Evaluating Template- and Model-Guided LLMs for Requirements Specification

Ikram Darif*‡, Zainab Benlagote†§, and Ghizlane El Boussaidi†¶
* University of Ottawa, Ottawa, Canada.
† École de technologie supérieure, Montreal, Canada
Email: ‡idarif@uottawa.ca, §zainab.benlagote.1@ens.etsmtl.ca, ¶ghizlane.elboussaidi@etsmtl.ca

***Abstract*—Requirements specification is a pivotal activity in software development. The use of structured templates for specifying requirements can improve their clarity, consistency, and analyzability, which is particularly important in safety-critical domains such as avionics to support verification and certification activities. However, template-based requirements specification is typically a manual, resource-intensive process that requires domain expertise. Large Language Models (LLMs) provide promising capabilities for automating this task, especially when augmented with domain knowledge, yet their effectiveness in template-based requirement specification remains unexplored. In this paper, we present an empirical evaluation of LLMs for generating template-based functional requirements from raw textual specifications, and we investigate the impact of enriching LLM inputs with explicit domain models compared to unstructured textual context. We conducted 138 experiments evaluating three LLMs with multiple prompting strategies on two avionics datasets containing 146 requirements. The results indicate that simple few-shot prompting is the most effective strategy for this task. Although domain models improve execution efficiency, the gains over unstructured text are not substantial enough to justify the overhead of creating and maintaining those models.**

***Index Terms*—Requirement Specification, Requirement Templates, Model-Driven Engineering, Large Language Models**

I. Introduction

Requirements are fundamental software artifacts [1], particularly in safety-critical systems (SCS) which must comply with strict certification standards. In industrial contexts, requirements are typically specified in Natural Language (NL), which can introduce ambiguity and inconsistency. In fact, approximately 70% of software defects originate from faulty requirements, further highlighting the importance of accurately specified requirements [2]. Controlled Natural Language (CNL) addresses these issues by constraining the vocabulary, syntax, and semantics of a base language through predefined templates with fixed parts and placeholders [3], thereby improving requirement quality while preserving readability [4].

With the rapid advancement of Artificial Intelligence (AI), Large Language Models (LLMs) have been widely deployed to support various software engineering phases, particularly Requirements Engineering (RE) [5]. LLMs are advanced models with billions of parameters trained on vast datasets, featuring sophisticated capabilities in NL processing and generation [6], [7]. These capabilities make them highly relevant for RE in general, and requirements specification in particular. Furthermore, they are capable of processing structured data, such as models, thereby enabling the implementation of Model-Driven Engineering (MDE) within LLM-based pipelines. Leveraging MDE facilitates the augmentation of the LLM with structured domain context, while facilitating and automating downstream RE tasks such as testing [8].

While various CNL approaches for requirements specification have been proposed in literature (e.g., [4], [9]–[12]), they do not leverage the reasoning capabilities of LLMs. On the other hand, the majority of LLM-based requirements specification approaches do not utilize CNL. Furthermore, they often do not rely on domain knowledge to ensure requirement correctness, and rarely incorporate structured domain knowledge, such as domain models, into the specification process.

In this paper, we empirically evaluate the use of LLMs for generating template-based requirement specification. Specifically, we investigate their ability to reformulate raw textual requirements into the Operational Behavior Template, a functional-requirement template introduced in our previous work [4], [13]. We also examine whether domain models provide greater benefits than unstructured textual context when enriching prompts. We compare multiple prompting strategies: zero-shot, few-shot, traditional RAG, domain-model-enhanced (DM-enhanced) RAG, and hybrid approaches combining few-shot prompting with RAG. The evaluation comprises 138 experiments evaluating three models *GPT5.4*, *Llama 3.1:8B* and *Mistral:7B* across two datasets derived from the ARINC-653 standard [14] totaling 146 requirements. All experiments were assessed across three dimensions: logical decomposition, structural reasoning, and semantic grounding.

The results show that *GPT-5.4* outperforms *Llama* and *Mistral* across all experiments and prompting variations. The highest performance was achieved through few-shot prompting, suggesting that LLMs prioritize illustrative examples over domain context for this task, as they provide both task-related and domain-specific guidance. This advantage was consistent across all evaluation dimensions. An optimal configuration of eight examples achieved an overall correctness score of 0.83 and 0.75 across the two datasets. On the other hand, while leveraging domain models instead of unconstrained textual context enabled execution time savings, performance improvements over traditional RAG were marginal (e.g., 2.8% and 1.5%). This suggests that the overhead of creating and maintaining domain models might not be justified for this task. Finally, combining few-shot with RAG (both traditional and

DM-enhanced) yielded lower performance than simple few-shot prompting, indicating that excessive context may have introduced noise and confused the models.

The remainder of the paper is organized as follows. Section II presents the related work. Section III presents the background of the study. In Section IV, we outline the research method. Section V and Section VI present the evaluation results and provide a discussion of our findings, respectively. Finally, Section VII outlines the threats to validity and Section VIII concludes the paper and presents the future work.

II. Related Work

Several approaches and tools have been proposed to support the semi-formal specification of requirements using templates [4], [9]–[12]. For instance, FRET is a tool developed by NASA to facilitate the specification and analysis of requirements using a CNL called FRETis, while automating the translation of CNL requirements into temporal logic through a set of specialized algorithms [10]. MD-RSuT [4] is a tool for model-driven requirements specification based on seven templates derived from different certification standards. The tool supports the specification and auto-filling of requirements using domain models [4]. Similarly, the works in [11]and [12] provide term suggestions to assist in template completion, with the latter leveraging ontologies and NLP techniques for term selection. While existing approaches partially automate template-based requirements specification, they still require substantial manual effort for template completion and correctness verification. Moreover, they rarely exploit MDE, despite its demonstrated benefits for downstream RE tasks. Leveraging LLMs for template-based specification could effectively address these limitations through their advanced reasoning capabilities.

Several approaches explored the use of LLMs in RE [5], [15]–[21]. Existing work focused on generating requirements artifacts, such as user stories [17] and specification documents [16], as well as supporting requirements auto-completion [18], formalization [19], and quality assurance through clarity and correctness verification [20], [21]. However, these approaches generally do not target CNL requirements and rarely rely on templates. Moreover, approaches that leverage formal models primarily focus on translating requirements into formal specifications rather than improving NL requirements. To the best of our knowledge, no existing approach combines the reasoning capabilities of LLMs with MDE to support template-based NL requirements specification.

III. Background

In our previous research [4], [13], we proposed RESPECT, a model-driven and template-based approach for requirements specification and verification. RESPECT leverages MDE techniques to: (1) model requirement templates, and (2) link the template models to domain models in order to support requirements verification and auto-filling. RESPECT establishes a systematic process for the creation of templates, which was applied on ARINC-653, a standard from the avionics domain used for the certification of real-time operating systems [14].

> [<condition>
> [AND <condition>]* (one per line)
> Then]
> <operational behavior action>

Fig. 1. The operational Behavior Template

This application resulted in a set of seven templates covering both functional and data requirements [4].

To support RESPECT, we developed MDRSuT, an editor that enables template-based requirements specification and auto-completion using domain models. RESPECT was later extended to support the automated generation of test artifacts, further demonstrating the benefits of combining templates with MDE [8]. Despite these capabilities, requirements specification remains labor-intensive, as most template placeholders must still be completed manually. This study investigates whether LLMs can reduce this effort by automating template completion. We focus on the Operational Behavior Template, the most commonly used RESPECT template for specifying functional requirements [4]. The following sections introduce the two RESPECT components used in our study: the Operational Behavior Template and the domain models.

