontologies, and "Class"), does not refer to a scalar, adds unwanted imports. |
| **Qwen 3** | scalar nonNegativeInteger extends integer {<br>@dc:source("https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/datatypes.html#nonNegativeInteger")<br>} | BLEU-1: 0.43<br>ROUGE-1: 0.97<br>Cosine sim.: 0.87 | Uses incorrect syntax (e.g. "extends" beyond ontologies). |
| **OMLlama** | @dc:source "https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/datatypes.html#nonNegativeInteger"<br>scalar nonNegativeInteger < integer | BLEU-1: 1.0<br>ROUGE-1: 1.0<br>Cosine sim.: 1.0 | Accurate answer with perfect syntax, matching the ground truth. |

*Table 6: Example of a Pack 2 (scalars) prompt, responses from different models, and associated metrics' scores.*

| **Write a relation named hasConstraint that goes from Requirement to Constraint that is inverse functional.** | | | |
|---|---|---|---|
| **Expected** | relation hasConstraint [<br>from Requirement<br>to Constraint<br>inverse functional<br>] | | |
| **Mistral 7B** | hasConstraint a relation with<br>domain Requirement<br>range Constraint<br>inverseFunctional true; | BLEU-1: 0.40<br>ROUGE-1: 0.44<br>Cosine sim.: 0.87 | Uses incorrect syntax such as "with" and extraneous elements "domain", "range"; misses "from", "to". |
| **Qwen 3** | relation hasConstraint: Requirement -> Constraint [inverseFunctional] | BLEU-1: 0.26<br>ROUGE-1: 0.61<br>Cosine sim.: 0.93 | Uses incorrect syntax such as "->" and misses "from" and "to". |
| **OMLlama** | relation hasConstraint [<br>from Requirement<br>to Constraint<br>inverse functional<br>] | BLEU-1: 1.0<br>ROUGE-1: 1.0<br>Cosine sim.: 1.0 | Accurate answer with perfect syntax, matching the ground truth. |

Table 7: Example of a Pack 4 (relations) prompt, responses from different models, and associated metrics' scores.

| **Write a scalar property named ActivityEdge_kindOfRate with domain ActivityEdge and range common.data.core:RateKind, marked as functional, and include an rdfs:label annotation set to "kindOfRate".** | | | |
|---|---|---|---|
| **Expected** | @rdfs:label "kindOfRate"<br>scalar property ActivityEdge_kindOfRate [<br>domain ActivityEdge<br>range common.data.core:RateKind<br>functional<br>] | | |
| **Mistral 7B** | ```<br>property ActivityEdge_kindOfRate:<br>common.data.core:RateKind;<br>rdfs:label "kindOfRate";<br>functional;<br>``` | BLEU-1: 0.08<br>ROUGE-1: 0.85<br>Cosine sim.: 0.88 | Uses incorrect syntax (```, : and ;), does not refer to a scalar property, forgets “domain” and “range”. |
| **Qwen 3** | scalar property ActivityEdge_kindOfRate : ActivityEdge -><br>common.data.core:RateKind<br>functional<br>rdfs:label "kindOfRate" | BLEU-1: 0.57<br>ROUGE-1: 0.93<br>Cosine sim.:0.94 | Uses incorrect syntax (“->”), forgets “domain” and “range” |
| **OMLlama** | @rdfs:label "kindOfRate"<br>scalar property ActivityEdge_kindOfRate [<br>domain ActivityEdge<br>range common.data.core:RateKind<br>functional<br>] | BLEU-1: 1.0<br>ROUGE-1: 1.0<br>Cosine sim.: 1.0 | Accurate answer with perfect syntax, matching the ground truth. |

