



Can AI Help with the Formalization of Railway Cybersecurity Requirements?

Maurice H. ter Beek¹, Alessandro Fantechi^{1,2}, Stefania Gnesi¹,
Gabriele Lenzini³, and Marinella Petrocchi^{4,5}

¹ CNR-ISTI, Pisa, Italy

{maurice.terbeek,stefania.gnesi}@isti.cnr.it,
alessandro.fantechi@unifi.it

² DINFO, University of Florence, Florence, Italy

³ University of Luxembourg - SnT, Luxembourg, Luxembourg
gabriele.lenzini@uni.lu

⁴ CNR-IIT, Pisa, Italy

marinella.petrocchi@iit.cnr.it

⁵ IMT Scuola Alti Studi Lucca, Lucca, Italy

Abstract. Driven by and dependent on ICT, like almost everything today, railway transportation has become a critical infrastructure and, as such, is exposed to threats against communication of on-board and wayside components. The shift to cybersecurity brings up the need to comply with new security requirements, and once more security software engineers are confronted with a well-known problem: how to express informal requirements into unambiguous formal expressions that can be translated into enforceable policies or be used to verify the security of a system design. We have experience in translating natural language requirements from standards, regulations, and guidelines into Controlled Natural Language for Data Sharing Agreements (CNL4DSA), a formalism that serves the purpose of bridging natural and formal expressions. The translation of requirements is challenging, calling for a rigorous process of coding agreement between researchers. Following the trend of the time, in this paper, we question whether AI and, in particular, the novel Generative Language Models, can help us with this translation exercise. Previous work shows that AI can help in writing security code, although not always producing secure code; less studied is the quality of generative AI's working with controlled natural languages in writing requirements for security compliance. Can AI be a valuable tool or companion in this endeavour too? To answer this question, we engage ChatGPT and Microsoft 365 Copilot with the same challenges that we faced when translating cybersecurity requirements for railway systems into CNL4DSA. Comparing our results from some time ago with those of the machine, we found surprising insights, showing the high potentiality of using AI in requirements engineering.

Keywords: Controlled Natural Language · AI · ChatGPT · Copilot · Requirements Engineering · Moving Block Railway Signalling

1 Introduction

The railway industry has always been attentive to *safety* [9,11]. Ensuring safety does not only mean caring of customer welfare but also being careful in adopting potentially vulnerable modern technologies such as computer-based and communication-based signalling: therefore it should not surprise that even in the most advanced railway systems, critical decisions are still taken by centralized units (*e.g.*, the computation of “movement authorities” in Radio Block Centers, cf. Sect. 2). However, the modern railway is pressed to be more distributed: its network has become transnational with a larger capacity and higher availability demand and it is subject to stricter international resilience requirements. The vision for a more decentralized, distributed, and highly resilient railway system—*i.e.*, the vision of the railway system as a Collective Adaptive System (CAS)—is already latent in the goals of Europe’s Rail Joint Undertaking (EU-Rail).¹ It envisions, by the 2030s, a single European railway area, that could unify and overcome the different national systems.

This work focuses on a concern already present today, a first step on the path to realizing such vision: with the adoption of information and communication technologies, today’s railway industry should not only be safe, but also be resilient against cyberattacks [26,27]. Safety gets intertwined with *security*.

To realize appropriate domain-specific security defences and countermeasures, engineers have to identify, read, understand, and interpret relevant documents, regulations, and provisions.² Eventually, they have to elicit clearly defined requirements and implement them. These are renowned and challenging tasks. Requirements are often written in natural language, and their correct interpretation and implementation into systems, for instance, as policies in access control mechanisms, is threatened by vagueness and ambiguity (*cf.*, *e.g.*, [10,12]). We do not expect such problems to be different in the railway sector with respect to other industries, but because cybersecurity for railway systems is a relatively young discipline, there may be a lack of evidence that security requirements engineering tools that have successfully been used in other sectors (*e.g.*, in the banking sector) can work in railway as well.

Controlled Natural Languages (CNLs) can help write security requirements in a way that they remain understandable by humans to express themselves and communicate ideas. CNLs use controlled grammar, precise semantics, and the possibility to process statements automatically. In previous work [20], it was shown that one particular CNL, *viz.*, the *Controlled Natural Language for Data Sharing Agreement* (CNL4DSA)—originally developed to formally model legal contracts regulating data sharing [6,23]—could serve to formalize security requirements about the “ERTMS L3 moving block” next generation railway signalling systems to increase capacity on railway tracks, reduce costs, and improve reliability [1,13].

¹ <https://rail-research.europa.eu/>.