A. Operational Behavior Template

The operational behavior template specifies functional requirements, describing the behavior of a system or its components. As Figure 1 shows, it describes an action that an entity performs either globally or when condition(s) apply. An operational requirement includes zero or multiple conditions (denoted by the "*" notation), and exactly one action.

The Operational Behavior Template supports five condition types (Figure 2): *continuous-action, discontinuous-action, event-driven, value-related, and range-related* conditions. *Continuous-action*, *discontinuous-action*, and *event-driven* conditions describe actions occurring over time, intermittently triggering another action, or being triggered by a specific event, respectively. They share a common structure consisting of a *subject*, an *action verb*, an *object*, and optional *details*, differing mainly in their conditional keywords (while, when, and if), verb tense, and organization. *Value-related* and *range-related* conditions express constraints on parameter values and ranges. Both are defined by a *parameter name*, a *value* or *range*, and optional associated *entities* and *details*.

The operational behavior template supports two types of actions: *Operational actions*, describing an action that a subject performs on an object, and *Value-related actions*, which specify a subject setting a parameter to a certain value. Table I provides examples of ARINC-653 requirements specified according to the operational behavior template. Specifically, *R1* illustrates a requirement that includes a discontinuous action-driven condition, a value-related condition, and an operational action, *R2* illustrates a continuous-action driven condition, an

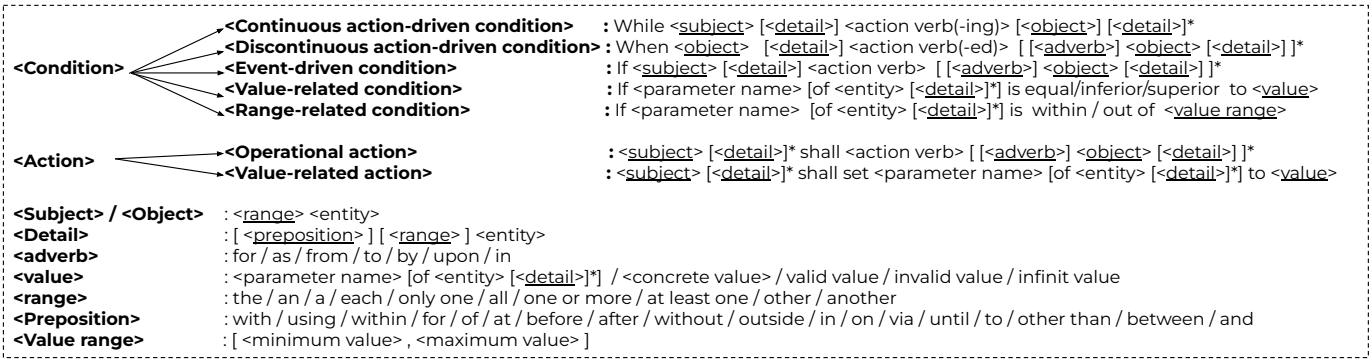

Fig. 2. The templates for conditions and actions

TABLE I
EXAMPLES OF OPERATIONAL BEHAVIOR REQUIREMENTS

ID	Specified Requirement
R1	When the **OS calls** the **RAISE_APPLICATION_ERROR service** AND If **ERROR_CODE** is equal to **DEADLINE_MISSED** Then the **OS** shall set **RETURN_CODE** to **INVALID_PARAM**
R2	While the **error handler process** is **executing** AND If The **ACQUIRE_MUTEX service** is **called** Then The **ACQUIRE_MUTEX service** shall **return** an **error**
R3	If **LOCK_LEVEL** of **a process within the partition** is superior to **0** Then The **OS** shall **schedule** the **error handler process**

event-driven condition, and a value-related action, and *R3* illustrates a value-related condition and an operational action.

B. Domain Models For Requirement Auto-filling

In SCS, requirements correctness is essential, as defects increase development costs, delay projects, and may compromise certification. This motivates the need for the verification of requirements against domain knowledge, or ideally, ensuring their correct-by-construction specification. Domain models help ensure correctness by providing a structured representation of domain knowledge through classes, attributes, types, and associations. Figure 3 presents a fragment of the ARINC-653 domain model for the Health monitor service, which is responsible for error detection and recovery. For example, the type *ErrorCodeType* defines valid error codes, while associations capture relationships between domain concepts, such as the *Error handler process* and the the *Process-level errors*.

In our previous work, we automatically filled placeholders in data requirements using predefined mapping rules between data requirement templates and domain models. However, this approach did not apply to the Operational Behavior Template, as its placeholders do not consistently map to domain model concepts. Since it is the most frequently used template, grounding its specification in domain knowledge remains important.

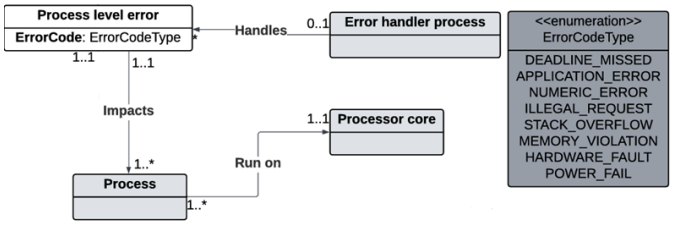

Fig. 3. Fragment of the ARINC-653 Health Monitor domain model

IV. RESEARCH METHOD

The goal of this empirical evaluation is to assess the effectiveness of LLMs in supporting template-based requirements specification, as well as the impact of domain models on the specification task. In this section, we detail the experimental setup. First, we define the research questions addressed by this study, the datasets used for the experiments, and the selected LLMs. Then, we outline the experimental configurations and the prompting techniques. Finally, we present the metrics used for analyzing the results of the experiments.

A. Research Questions

RQ.1: To what extent do State Of The Art (SOTA) LLMs and different prompting strategies effectively generate template-based requirements specifications? This RQ aims to evaluate and compare the performance of zero-shot, few-shot, and RAG across three SOTA LLMs for template-based requirements specification.

RQ.2: How does the use of domain models as RAG context compare to unstructured textual context for template-based requirements specifications? This RQ investigates whether domain models improve LLM performance and execution time compared to textual context.

B. Datasets

We perform our evaluation on two datasets from the avionics domain. Specifically, we cover two services of the ARINC-653 certification: the Health Monitor (HM) and Intra-Partition Communication (IPC) [14]. The HM is responsible for detecting and handling errors, while the IPC service manages

all communication between processes of a partition (i.e., a logically isolated execution unit of the application). These services were selected as: (1) they include requirements that differ in terms of content and structure, (2) their original requirements were reformulated using the Operational Behavior Template in our prior work [4], producing a ground truth (GT) specification, and (3) we have already developed the domain models for these services in our previous research [4], [22]. Both the specified requirements and domain models were evaluated and validated by an RE expert and our industrial partner. Both datasets were published as part of our previous work and are available through [23].

The HM dataset contains 30 input requirements refined into 53 operational behavior requirements, while the IPC dataset includes 68 input requirements refined into 93 requirements. Overall, the evaluation covers **98 input requirements** reformulated into **146 operational behavior requirements**.

C. Selected LLMs

To evaluate the performance of LLMs for template-based requirements specification, we selected three models which were frequently used in requirements specification research [5], [16], [17]: *GPT*, *Llama*, and *Mistral*. The GPT family has been shown in the literature to be highly effective in RE-related tasks; thus, *GPT-5.4* was selected as the most recent model. Similarly, *Llama* and *Mistral* represent prominent open-source models that have proven effective for this task [5], [16], [17]. Thus, we selected the newest versions that we could implement via Ollama, i.e., *Llama 3.1:8B* and *Mistral:7B*. Our LLMs selection ensures coverage of: (1) both proprietary and open-source LLMs, (2) diverse model families, and (3) LLMs most frequently used in literature for the specification task.