*Table 8: Example of a Pack 8 (scalar properties) prompt, responses from different models, and associated metrics' scores.*

| **Write an aspect named IState that specializes AbstractNamedElement and include an annotation label “Istate”.** | | | |
|---|---|---|---|
| **Expected** | @rdfs:label "IState"<br>aspect IState < AbstractNamedElement | | |
| **Mistral 7B** | IState aspect Specializes AbstractNamedElement {<br>Annotation label "IState";<br>} | BLEU-1: 0.33<br>ROUGE-1: 0.77<br>Cosine sim.: 0.73 | Uses incorrect syntax (“;”, “specializes”). |
| **Qwen 3** | scalar xsd:boolean specializes rdfs:Literal {<br>annotation dc:source “https://www.w3.org/TR/owl2-syntax/#Boolean_Values”<br>} | BLEU-1: 0.44<br>ROUGE-1: 0.91<br>Cosine sim.: 0.93 | Uses incorrect syntax (“specializes”),unwanted URL, refers to a scalar and not aspect. |
| **OMLlama** | aspect IState < AbstractNamedElement<br>@rdfs:label "IState" | BLEU-1: 1.0<br>ROUGE-1: 1.0<br>Cosine sim.: 0.98 | The label is added after the aspect specialization. |

Table 9: Example of a Pack 8 (specializations) prompt, responses from different models, and associated metrics' scores.

Mistral 7B, the worst performing one, systematically outputs verbose code with incorrect OML syntax. While Qwen 3 syntax is still incorrect, the output resembles more the OML syntax. Finally, OMLlama’s answers are almost always identical to the ground truth, albeit some mismatches are observed (for instance, in Table 9, a label is added after an OML element declaration, instead of before).

### 4.4. Similarity evolution with increasing complexity

We also investigated whether there was a difference in the OML modeling accuracy that LLMs show when faced with OML packs of increasing complexity. Fig. 2 shows the evolution of cosine similarity, BLEU-1, and ROUGE-1 metrics (averaged across LLMs) from pack one (lowest complexity) to pack eleven (highest complexity). Packs with less than 10 samples in the test dataset have been removed to avoid contamination of the results from non-statistically significant samples.

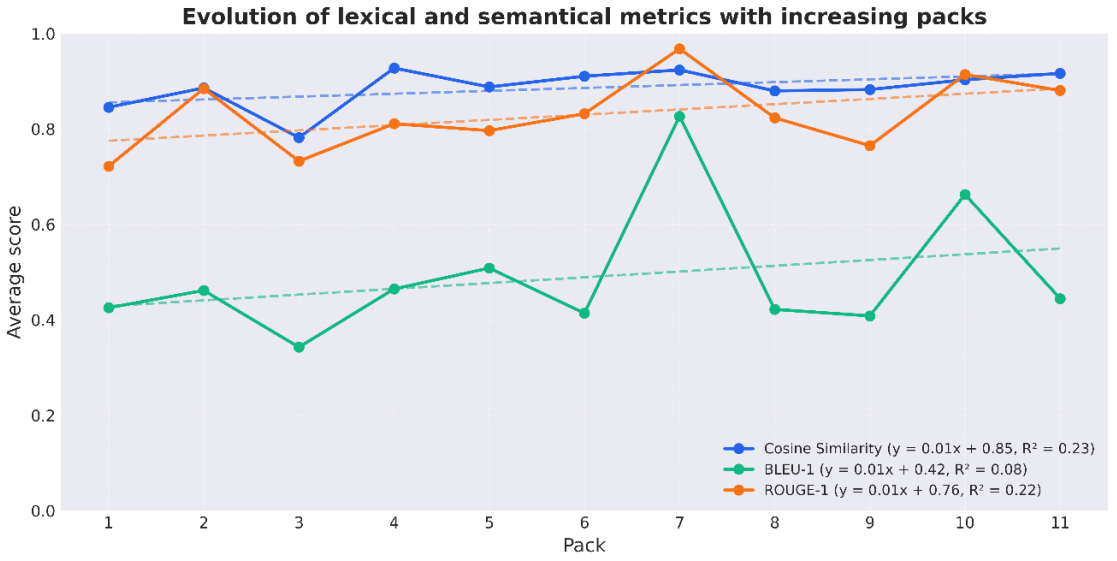



Fig. 2: Evolution of the average metric's values across packs of increasing OML complexity.