² Documents of reference for the implementation of secure-by-design railway systems are either those of cybersecurity or are domain-specific, *e.g.*, the [Technical Specifications for Interoperability \(TSIs\)](#).

However, the work of engineers who translate Natural Language (NL) into CNL expressions is still based on experience and training. To reduce the risk of misinterpretation, a team of “coders” must work together, confronting, discussing, and assessing their translations for quality and expressivity.

Now, the question we want to explore in this paper is whether AI tools can help experts in the task of formalizing (cybersecurity) requirements, and we choose the railway domain as a use case. We follow the current hype concerning the capability of AI-based tools to interpret NL requirements (*e.g.*, for inconsistency or ambiguity detection). Indeed, the rise of *Large Language Models* (LLMs) and their application in AI-based tools has opened the path to new in-depth automatic analysis of NL requirements [7, 8].

In this regard, we decided to first exploit one of the larger LLMs available, *viz.*, ChatGPT, the chatbot by OpenAI built on top of OpenAI’s GPT3.5 model [4]. After a first experiment with ChatGPT, we replicate the process with Microsoft 365 Copilot, the chatbox powered by GPT-4, OpenAI’s LLM. We followed a similar line of *prompting* as with ChatGPT. Then, we compared the quality of the answers provided by the two tools. The comparison is meant to gather evidence on the utility of using AI in general for requirement analysis.

Even though one tool might be better suited to the task than the other, we intend to discuss the experience rather than champion a tool over another. The aim of the experiment is to exercise the capability of LLMs (ChatGPT, Copilot, and maybe others in the future) to support the human expert in the task of formalizing security railway system requirements in CNL4DSA.

2 Translating Railway Requirements Into CNL4DSA

The European Railway Traffic Management System (ERTMS) is a set of international standards for the interoperability, performance, reliability, and safety of modern European rail transport [13, 17]. It relies on the European Train Control System (ETCS), an automatic train protection system that continuously supervises the train, ensuring not to exceed the safety speed and distance. The current standards distinguish four levels (0-3) of operation of ETCS signalling systems, depending largely on the role of trackside equipment and on the way information is transmitted to and from trains. In particular, in the next generation Level 3 signalling systems, the train carries the Location Unit (LU) and OnBoard Unit (OBU) components, while the Radio Block Center (RBC) is a trackside component. The LU receives the train’s location from a Global Navigation Satellite System (GNSS), and sends this location, together with messages ensuring of the train’s integrity, to the OBU, which, in turn, sends the location to the RBC. Upon receiving a train’s location, the RBC sends a Message Authority (MA) to the OBU (together with speed restrictions and route configurations), indicating the space the train can safely travel based on the safety distance with preceding trains. The RBC computes the MA by communicating with neighbouring RBCs and by exploiting its knowledge of the positions of switches and other trains (head and tail position) by communicating with a Route Management System (RMS) [2].

2.1 ERTMS L3 Signalling System Security Requirements

Work in [2,3] extracts a series of requirements from the general description of the next generation ERTMS L3 signalling system. We present five of them, below:

Temporal Requirements

#	Description
R1	GNSS must send the train location to LU every 5 s.
R2	If the train position cannot be received within the maximum time limit, the OBU must transit to degraded mode.
R3	If the traiz integrity cannot be confirmed within the maximum time limit, OBU shall order the brake activation.

Alarm Triggering Requirements

#	Description
R4	If the connection between the RBC and OBU is lost, OBU must trigger an alarm.
R5	Once OBU receives an alarm, it must send it to RBC.

R1–R5 can be expressed in a Controlled Natural Language. Here, we translate the requirements in *Controlled Natural Language for Data Sharing Agreements* (CNL4DSA). The language was originally introduced to reduce the barrier of adoption of contractual agreements in terms of privacy as well as to ensure the agreement mapping to formal languages that allow its automatic verification [23]. A DSA is essentially a contract between two or more parties to agree on some terms and conditions with respect to data sharing and usage. This language can also support the enforcement of privacy and security of electronic data exchange, even in scenarios where multiple users have management rights on the same set of data [15]. Since its definition, it has been used in various fields, including healthcare, for expressing and analyzing guidelines related to clinical data [22], and product lines, to specifying and analyzing product families [16]. The interested reader can find the CNL4DSA syntax in [20].

We observe that **R1–R5** express obligations, as they all express some mandatory requirement for the L3 signalling system. We name subjects ‘subject’ and objects ‘object’ followed by a number (*e.g.*, subject1, subject2), while we name actions with a phrase, in slanted style, that reminds their doing (*hasCategory*). Actions are used in infix (*e.g.*, subject1 *Trigger* objects1), with the exception of the action of *s* sending *o* to *s'*, which we write in a mixed prefix-infix form as *s sendTo(s', o)*.