D. Experimental Configurations

In order to address the research questions, we define and implement six experimental configurations. For RQ1, we evaluate the selected LLMs across three prompting techniques: zero-shot, few-shot, and Retrieval-Augmented Generation (RAG). Thus, we cover three configurations. The first configuration *C1* prompts the LLMs directly without providing domain context or examples. It aims to assess the models' reasoning capabilities. In the second configuration *C2*, we provide examples of requirements and their specification to the LLM. This configuration aims to evaluate the necessity and effectiveness of few-shot examples in guiding the LLM. In the third configuration *C3*, we augment the prompt with domain context extracted from the ARINC-653 specification through RAG. This configuration aims to assess the impact of domain-specific knowledge on specification accuracy.

To address RQ2, we implement three additional configurations. In configuration *C4*, we use a domain model enhanced (DM-enhanced) RAG where we provide the LLM with context in the form of several domain model snippets. This configuration enables a direct comparison between traditional textual RAG (*C3*) and DM-enhanced RAG to determine the impact of model-based context on both LLM performance and execution time. In the final configurations *C5* and *C6*, we combine both few-shot and RAG, and few-shot and DM-enhanced RAG, respectively. These configurations evaluate whether providing both task-related context (examples) and domain-related context optimizes LLM effectiveness.

Overall, this study covers six experimental configurations evaluated across three LLMs and two datasets, while considering various hyperparameter variations for each configuration, resulting in a total suite of **138 experiments**.

E. Prompting Techniques

Aligned with with the experimental configurations in Section IV-D, we assess various prompting techniques for template-based requirement specification: zero-shot, few-shot, traditional RAG, and DM-enhanced RAG. For the four techniques, we use the base prompt presented in Figure 4. The difference lies in the context added for each technique. For instance, in both traditional and DM-enhanced RAG, we add a 'Domain context' section where the retrieved information is provided. Similarly, for few-shot, an 'Examples' section is added to inject the selected examples into the prompt.

Zero-Shot. As illustrated in Figure 4, the prompt is structured into eight sections. The first section prompts the LLM to adopt a persona of a requirement engineer expert in the ARINC-653 standard and defines the specification task. The second section *"Requirement to analyze"* includes the input requirement that LLM should specify according to the template. The third section *"Instructions"* provides a list of instructions and steps that guide the LLM through the specification. For instance, it instructs the LLM to first divide the input requirement into atomic requirements before applying the template.

The fourth section defines the format of the output. Specifically, the LLM is instructed to generate a rationale and a confidence score alongside the specified requirements for each input requirement. Prompting the LLM to provide explicit rationales enhances their reasoning capabilities and supports credibility [24]. Furthermore, providing confidence scores enhances the interpretability of the LLM's outputs [25].

The *"Definition of the template"* section provides a description of the template. Specifically, it lists and defines the types of conditions and actions supported by the operational behavior template. The subsequent sections then present the template itself (*"Textual Template"*) as well as the templates for conditions and actions (*"Templates for Conditions and Actions"*), as described in section III. Finally, the "Notations Clarification" section explains the notations used to document the templates, e.g., that "< >" denotes placeholders that must be filled. In the case of the few-shot and RAG experiments, an additional context section is added at the end of the prompt.

Traditional RAG. For RAG, we used Chapter 2 and Appendices E and H of the ARINC-653 specification as context [14]. These sections were selected as: (1) chapter 2 provides an extensive description of the ARINC-653 services, including the HM and IPC, (2) appendix E defines all ARINC specification types and functions, and (3) appendix H includes schema code detailing these types and their valid value ranges.

You are a requirement engineer expert in the ARINC-653 certification. You will be provided with a functional requirement which was extracted from the Health Monitor section of the ARINC-653 specification.

Given the following instructions, template definitions, and domain model information, your task is to analyze the requirement, divide it if necessary, and reformulate it according to the template.
 # Requirement to Analyze:

#Instructions:
- If the requirement includes multiple subjects, action verbs, or objects within the action section, divide it into multiple requirements that each covers one subject, one action verb, and one object. You must generate all the requirements.
- If an element of the template is missing from the requirement in order to match the template, identify the correct information from the domain model.
- If a missing element cannot be identified from the domain model nor from the domain context, leave it empty and explain why in the rationale.
- Make the requirement verifiable when possible. If a condition or an action can be specified by a parameter and its value, replace it by the parameter attribute to make the requirement more verifiable.
- You must not add a condition if it is not specified in the input requirement or if it does not provide new information.
- If a requirement is already atomic and matches the template, do not change the wording unless it contradicts the Domain Model.
- The output requirement(s) should be written exactly according to the template. Do not add or remove any element from the template.
- The output requirement(s) should be in active voice.
- The output requirement(s) should not describe a different action or be semantically different from the input requirement.
- Provide a confidence score of the reformulated requirement(s), from a score of 0 to 3, with 0 being not confident at all and 3 being the most confident.
- Provide a rationale explaining all the decisions made during the reformulation. Provide only one paragraph for the rationale.

#Output Format
The output MUST be structured using these exact labels, each on a new line:
Reformulated Requirement(s): [Requirement(s)]
Rationale: [EXPLANATION]
Confidence Score: [SCORE]

 # Definition of the template:
- The template is used to specify functional requirements, describing an action that the system or its components perform under specific conditions. It includes two blocks: one block for conditions and another block for an action. The template allows zero or multiple conditions but only one action within the requirement.
- The template supports five types of conditions: (1) continuous action-driven conditions: describe a continuous action that triggers another action, (2) Discontinuous action-Driven conditions: describe a discontinuous action that can trigger another action, (3) Event-driven conditions: describe a constraint that is triggered by an event occurring at a specific moment of time, (4) Value-related conditions: specify whether a parameter is equal, inferior, or superior to a certain value, and (5) Range-related conditions: specify whether a parameter is within or out of a certain range.
- The template supports two types of actions: (1) operational actions specify an action that a subject shall perform on an object, and (2) value-related actions specify that a subject shall set a parameter to a certain value.
- The structure of the template is described in the textual template below. The templates for all types of conditions and actions are described in the predefined templates below.

 # Textual Template:
- The reformulated requirement MUST be structured using this exact template, with each line of the template placed on a new line:
[when / while / if
<condition>
[AND <condition>]*
Then]
<operational behavior action>

 # Predefined Templates:
<Continuous action-driven condition> : While <subject> [<detail>] <action verb(-ing)> [<object>] [<detail>]*
<Discontinuous action-driven condition> : When <object> [<detail>] <action verb(-ed)> [[<adverb>] <object> [<detail>]]*
<Event-driven condition> : If <subject> [<detail>] <action verb> [[<adverb>] <object> [<detail>]]*
<Value-related condition> : If <parameter name> [of <entity> [<detail>]*] is equal/inferior/superior to <value>
<Range-related condition> : If <parameter name> [of <entity> [<detail>]*] is within / out of <value range>
<Operational action> : <subject> [<detail>]* shall <action verb> [[<adverb>] <object> [<detail>]]*
<Value-related action> : <subject> [<detail>]* shall set <parameter name> [of <entity> [<detail>]*] to <value>

<Subject> / <Object> : <range> <entity>
<Detail> : [<preposition>] [<range>] <entity>
<adverb> : for / as / from / to / by / upon / in
<value> : <parameter name> [of <entity> [<detail>]*] / <concrete value> / valid value / invalid value / infinite value
<range> : the / an / a / each / only one / all / one or more / at least one / other / another
<Preposition> : with / using / within / for / of / at / before / after / without / outside / in / on / via / until / to / other than / between / and
<Value range> : [<minimum value> , <maximum value>]

 # Notations clarification:
- < > denotes the variable parts that should be filled.
- [ ] denotes an optional block within the template.
- * denotes the possibility of having zero or many instances of a block.
- "/" denotes the options that can be used to fill the variable parts.