Fig. 2 does not indicate a clear relation between OML complexity and LLM accuracy. If any, it shows a slight inverse relation, which is unexpected, albeit with very weak evidence (with $R^2$ values ranging from 0.08 to 0.23). This may suggest that the setup of the OML packs as explained (Table 2) does not properly follow a language complexity progression, or rather, what humans perceive as complex for learning differs from learning complexity for machines.

## 5. Conclusion and limitations

In this paper, we have presented a methodology to fine-tune an open-source Large Language Model (LLM), the Llama 3.1 Instruct 8B, for Ontological Modeling Language (OML) code generation, creating the domain-specific OMLlama.

We have shown that, even for a small language with few publicly available repositories, fine-tuning techniques such as LoRA can be applied leading to a significant increase of accuracy and placing the specialized model above general-purpose ones. The accuracy results and the presented examples highlight the opportunities that exist in the creation of specialized LLMs for SE modeling, going beyond the more widely applied Prompt Engineering and Retrieval Augmented Generation techniques.

However, a limited understanding of the relation between the LLM size or the OML complexity and LLM accuracy calls for additional research into the topic. In our paper, we only tested open-source LLMs, leaving aside proprietary LLMs such as GPT, Claude, or Gemini, which are more performant and can be accessed via external APIs. In addition, we only compared general-purpose LLMs, leaving behind code-specific ones such as Codellama or Codegemma. An extension of the evaluations into these sets of LLMs would improve the number of data points in our study.

From the data side, a rework into the classification of OML into different packs, achieving more granular packs with a more progressive learning curve, would improve the study of the relation between OML complexity and LLM accuracy. And from the training side, the accuracy of the fine-tuned LLM being high can come at the cost of overfitting, which could be further studied to established if there has been a loss of generalization capabilities.

Lastly, fine-tuning a specialized LLM is only an initial step to making it useful for a domain-specific task, and it precedes integration with development tools and establishing robust methodologies. In future work, we aim to demonstrate the usage of OMLlama in the context of an Integrated Development Environment (IDE) through a CoPilot plugin, showing its usefulness on a real modeling use case.

## References


[1] C. Ling *et al.*, “Domain Specialization as the Key to Make Large Language Models Disruptive: A Comprehensive Survey,” *ACM Comput. Surv.*, vol. 58, no. 3, p. 79:1-79:39, Oct. 2025, doi: 10.1145/3764579.

[2] C. Team *et al.*, “CodeGemma: Open Code Models Based on Gemma,” June 19, 2024, *arXiv*: arXiv:2406.11409. doi: 10.48550/arXiv.2406.11409.

[3] “Introducing Code Llama, a state-of-the-art large language model for coding,” Meta AI. Accessed: Nov. 17, 2025. [Online]. Available: https://ai.meta.com/blog/code-llama-large-language-model-coding/

[4] “Codestral | Mistral AI.” Accessed: Nov. 17, 2025. [Online]. Available: https://mistral.ai/news/codestral

[5] M. Liu *et al.*, “ChipNeMo: Domain-Adapted LLMs for Chip Design,” Apr. 04, 2024, *arXiv*: arXiv:2311.00176. doi: 10.48550/arXiv.2311.00176.

[6] H. Zou *et al.*, “TelecomGPT: A Framework to Build Telecom-Specfic Large Language Models,” July 12, 2024, *arXiv*: arXiv:2407.09424. doi: 10.48550/arXiv.2407.09424.

[7] K. A. Rafique, S. Shah, Š. Dalecke, and C. Grimm, “Enhancing Model-Based Systems Engineering with Large Language Models,” *INCOSE International Symposium*, vol. 35, no. 1, pp. 1523–1543, 2025, doi: 10.1002/iis2.70067.