R1: IF c1 THEN MUST f1

where *c1* is a composite context and *f1* is an atomic fragment, defined as follows:

- *c1* = **IF** subject1 *hasRole* ‘GNSS’ **AND** subject2 *hasRole* ‘LU’ **AND** object1 *hasCategory* ‘TrainPosition’ **AND** object1 *isProvidedBy* subject1 **AND** object2 *hasCategory* ‘ElapsedTime’ **AND** object2 *hasValue* ‘5’.
- *f1* = (subject1, *sendTo*(subject2), object1)

where *sendTo(s')* is the action of sending to *s'*.

R2: IF c1 THEN MUST f1

where c1 is a composite context and f1 is an atomic fragment, defined as follows:

- c1 = subject1 hasRole ‘OBU’ AND object1 hasCategory ‘TrainPosition’ AND NOT object1 isReceivedBy subject1 AND object2 hasCategory ‘ElapsedTime’ AND object2 hasValue ‘maxtimelimit’ AND object3 hasCategory ‘mode’ AND object3 hasValue ‘Degraded’.
- f1 = ⟨subject1 ,Transit, object3⟩

R3: IF c1 THEN MUST f1

where c1 is a composite context and f1 is an atomic fragment, defined as follows:

- c1 = subject1 hasRole ‘train’ AND object1 hasRole ‘Integrity’ AND object1 isRelatedTo subject1 AND NOT object1 hasStatus ‘Confirmed’ AND object2 hasCategory ‘ElapsedTime’ AND object2 hasValue ‘MaxTimeLimit’ AND subject2 hasRole ‘OBU’ AND object3 hasCategory ‘OrderToBrake’.
- f1 = ⟨subject2, SentTo(subject1), object3⟩

R4: IF c1 THEN MUST f1

where c1 is a composite context and f1 is an atomic fragment, defined as follows:

- c1 = subject1 hasRole ‘RBC’ AND subject2 hasRole ‘OBU’ AND object1 hasCategory ‘Connection-RBC-OBU’ AND object1 hasStatus ‘Lost’ AND object2 hasCategory ‘Alarm’.
- f1 = ⟨subject2,Trigger,object2⟩

R5: AFTER f1 THEN MUST f2

where f1 and f2 are atomic fragments, defined as follows:

- f1 = ⟨subject1,Receive,object1⟩, where subject1 hasRole ‘OBU’ AND object1 hasCategory ‘alarm’;
- f2 = ⟨subject1,SendTo(subject2),object1⟩, where subject2 hasRole ‘RBC’.

3 Prompting Methodology

The practice of prompting machines (*i.e.*, asking questions to a bot so as to have meaningful answers back) is relatively old. It can be linked back to the famous Turing test, where the human’s questions are meant to figure out who is who behind a door, a person or an intelligent machine.

To refer to a more layman experience with actual human-computer interaction, we can recall Eliza, the bot that was capable of turning statements into questions, from the 70s. Its Emacs version (*i.e.*, Emacs doctor), still running today, offers a service as if it were a Rogerian psychotherapist to whom Emacs users can playfully chat by asking for advice.

Originally, the practice has gained momentum with the availability of expert systems [28] but only recently, with the advent of generative AI, it has become “engineering” (called *prompting engineering* [24]). The use of AI, and in particular to LLMs, to explore and generate knowledge is now booming, and the community has started proposing methodologies to guide engineering to a more scientifically sound line of prompting. We did not favour one particular strategy but resorted to a best practice. In fact, computer science researchers have debated mainly common-sense guidelines for effective prompting (cf., *e.g.*, [14,31]) that consider how AI models are trained and what they are good for and good at. Reasonable suggestions include the following: (i) Use Unambiguous Terminology; (ii) Ask One Question at a Time; (iii) Provide Context and Constraints; (iv) Utilize Prompts and Examples; and (v) Review and Refine.

Inspired by these suggestions, we decided to follow one of such best practices, for instance, to ensure that our tools understood the vocabulary used when talking of railway systems, as well as, the syntax of expressions in CNL4DSA. Then we took care of showing a few examples, a post-training phase if we wish to call it so, for later challenging our tools to provide some preliminary translation with the aim of correcting answers and letting AI learn by chatting with us. Eventually, we pose “the question”. This is when we take AI’s output seriously, and when we assess the AI’s answering quality by comparing the output (*i.e.*, CNL4DSA formula and relative explanation) with the formula that we came up with at the time when we performed the same exercise as expert researchers. It is the phase where we evaluate the performance of a machine versus that of us expert humans, so to speak.