Fig. 4. Base Prompt for the experiments

To evaluate various RAG configurations, we analyzed two parameters: chunk size and the number of retrieved chunks. For chunk size, we compared two values: 500 and 1000 characters, with 200 character overlap, enabling the inclusion of sufficient context while evaluating the impact of varying sizes. For the number of chunks, we experimented with 2, 4, 6, 8, and 10 chunks, covering a broad range of context lengths.

Few-Shot. For the few-shot evaluation, we experimented with 2, 4, 6, 8, and 10 examples. Following prior work [26]–[28] examples were selected dynamically for each requirement from the remaining dataset based on semantic similarity. This approach was preferred over a train-test split due to the limited dataset size, which would otherwise reduce the test set and weaken the robustness of the evaluation.

DM-enhanced RAG. For domain-model-based RAG, we used a domain model comprising four sub-packages: HM errors, HM configuration tables, partition management, and intra-partition communication. Partition management was included because of its dependencies with the other sub-packages and the domain concepts it provides. The model was represented textually using PlantUML[1], a widely used and LLM-friendly modeling language [29], [30]. Following established chunking guidelines [31], each chunk contained a single class, association, or enumeration to preserve semantic completeness. Since the chunks are smaller and the same concept can be included in multiple chunks, we experimented with 5, 10, 15, and 20 chunks, as well as providing the entire domain model as context.

Listing 1 presents a snippet of the HM domain model in PlantUML. It illustrates the *Process level error* and *Error handler process* classes, the *Handles* association between them, and the *ErrorCodeType* as defined in Figure 3. During the chunking process, for instance, this snippet will be divided into four chunks, each covering a model entity: the *Process level error* class, the *Error handler process* class, the *Handles* association, and the *ErrorCodeType* enumeration.

[1]https://plantuml.com/en/

Listing 1. Snippet of the HM domain model in PlantUML.

```
@startuml
class "Process level error" {
    ErrorCode: ErrorCodeType
}
class "Error handler process"
"Error handler process" "0..1" --> "*" "Process
    level error": Handles
enum ErrorCodeType {
    DEADLINE_MISSED
    APPLICATION_ERROR
    NUMERIC_ERROR
    ILLEGAL_REQUEST
    STACK_OVERFLOW
    MEMORY_VIOLATION
    HARDWARE_FAULT
    POWER_FAIL
}
@enduml
```

F. Metrics

Template-based specification cannot be evaluated meaningfully using a single absolute metric. It requires multiple dimensions taking into consideration the model's ability to use domain information and adhere to the template. Thus, we decompose the evaluation into three stage-specific metrics which assess: (1) the correctness of the decomposition, (2) the correctness of the selected template, and (3) the correctness of the template instantiation, conditional on selecting the correct template. Accordingly, our evaluation includes metrics that cover three cognitive capabilities of LLMs that reflect the stages of the specification process: logical decomposition, structural reasoning, and semantic grounding. Each metric exclusively measures its corresponding capability without being penalized for incorrectly executed previous steps. Structural reasoning is evaluated only for requirements where decomposition was executed correctly and sematic grounding is measured strictly for requirements that demonstrated correct structural reasoning. In the following, we define each dimension and its metrics.

Logical decomposition. This dimension evaluates whether the requirement was correctly decomposed into atomic requirements. An atomic requirement describes a single capability of the system, describing the action with one subject, one action verb, one object, and optionally other details related to the object. To measure logical decomposition, we consider two metrics: Atomicity Precision (*AP*), which measures whether generated requirements are truly atomic, and Atomicity Recall (*AR*), which measures whether all expected atomic requirements were extracted.

$$AP = \frac{\text{Number of correctly generated atomic requirements}}{\text{Number of generated requirements}} \tag{1}$$

$$AR = \frac{\text{Number of correctly generated atomic requirements}}{\text{Number of expected atomic requirements}} \tag{2}$$

Structural reasoning. This dimension assesses whether the LLM selected the correct templates for conditions and actions. It is evaluated through one metric: Template Accuracy (*TA*), which measures the percentage of templates that were correctly selected by the model relative to the total templates within the correctly decomposed requirements.

$$TA = \frac{\text{Number of correctly selected templates}}{\text{Number of templates in correctly decomposed requirements}} \tag{3}$$

Semantic grounding. This dimension evaluates the correctness of the information used to fill the placeholder slots within the condition and action templates. It is assessed through one metric, Slot Filling Precision (*SFP*), which measures the percentage of slots that were correctly filled out of the total slots of correctly selected templates.

$$SFP = \frac{\text{Number of correctly filled slots}}{\text{Number of slots in correctly selected templates}} \tag{4}$$

To derive a single score that reflects the overall correctness of the specification across all dimensions, we compute a weighted combination of the correctness metrics. The Overall Correctness (*OC*) metric aggregates the metrics based on their relative significance to the specification process and the research objectives. Since the evaluation focuses exclusively on correctness, *OC* is computed using only the correctness metrics, namely (TA), (SFP), (AP). There are no predefined weights for these metrics. The weights should be determined based on the evaluator's objectives. In our case, since our main goal is to evaluate the LLM's ability to adhere to the template structure and accurately leverage domain information, we allocate a higher weight (0.4) to Template Accuracy (TA) and Semantic Grounding (SFP) compared to Logical Decomposition (AP 0.2).

$$OC = \alpha \cdot AP + \beta \cdot TA + \gamma \cdot SFP \tag{5}$$

G. Metrics Calculation

To ensure objective and consistent evaluation, we developed scripts to calculate all metrics, using the regex library. The metrics scripts were designed to accommodate the variations of LLM outputs to the best of ability. For Atomic Precision (*AP*) and Atomic Recall (*AR*), a requirement is deemed "correctly generated" if: (1) the input requirement is decomposed into the correct number of requirements, and (2) each requirement contains only one subject, one action verb, and one object. For Template Accuracy (*TA*), a condition or an action is "correctly selected" if: (1) the type matches the ground truth (GT), and (2) it adheres to the template structure.

For SFP on the other hand, a slot is considered "correctly filled" only if the content matches the one expected in the GT. Since the templates include various slots with some being optional and can occur multiple times with various structures (e.g., "detail" slot), we only included mandatory slots in the analysis, making the analysis focused and flexible. We only assess the "subject", "action verb, and "object" slots for continuous, discontinuous, and event driven conditions, as well

TABLE II
EXPERIMENTAL RESULTS OF ZERO-SHOT LLMS

LLM	HM Dataset					IPC Dataset				
	AP	*AR*	*TA*	*SFP*	*OC*	*AP*	*AR*	*TA*	*SFP*	*OC*
GPT-5.4	**0.76**	**0.64**	**0.80**	**0.56**	**0.70**	**0.80**	**0.82**	**0.75**	**0.44**	**0.64**
Llama	0.45	0.42	0.61	0.37	0.48	0.42	0.51	0.56	0.38	0.46
Mistral	0.57	0.38	0.63	0.40	0.53	0.66	0.57	0.67	0.30	0.52

as operation actions. While we assess the "parameter name" and "parameter value" slots for value and range related conditions, and the "subject", "parameter name", and "parameter value" slots for value conditions. To ensure flexibility, the SFP script is case-insensitive, ignores verb tense (e.g., "is created" and "creates" are treated as equivalent), and treats variations in terminology (e.g., "OS," "O/S," and "operating system"). Furthermore, the script ignores articles (e.g., "the," "a") to focus on core concepts. However, despite this flexibility, the script remains strict specifically regarding subject and object content. For example, if the GT requires "detected error" but the model generates "error", the output is marked incorrect.