[8] F. Pan, V. Zolfaghari, L. Wen, N. Petrovic, J. Lin, and A. Knoll, “Generative AI for OCL Constraint Generation: Dataset Collection and LLM Fine-tuning,” in *2024 IEEE International Symposium on Systems Engineering (ISSE)*, Oct. 2024, pp. 1–8. doi: 10.1109/ISSE63315.2024.10741141.

[9] “Ontological Modeling Language v2.” Accessed: Nov. 17, 2025. [Online]. Available: https://www.opencaesar.io/oml/

[10] A. Sreekumar, “Developing an OML Co-Pilot - exploring AI-assisted Ontological Modeling.” Accessed: Nov. 17, 2025. [Online]. Available: https://www.opencaesar.io/events/onto-Nexus-Forum-2025/Talk-21

[11] V. B. Parthasarathy, A. Zafar, A. Khan, and A. Shahid, “The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities,” Oct. 30, 2024, *arXiv*: arXiv:2408.13296. doi: 10.48550/arXiv.2408.13296.

[12] W. Zhang *et al.*, “MBSE Co-Pilot: A Research Roadmap,” *Systems Engineering*, vol. n/a, no. n/a, p. e70011, 2025, doi: 10.1002/sys.70011.

[13] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, “A Survey on Large Language Models for Code Generation,” *ACM Trans. Softw. Eng. Methodol.*, July 2025, doi: 10.1145/3747588.

[14] S. Joel, J. Wu, and F. Fard, “A Survey on LLM-based Code Generation for Low-Resource and Domain-Specific Programming Languages,” *ACM Trans. Softw. Eng. Methodol.*, Oct. 2025, doi: 10.1145/3770084.

[15] T. B. Brown *et al.*, “Language models are few-shot learners,” in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, in NIPS ’20. Red Hook, NY, USA: Curran Associates Inc., Dec. 2020, pp. 1877–1901.

[16] P. Lewis *et al.*, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2020, pp. 9459–9474. Accessed: Dec. 04, 2025. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[17] Z. Li, S. Husung, and H. Wang, “LLM-Assisted Semantic Alignment and Integration in Collaborative Model-Based Systems Engineering Using SysML v2,” Aug. 22, 2025, *arXiv*: arXiv:2508.16181. doi: 10.48550/arXiv.2508.16181.

[18] Y. Wang, N. Ge, J. Liu, Z. Cao, Z. Chen, and C. Hu, “Generating SysML Behavior Models via Large Language Models: an Empirical Study,” in *Proceedings of the 16th International Conference on Internetware*, in Internetware ’25. New York, NY, USA: Association for Computing Machinery, Oct. 2025, pp. 366–377. doi: 10.1145/3755881.3755926.

[19] E. Cibrián, J. Olivert-Iserte, J. Llorens, and J. M. Álvarez-Rodríguez, “An agent-based approach for the automatic generation of valid SysMLv2 Models in industrial contexts,” *Computers in Industry*, vol. 172, p. 104350, Nov. 2025, doi: 10.1016/j.compind.2025.104350.

[20] R. Wu, G. Wen, X. Zhang, X. Liang, C. Yang, and W. Gui, “Knowledge-Augmented Conversational Agents for Dependency-Aware SysML v2 Model Generation and Refinement,” in *2025 IEEE 26th China Conference on System Simulation Technology and its Applications (CCSSTA)*, July 2025, pp. 581–586. doi: 10.1109/IEEECONF65522.2025.11137199.