V. EXPERIMENTAL RESULTS

To evaluate LLM performance across configurations, we calculated the metrics described in the previous section. For readability, tables displaying results highlight the highest score per metric in bold, while the overall best-performing configuration across all experiments is highlighted in red.

A. RQ.1: Comparative analysis of SOTA LLMs and Prompt Strategies for Template-Based Requirements Specification

To answer this RQ, we evaluated the performance of the three LLMs for zero-shot, traditional RAG, and few-shot across the two datasets.

Zero-shot. Table II presents the results for the zero-shot experiments. *GPT* consistently outperforms the open-source LLMs across all metrics, achieving 0.7 and 0.64 in OC for HM and IPC, respectively. For open-source models, *Mistral* outperforms *Llama* with regards to OC and most metrics.

Traditional RAG. Table III presents the results of the traditional RAG experiments. *GPT* consistently achieves the highest performance across all configurations. Regarding chunk size, 500 and 1000 tokens yield comparable results. However, chunk size of 500 is slightly more effective for the HM dataset, reaching an overall correctness score above 0.70 in all three experiments. Overall, variations in chunk size do not provide a substantial difference. On the other hand, the number of chunks has a more evident impact, with *OC* consistently improving as the number of chunks increases, with configurations of 8 and 10 chunks yielding the highest scores. However, even the best-performing RAG configurations do not show considerable improvement over simple zero-shot prompting.

For open-source models, *Mistral* outperforms *Llama* in most configurations. *Llama* achieved its peak performance with 8 chunks of 500 characters (OC of 0.52 and 0.49 for the HM and IPC), while *Mistral* peaked with 6 chunks of 500 characters (OC of 0.53 and 0.52). Additionally, while *Llama* occasionally improved upon its zero-shot performance, *Mistral* failed to reach its zero-shot baseline across all RAG configurations. For

TABLE III
EXPERIMENTAL RESULTS OF TRADITIONAL RAG

LLM	Chunk Size	No. Chunks	HM Dataset					IPC Dataset				
			AP	*AR*	*TA*	*SFP*	*OC*	*AP*	*AR*	*TA*	*SFP*	*OC*
GPT-5.4	**500c**	**2 Chunks**	0.65	0.53	0.80	0.56	0.67	0.72	0.75	0.68	0.53	0.63
		4 Chunks	0.66	0.51	0.80	0.62	0.70	0.78	0.78	0.73	0.47	0.64
		6 Chunks	0.80	0.66	0.64	0.60	0.66	0.78	0.78	0.71	0.49	0.64
		8 Chunks	0.84	0.58	0.70	0.67	**0.72**	0.80	0.77	**0.75**	0.48	0.65
		10 Chunks	**0.87**	**0.77**	**0.82**	0.52	0.71	0.79	0.80	0.72	0.53	**0.66**
	1000c	**2 Chunks**	0.77	0.62	0.72	0.58	0.67	0.85	**0.83**	0.69	0.47	0.63
		4 Chunks	0.63	0.51	0.80	0.48	0.64	0.82	0.81	0.68	0.44	0.61
		6 Chunks	0.66	0.51	0.72	0.52	0.63	0.76	0.76	0.70	0.48	0.62
		8 Chunks	0.62	0.47	0.75	**0.72**	0.71	**0.86**	0.81	0.71	0.51	**0.66**
		10 Chunks	0.79	0.62	0.78	0.56	0.69	0.82	0.81	0.66	**0.58**	**0.66**
Llama	**500c**	**2 Chunks**	0.54	0.40	0.56	0.48	0.52	0.33	0.40	0.70	0.24	0.44
		4 Chunks	0.39	0.34	0.68	0.42	0.52	0.51	0.56	0.65	0.27	0.47
		6 Chunks	0.49	0.40	0.57	0.34	0.46	0.50	0.56	0.66	0.29	0.48
		8 Chunks	0.34	0.28	0.67	0.46	0.52	0.47	0.55	0.62	0.38	0.49
		10 Chunks	0.33	0.30	0.48	0.39	0.41	0.41	0.48	0.62	0.33	0.46
	1000c	**2 Chunks**	0.42	0.34	0.56	0.32	0.44	0.48	0.46	0.57	0.27	0.43
		4 Chunks	0.43	0.40	0.54	0.34	0.44	0.49	0.53	0.66	0.25	0.46
		6 Chunks	0.37	0.30	0.48	0.35	0.41	0.45	0.48	0.64	0.26	0.45
		8 Chunks	0.29	0.28	0.55	0.30	0.40	0.41	0.45	0.56	0.16	0.37
		10 Chunks	0.40	0.32	0.65	0.29	0.46	0.38	0.42	0.64	0.23	0.42
Mistral	**500c**	**2 Chunks**	0.38	0.30	0.66	0.25	0.44	0.54	0.56	0.69	0.29	0.50
		4 Chunks	0.44	0.34	0.68	0.27	0.47	0.47	0.41	0.67	0.28	0.47
		6 Chunks	0.63	0.49	0.66	0.34	0.53	0.52	0.51	0.75	0.30	0.52
		8 Chunks	0.52	0.42	0.72	0.31	0.52	0.55	0.53	0.65	0.25	0.47
		10 Chunks	0.50	0.40	0.65	0.28	0.47	0.50	0.43	0.73	0.22	0.48
	1000c	**2 Chunks**	0.45	0.32	0.72	0.24	0.47	0.52	0.54	0.73	0.27	0.50
		4 Chunks	0.46	0.36	0.64	0.29	0.46	0.54	0.48	0.72	0.20	0.48
		6 Chunks	0.34	0.28	0.62	0.30	0.44	0.44	0.45	0.63	0.21	0.42
		8 Chunks	0.37	0.36	0.61	0.22	0.41	0.43	0.43	0.63	0.24	0.43
		10 Chunks	0.37	0.34	0.69	0.23	0.44	0.36	0.31	0.67	0.19	0.42

chunk size, 500 characters consistently yields higher *OC* for both models. However, unlike *GPT*, *Llama* and *Mistral* yielded fluctuating results regardless of the number of chunks.

Regarding atomicity (*AP* and *AR*), *GPT* clearly outperforms *Llama* and *Mistral*, often achieving nearly twice their scores. For HM, precision is consistently higher than recall, indicating that GPT is conservative: it identifies decompositions accurately but misses some requirements that should be decomposed. In contrast, for IPC, recall is generally higher than precision, meaning that *GPT* identifies more decompositions at the cost of some incorrect ones. This difference might stem from variations in dataset content and the proportion of decomposed requirements (37% for HM vs. 29% for IPC).

With regards to *TA*, while *GPT* consistently provides higher scores, *Mistral* provides comparable results in most experiments. However, slot filling (*SFP*) remains a challenge for all models. While *GPT* achieves the highest scores, they remain modest, ranging between 0.50 and 0.65. This is largely due to *GPT* often reverting to passive voice despite prompt instructions, failing to match the GT. This performance is also due to the inherent rigidity of the evaluation script. *Llama* and *Mistral* demonstrate even lower *SFP*, with scores falling to 0.29 and 0.16 for *Llama*, and 0.23 and 0.19 for *Mistral*, for HM and IPC, respectively.

Overall, Providing domain context via traditional RAG does not seem to offer a substantial advantage over zero-shot prompting.

Few-shot. The results for the few-shot experiments are presented in Table IV. Consistent with the previous techniques, *GPT* outperforms *Llama* and *Mistral* across all dimensions. The highest scores were achieved using 8 or 10 examples, with 8 examples providing the overall highest scores for *TA*, *SFP*, and *OC* across all experiments in this study. This configuration achieved an *OC* of 0.75 and 0.83 for HM and IPC, respectively. The improvement was much more evident for *TA*, as few-shot reached 0.78 compared to 0.58 in traditional RAG for IPC.

We initially expected few-shot prompting to primarily improve atomicity (*AP* and *AR*) by illustrating decomposition,

TABLE IV
EXPERIMENTAL RESULTS OF FEW-SHOT LLMS

LLM	No. Examples	HM Dataset					IPC Dataset				
		AP	*AR*	*TA*	*SFP*	*OC*	*AP*	*AR*	*TA*	*SFP*	*OC*
GPT-5.4	**2 Examples**	0.66	0.58	0.79	0.67	0.72	**0.83**	0.84	0.82	0.63	0.75
	4 Examples	0.64	0.57	0.79	0.63	0.70	**0.83**	0.84	0.86	0.72	0.80
	6 Examples	0.67	0.60	0.77	0.69	0.72	0.80	0.81	0.84	0.73	0.79
	8 Examples	**0.71**	**0.64**	**0.85**	0.67	**0.75**	0.82	0.83	**0.89**	**0.78**	**0.83**
	10 Examples	0.62	0.58	0.75	**0.77**	0.73	0.82	**0.85**	0.87	0.74	0.81
Llama	**2 Examples**	0.62	0.58	0.73	0.31	0.54	0.30	0.35	0.71	0.54	0.56
	4 Examples	0.47	0.40	0.67	0.41	0.53	0.33	0.35	0.76	0.57	0.60
	6 Examples	0.31	0.30	0.65	0.53	0.53	0.40	0.46	0.69	0.57	0.58
	8 Examples	0.29	0.28	0.56	0.57	0.51	0.31	0.37	0.71	0.55	0.57
	10 Examples	0.35	0.32	0.58	0.18	0.37	0.32	0.35	0.67	0.60	0.57
Mistral	**2 Examples**	0.47	0.38	0.72	0.45	0.56	0.37	0.34	0.76	0.56	0.60
	4 Examples	0.31	0.23	0.75	0.51	0.57	0.26	0.24	0.69	0.34	0.46
	6 Examples	0.42	0.28	0.75	0.39	0.54	0.27	0.24	0.79	0.45	0.55
	8 Examples	0.41	0.30	0.76	0.55	0.61	0.35	0.32	0.73	0.54	0.58
	10 Examples	0.49	0.34	0.69	0.55	0.59	0.23	0.19	0.76	0.58	0.58

TABLE V
EXPERIMENTAL RESULTS OF ZERO-SHOT WITH DOMAIN MODEL RAG

LLM	No. Chunks	HM Dataset					IPC Dataset				
		AP	*AR*	*TA*	*SFP*	*OC*	*AP*	*AR*	*TA*	*SFP*	*OC*
GPT-5.4	**5 Chunks**	0.60	0.51	0.78	0.53	0.64	0.82	0.83	**0.75**	0.48	0.66
	10 Chunks	0.69	0.55	0.73	**0.62**	0.68	0.76	0.77	0.65	0.49	0.61
	15 Chunks	0.66	0.55	0.78	0.59	0.68	0.80	0.80	0.72	0.46	0.63
	20 Chunks	0.68	0.57	0.71	0.59	0.66	0.79	0.82	0.73	0.50	0.65
	Entire Model	**0.92**	**0.85**	**0.85**	0.54	**0.74**	**0.84**	**0.83**	0.70	**0.55**	**0.67**
Llama	**5 Chunks**	0.64	0.53	0.59	0.34	0.50	0.50	0.56	0.58	0.22	0.42
	10 Chunks	0.43	0.34	0.70	0.36	0.51	0.55	0.62	0.55	0.25	0.43
	15 Chunks	0.45	0.38	0.62	0.40	0.50	0.53	0.56	0.60	0.35	0.49
	20 Chunks	0.57	0.45	0.57	0.32	0.47	0.41	0.47	0.47	0.21	0.35
	Entire Model	0.29	0.28	0.62	0.33	0.44	0.36	0.39	0.58	0.17	0.37
Mistral	**5 Chunks**	0.59	0.42	0.71	0.41	0.57	0.61	0.59	0.62	0.23	0.46
	10 Chunks	0.36	0.25	0.68	0.26	0.45	0.63	0.68	0.64	0.23	0.47
	15 Chunks	0.54	0.40	0.67	0.25	0.48	0.49	0.55	0.61	0.20	0.42
	20 Chunks	0.39	0.30	0.53	0.11	0.33	0.59	0.66	0.59	0.21	0.44
	Entire Model	0.00	0.00	0.00	0.00	0.00	0.06	0.03	0.33	0.00	0.14

while domain context (RAG) would mainly enhance *TA* and *SFP* by providing domain-specific knowledge. However, these results, supported by the DM-RAG findings discussed in the following section, suggest otherwise. This demonstrates that LLMs rely more heavily on concrete task examples than contextual domain knowledge, as they provide both task-related guidance and the domain context that guide the specification. Compared to zero-shot, few-shot prompting provided a 5% increase in *OC* for HM and a notable 19% increase for IPC.

For open-source models, few-shot also outperformed traditional RAG, achieving the highest *OC* scores in three out of four cases with a lower number of examples. *Llama* reached peak performance (OC of 0.54 and 0.60 for HM and IPC) with 2 and 4 examples, while *Mistral* peaked with 2 examples for IPC. This confirms that both proprietary and open-source LLMs prioritize illustrative examples over domain context for template-based specification. The difference in the optimal number of examples, where *GPT* favored a higher count and open-source models favored fewer might originate from the limited capacity of smaller models to process long context prompts. Finally, when comparing the two open-source models, *Mistral* outperformed *Llama* in 8 out of 10 experiments.

Overall, few-shot prompting provides a more significant improvement over zero-shot and traditional RAG, highlighting the importance of illustrative examples for this task.

B. RQ.2: Comparative Study of Domain Models and Unstructured Text as RAG Context

To answer this RQ, we compare the results of traditional RAG with those of DM-enhanced RAG regarding both evaluation metrics and execution time. Furthermore, we analyze and compare their performance when combined with few-shot.

Traditional RAG vs. DM-enhanced RAG. Table V presents the results of DM-enhanced RAG. Consistent with prior observations, *GPT* outperforms *Llama* and *Mistral*. GPT achieved its best overall results when provided with the entire domain model, reaching an *OC* of 0.74 for HM and 0.67 for IPC. This suggests that the model is sufficiently compact to provide comprehensive context without overwhelming the LLM. However, the improvement over traditional RAG was marginal (2.8% for HM and 1.5% for IPC), offering limited justification for the effort required to create and maintain domain models. Although DM-enhanced RAG achieved high *AP*, *AR*, and *TA* scores—yielding the best results for the HM dataset—its *SFP* was lower than that of traditional RAG. *GPT* often identified the correct domain concepts but used passive constructions or terminology that differed from the GT, reducing *SFP*. Overall, few-shot prompting remained the best-performing approach, particularly for IPC.