[21] R. B. Ryan Longshore and R. Madachy, “Leveraging Generative AI to Modify and Query MBSE Models,” Acquisition Research Program, Technical Report, May 2024. Accessed: Nov. 17, 2025. [Online]. Available: https://dair.nps.edu/handle/123456789/5113

[22] “openCAESAR,” GitHub. Accessed: Nov. 17, 2025. [Online]. Available: https://github.com/opencaesar

[23] “Ontological Modeling Language v2.” Accessed: Dec. 04, 2025. [Online]. Available: https://www.opencaesar.io/oml/

[24] D. D. McCracken and E. D. Reilly, “Backus-Naur form (BNF),” in *Encyclopedia of Computer Science*, GBR: John Wiley and Sons Ltd., 2003, pp. 129–131.

[25] OpenAI *et al.*, “gpt-oss-120b & gpt-oss-20b Model Card,” Aug. 08, 2025, *arXiv*: arXiv:2508.10925. doi: 10.48550/arXiv.2508.10925.

[26] “Transformers.” Accessed: Dec. 04, 2025. [Online]. Available: https://huggingface.co/docs/transformers/index

[27] “Unsloth Docs | Unsloth Documentation.” Accessed: Nov. 18, 2025. [Online]. Available: https://docs.unsloth.ai

[28] E. J. Hu *et al.*, “LoRA: Low-Rank Adaptation of Large Language Models,” Oct. 16, 2021, *arXiv*: arXiv:2106.09685. doi: 10.48550/arXiv.2106.09685.

[29] I. Loshchilov and F. Hutter, “SGDR: Stochastic Gradient Descent with Warm Restarts,” arXiv.org. Accessed: Dec. 10, 2025. [Online]. Available: https://arxiv.org/abs/1608.03983v5

[30] I. Loshchilov and F. Hutter, “Decoupled Weight Decay Regularization,” Jan. 04, 2019, *arXiv*: arXiv:1711.05101. doi: 10.48550/arXiv.1711.05101.

[31] “Ollama.” Accessed: Nov. 18, 2025. [Online]. Available: https://ollama.com

[32] “Llama 3.3 | Model Cards and Prompt formats.” Accessed: Nov. 19, 2025. [Online]. Available: https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_3/

[33] A. Yang *et al.*, “Qwen3 Technical Report,” May 14, 2025, *arXiv*: arXiv:2505.09388. doi: 10.48550/arXiv.2505.09388.

[34] G. Team *et al.*, “Gemma 3 Technical Report,” Mar. 25, 2025, *arXiv*: arXiv:2503.19786. doi: 10.48550/arXiv.2503.19786.
[35] M. Abdin *et al.*, “Phi-4 Technical Report,” Dec. 12, 2024, *arXiv*: arXiv:2412.08905. doi: 10.48550/arXiv.2412.08905.
[36] A. Grattafiori *et al.*, “The Llama 3 Herd of Models,” Nov. 23, 2024, *arXiv*: arXiv:2407.21783. doi: 10.48550/arXiv.2407.21783.
[37] A. Q. Jiang *et al.*, “Mistral 7B,” Oct. 10, 2023, *arXiv*: arXiv:2310.06825. doi: 10.48550/arXiv.2310.06825.
[38] “sentence-transformers/all-MiniLM-L6-v2 · Hugging Face.” Accessed: Nov. 19, 2025. [Online]. Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
[39] “cosine_similarity,” scikit-learn. Accessed: Nov. 19, 2025. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html
[40] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “BLEU: a method for automatic evaluation of machine translation,” in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL ’02*, Philadelphia, Pennsylvania: Association for Computational Linguistics, 2001, p. 311. doi: 10.3115/1073083.1073135.
[41] C.-Y. Lin, “ROUGE: A Package for Automatic Evaluation of Summaries”.
[42] “NLTK :: nltk.translate.bleu_score.” Accessed: Nov. 19, 2025. [Online]. Available: https://www.nltk.org/_modules/nltk/translate/bleu_score.html
[43] *rouge-score: Pure python implementation of ROUGE-1.5.5.* Python. Accessed: Nov. 19, 2025. [OS Independent]. Available: https://github.com/google-research/google-research/tree/master/rouge