For the open-source models, DM-enhanced RAG yielded performance comparable to traditional RAG, with no clear advantage for either technique. For *Llama*, peak *OC* scores were 0.51 and 0.49 (compared to 0.52 and 0.49 in traditional RAG), while *Mistral* reached 0.57 and 0.47 (compared to 0.53 and 0.52) for HM and IPC, respectively. However, a decline was particularly evident in the SFP metric. Output analysis showed that *Llama* and *Mistral* often struggled to interpret domain-model information, frequently hallucinating details that altered the original requirement. For instance, the requirement "*These mechanisms (buffers and blackboards) support the communication of messages between multiple source and destination processes*" was reformulated by *Llama* as "*When a process uses an intra-partition communication mechanism, then it shall set the parameter MAX_NB_MESSAGE of the Buffer with BUFFER_NAME <BUFFER_NAME> to the valid value*". Unlike *GPT*, providing the full domain model degraded the performance of the open-source models. *Mistral* achieved *OC* scores as low as 0.00 and 0.14, largely because it frequently generated non-compliant JSON outputs which were marked as incorrect. These findings suggest that open-source models have greater difficulty leveraging domain-model context. Overall, few-shot prompting remained the most effective approach for both proprietary and open-source models.

Beyond performance, we compared the execution time of traditional RAG and DM-enhanced RAG. Table VI presents the average execution time across all configurations. DM-enhanced RAG consistently reduces execution time compared to traditional RAG, with savings ranging from 22 to 39 seconds for HM and 3 to 152 seconds for IPC. The difference was the most evident for *Llama*. These results are consistent with our expectations, since traditional RAG requires scanning through long unstructured text while DM-enhanced RAG leverages a structured and compact model. We anticipate that this gain would be more substantial for larger datasets.

Overall, while DM-enhanced RAG reduces execution time compared to traditional RAG, the performance gains remain marginal. Few-shot prompting remains the most effective strategy for template-based requirements specification.

TABLE VI
AVERAGE EXECUTION TIME OF TRADITIONAL RAG AND DOMAIN MODEL ENHANCED RAG

LLM	HM Dataset		IPC Dataset	
	RAG	DM-RAG	RAG	DM-RAG
GPT-5.4	195 s	177 s	470 s	441 s
Llama	328 s	289 s	545 s	393 s
Mistral	257 s	233 s	355 s	352 s

TABLE VII
EXPERIMENTAL RESULTS OF FEW-SHOT WITH DOMAIN MODEL AND TRADITIONAL RAG

RAG	LLM	HM Dataset					IPC Dataset				
		AP	*AR*	*TA*	*SFP*	*OC*	*AP*	*AR*	*TA*	*SFP*	*OC*
Standard	**GPT-5.4**	**0.83**	**0.85**	0.72	**0.60**	0.69	0.81	0.83	**0.90**	0.70	0.80
	Llama	0.21	0.17	0.64	0.24	0.39	0.26	0.31	0.79	0.60	0.61
	Mistral	0.47	0.34	0.78	0.33	0.54	0.26	0.24	0.88	0.36	0.55
DM	**GPT-5.4**	0.80	**0.85**	**0.85**	0.52	**0.71**	**0.84**	**0.86**	0.89	**0.72**	**0.81**
	Llama	0.42	0.40	0.74	0.53	0.59	0.32	0.32	0.74	0.56	0.58
	Mistral	0.49	0.34	0.73	0.51	0.59	0.24	0.22	0.79	0.50	0.56

Combining RAG with Fewshot. While few-shot yielded the highest performance compared to RAG techniques, it remains unclear whether combining few-shot with RAG would provide further benefits or if the inclusion of few-shot would cause one RAG technique to outperform the other. To investigate this, we compare the combination of few-shot with both traditional and DM-enhanced RAG. For these experiments, we selected the optimal configurations for few-shot, traditional RAG, and DM-enhanced RAG for each model based on the average *OC* scores between the two datasets from our previous experiments. The selected configurations were: (1) *GPT*: 8 examples, 8 chunks of 500 characters, and the entire domain model; (2) *Llama*: 4 examples, 8 chunks of 500 characters, and 15 domain model chunks; and (3) *Mistral*: 8 examples, 6 chunks of 500 characters, and 5 domain model chunks.

Table VII provides the results of the experiments. The results exhibit patterns consistent with our previous findings. While incorporating a DM with few-shot yielded marginally better results than traditional RAG combined with few-shot (2.9% and 1.25% improvements for HM and IPC), this gain remains insignificant. Furthermore, simple few-shot consistently outperformed few-shot combined with either RAG technique. This drop may be attributed to the excessive content added to the prompt, which might have introduced noise and confused the LLM. For *Llama* and *Mistral*, the results of these combinations are comparable to simple few-shot prompting.

Overall, combining few-shot with RAG does not provide any benefits. Simple few-shot suffices, yielding the highest results across all conducted experiments.

VI. DISCUSSION

a) ***RQ1.****:* The experimental results demonstrate the superiority of *GPT* over *Llama* and *Mistral*, with *Mistral* outperforming *Llama*. Zero-shot yielded decent results, particularly for the HM dataset, indicating that the instructions and template structure provided in the prompt allow for relatively accurate requirement decomposition and template selection even in the absence of external context or examples.

Regarding traditional RAG, a chunk size of 500 characters consistently yielded higher performance across all models, by minimizing the noise often present in 1000 character chunks. All models achieved their highest *OC* scores when provided with 8 chunks. While OC for *Llama* and *Mistral* fluctuated with the number of chunks, *GPT-5.4* exhibited a linear performance increase, reflecting its superior capacity for information analysis and utilization. These findings were further confirmed by the DM-enhanced RAG experiments, as *GPT* performed optimally when augmented with the entire model, whereas open-source models suffered, which can be attributed to their limited capacity for processing large input.

b) ***RQ2.****:* Ultimately, leveraging a domain model instead of textual context did not yield the performance improvements we expected. This may be attributed to the inherent complexity of domain models. While they offer structured information, they lack the descriptive details present in textual content. Moreover, the performance of DM-enhanced RAG is highly sensitive to the quality and completeness of the input DM. Furthermore, given the significant time, effort, and domain expertise required to build high-quality DMs, this technique is less cost-effective than utilizing standard textual context.

Few-shot prompting proved to be the most effective technique across all evaluation dimensions. This indicates that LLMs perform best when provided with illustrative examples as they illustrate the specification steps (i.e., decomposition, template selection, and slot filling). According to our experiments, the optimal configuration for template-based requirements specification, is ***GPT-5.4*** **using 8 few-shot examples**. Furthermore, our experiments showed that combining few-shot with RAG techniques provides no benefits, as the long context appears to introduce noise that confuses the models.

VII. THREATS TO VALIDITY

Some threats might impact the internal, and external validity of our evaluation.

For internal validity, the GT and the domain models were manually constructed by one of the authors which might introduce bias. To mitigate this, both the specified requirements and domain models were validated by an RE expert and our industrial partner's team. Another threat is that the evaluation of the metrics was performed by the authors. We addressed this threat by implementing a script that strictly adheres to the guidelines described in this paper, thereby eliminating human subjectivity and ensuring homogeneous results across all experiments. However, the scripts can be strict. This is an inherent limitation of automated text analysis script. Another threat relates to the variability in how requirements can be specified. While this is particularly evident for non-SCS requirements, SCS specifications rely on a more constrained vocabulary, which helps mitigate this threat. Finally, LLMs can generate varying outputs due to their nondeterministic nature. To mitigate this threat, we set the model temperature to 0.2 across all experiments, which largely reduces the randomness.

Regarding external validity, our evaluation is limited to a single template and our datasets are limited to requirements extracted from one certification standard: the ARINC-653 from the avionic domain. However, the template is general

and applicable across different domains,as demonstrated in our previous work [4], and both datasets include requirements with varying structures and content. We plan to extend our evaluation to additional domains as part of future work.

VIII. Conclusion

This paper presents an empirical evaluation of LLM-enabled, template-based requirements specification, comparing various prompting strategies and context-enrichment techniques. The study targets the operational behavior template for functional requirements. We assessed the performance of zero-shot, traditional RAG, and few-shot prompting, and compared traditional RAG against DM-enhanced RAG. The evaluation comprised 138 experiments across three LLMs using two datasets from the ARINC-653 avionics standard. All experiments were assessed for logical decomposition, structural reasoning, and semantic grounding.

The results indicate that *GPT-5.4* outperformed *Mistral* and *Llama*, with *Mistral* showing better overall performance than *Llama*. While leveraging domain models rather than unstructured textual context reduced execution time, it yielded comparable overall correctness, suggesting that the overhead of creating and maintaining domain models may not be justified for the task. Few-shot prompting achieved the highest performance, reaching an overall correctness score of 0.83 and 0.75 for the two datasets when using *GPT-5.4* with an optimal configuration of eight examples. Furthermore, combining few-shot with RAG proved less effective than few-shot alone, demonstrating that illustrative examples are sufficient.

As future work, we plan to compare the cost of LLMs (e.g., in terms of token usage) against manual effort to assess their value in industrial settings. Furthermore, we aim to explore the use of a multi-agent architecture to support the various specification steps (i.e., decomposition, template selection, and slot filling). We will also extend the study to include other safety-critical domains (e.g., automotive), and analyze the rationale and confidence scores generated by the LLMs.

References

[1] H. Hofmann and F. Lehner, "Requirements engineering as a success factor in software projects," *IEEE Software*, vol. 18, 2001.

[2] P. Pitchford. (2024, April) Better requirements analysis eliminates design defects. Accessed: 2025-08-07. [Online]. Available: https://www.eetimes.com/better-requirements-analysis-eliminates-design-defects/

[3] T. Kuhn, "A survey and classification of controlled natural languages," *Computational Linguistics*, 2014.

[4] I. Darif, G. El Boussaidi, and S. Kpodjedo, "Requirements specification using templates: a model-driven approach," *Softw. Syst. Model.*, 2025.

[5] M. A. Zadenoori, J. Dąbrowski, W. Alhoshan, L. Zhao, and A. Ferrari, "Large language models (llms) for requirements engineering (re): A systematic literature review," 2025. [Online]. Available: https://arxiv.org/abs/2509.11446

[6] T. Wu, L. Luo, Y.-F. Li, S. Pan, T.-T. Vu, and G. Haffari, "Continual learning for large language models: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2402.01364

[7] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models," *ACM Trans. Intell. Syst. Technol.*, 2025.

[8] I. Darif, G. El-Boussaidi, S. Kpodjedo, P. Padmanabhan, and A. Paz, "On the generation of input space model for model-driven requirements-based testing," in *International Conference on Model-Driven Engineering and Software Development*, 2025.

[9] I. Darif, G. El Boussaidi, S. Kpodjedo, and C. Politowski, "Controlled natural language for requirements specification: A systematic literature review," *ACM Comput. Surv.*, vol. 58, no. 7, 2026.

[10] A. Katis, A. Mavridou, D. Giannakopoulou, T. Pressburger, and J. Schumann, "Capture, analyze, diagnose: Realizability checking of requirements in fret," in *Computer Aided Verification*, S. Shoham and Y. Vizel, Eds. Springer International Publishing, 2022.

[11] L. V. Barcelos and R. D. Penteado, "Elaboration of software requirements documents by means of patterns instantiation," *Journal of Software Engineering Research and Development*, vol. 5, 2017.

[12] C. Antoniou and N. Bassiliades, "A tool for requirements engineering using ontologies and boilerplates," *Automated Software Engineering*, vol. 31, no. 1, p. 5, 2023.

[13] I. Darif, C. Politowski, G. El Boussaidi, I. Benzarti, and S. Kpodjedo, "A model-driven and template-based approach for requirements specification," in *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2023.

[14] SAE, "ARINC Specification653P1-4. Avionics application software standard interface," p. 285, 2015.

[15] A. Hemmat, M. Sharbaf, S. Kolahdouz-Rahimi, K. Lano, and S. Y. Tehrani, "Research directions for using llm in software requirement engineering: a systematic review," *Frontiers Comput. Sci.*, vol. 7, 2025.

[16] M. Krishna, B. Gaur, A. Verma, and P. Jalote, "Using llms in software requirements specifications: An empirical evaluation," in *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, 2024.

[17] L. Pasquale, A. Ragone, E. Piemontese, and A. A. Darban, "Exploring the use of llms for requirements specification in an it consulting company," in *International Requirements Engineering Conference*, 2025.

[18] X. Lian, J. Ma, H. Lv, and L. Zhang, "Reqcompletion: Domain-enhanced automatic completion for software requirements," in *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, 2024.

[19] P. Kogler, A. Falkner, and S. Sperl, "Reliable generation of formal specifications using large language models," in *Software Engineering 2024 (SE 2024) - Companion*. Gesellschaft für Informatik e.V., 2024.

[20] M. R. Hasan, J. Li, I. Ahmed, and H. Bagheri, " Automated Repair of Alloy Specifications in the Era of Large Language Models ," *IEEE Transactions on Software Engineering*, no. 01, 2026.

[21] M. Alhanahnah, M. Rashedul Hasan, L. Xu, and H. Bagheri, "An empirical evaluation of pre-trained large language models for repairing declarative formal specifications," *Empirical Softw. Engg.*, 2025.

[22] I. Darif, C. Politowski, G. E. Boussaidi, and S. Kpodjedo, "A domain specific language for the arinc 653 specification," in *International Symposium on Software Reliability Engineering Workshops*, 2022.

[23] I. Darif, "Requirements specification using templates : A model-driven approach," 2024. [Online]. Available: https://doi.org/10.5281/zenodo.11176929

[24] T. Xu, S. Wu, S. Diao, X. Liu, X. Wang, Y. Chen, and J. Gao, "Sayself: Teaching llms to express confidence with self-reflective rationales," 2024. [Online]. Available: https://arxiv.org/abs/2405.20974

[25] G. Detommaso, M. Bertran, R. Fogliato, and A. Roth, "Multicalibration for confidence scoring in llms," 2024. [Online]. Available: https://arxiv.org/abs/2404.04689

[26] X. Li, K. Lv, H. Yan, T. Lin, W. Zhu, Y. Ni, G. Xie, X. Wang, and X. Qiu, "Unified demonstration retriever for in-context learning," 2023. [Online]. Available: https://arxiv.org/abs/2305.04320

[27] S. Wu, Y. Xiong, Y. Cui, H. Wu, C. Chen, Y. Yuan, L. Huang, X. Liu, T.-W. Kuo, N. Guan, and C. J. Xue, "Retrieval-augmented generation for natural language processing: A survey," 2025. [Online]. Available: https://arxiv.org/abs/2407.13193

[28] F. Niu, R. Pan, L. C. Briand, and H. Hu, "Tvr: Automotive system requirement traceability validation and recovery through retrieval-augmented generation," 2026. [Online]. Available: https://arxiv.org/abs/2504.15427

[29] P. F. Hans-Georg Fill, "Conceptual modeling and large language models: Impressions from first experiments with chatgpt," Enterprise Modelling and Information Systems Architectures (EMISAJ) – International Journal of Conceptual Modeling: Vol. 15, Nr. 3, Berlin, pp. 1–36, 2023.

[30] A. Ferrari, S. Abualhaija, and C. Arora, "Model generation with llms: From requirements to uml sequence diagrams," in *IEEE 32nd International Requirements Engineering Conference Workshops (REW)*, 2024.

[31] K. C. Li, V. Zolfaghari, N. Petrovic, F. Pan, and A. Knoll, "Optimizing retrieval augmented generation for object constraint language," 2025. [Online]. Available: https://arxiv.org/abs/2505.13129