To give minimal ecological validity to this research, we independently used two different AI tools: the free version of ChatGPT (the Italian team) and a licensed Microsoft 365 Copilot (the Luxembourg team). Then, we compared the results, devising a more general discussion about AI tools’ helpfulness in requirements engineering for railway cybersecurity.

Needless to say, our experiment has evident limitations, starting from the handful number of requirements considered: only five. However, even such a small number can be sufficient to give us elements to formulate further questions or hypotheses—following a sort of weak version of the grounded theory used in qualitative studies in behavioural sciences useful to devise future work to be pursued with more scientific rigour.

4 Ask ChatGPT!

ChatGPT³ is a chatbot developed by OpenAI and built on top of OpenAI’s GPT3.5 model [4]. It gathered a lot of attention from both research and industrial communities because it can solve a lot of tasks concerning a wide array of domains of knowledge.

OpenAI’s ChatGPT interacts in a conversational way, by answering both direct and follow-up questions, highlighting incorrect premises, and rejecting

³ <https://openai.com/blog/chatgpt>.

some inappropriate requests. ChatGPT is based on a Transformer architecture, which is becoming the preferred model for NL Processing (NLP) problems [29].

Following the methodology introduced in Sect. 3, we prompt ChatGPT with some tests that we made to get ChatGPT to translate the security requirements shown earlier, first showing it the syntax of the language and trying to guide it as best as we could.

This investigation is mainly driven by curiosity, a preliminary exploration into the world of AI, rather than by scientific research defined with rigour. The purpose is to get an idea of how this version of GPT responded to our simple queries. However, since we have followed a research process—even if we cannot call it an established methodology—for the sake of replicability, we point out the following elements of our process in this phase: (i) The version of ChatGPT used is the one that is freely available at the time of writing (May 2024), simply by registering on the OpenAI site (version 3.5);⁴ (ii) We did not use paid versions of GPT models, like OpenAI version 4, which, as advertised on their site, is their *most capable model*; (iii) OpenAI itself has let it be known that if we wait a while, a new, more experienced free version will be available;⁵ (iv) We did not train the model specifically to learn semi-formal languages like CNL4DSA. We did not do any fine-tuning; (v) We did not use strictly defined prompt patterns.

Having said that, to query ChatGPT, we opened a new chat, and all questions were asked in the same work session.

First, we let ChatGPT know that we have introduced it to the syntax of a language known as CNL4DSA. Figure 1 shows the prompt we gave to the LLM, plus its answer. Then, we gave ChatGPT the CNL syntax as input, plus some contextual information about what fragments, composite fragments, and contexts are.

Before asking the tool to translate the requirements presented in Sect. 2.1, we gave it an example of a translation. Specifically, requirement **R1**, on the communication, by GNSS, of the train position to the LU every 5s. We now describe what we wrote as a prompt.

As we can see from the screenshot in Fig. 2, after entering the request in NL, ChatGPT tried to make a first translation, but it did not take into account the conditions that trigger the fragment. Let us see what happens when we give some more information about fragments and contexts constructs and show the sample translation of **R1** (cf. Figs. 3 and 4).

The following construct is the answer we were given after showing how we translated **R1**. Basically, ChatGPT copied what it was given as our translation:

```
R1: IF subject1 hasRole 'GNSS' AND subject2 HASROLE 'LU' AND
  object1 HASCATEGORY 'TrainPosition' AND object1
ISPROVIDEDBY subject1 AND object2 HASCATEGORY
  'ElapsedTime' AND object2 HASVALUE '5' THEN MUST subject1
  sendTo(subject2, object1)
```

⁴ <https://chatgpt.com/>.

⁵ <https://openai.com/index/gpt-4o-and-more-tools-to-chatgpt-free/> (accessed 16/5/24).

You
Good morning Chat GPT, I show you the syntax of a language I defined a few years ago. Among other things, it is used to describe security requirements. Here is the syntax.

ChatGPT
Good morning! I'd be happy to help you with your language syntax. Please go ahead and share it with me, and I'll do my best to assist you with understanding and interpreting it.

You
OK, here's the syntax, together with some explanations. The language is called CNL4DSA, a rather mysterious name.

ChatGPT
Great, let's take a look at the CNL4DSA syntax along with its explanations. Whenever you're ready, please share the details!

Fig. 1. First prompt: explaining the task

You
I wish you could write some safety requirements for a railway system in CNL4DSA. I'll give you an example first, a requirement for a Global Navigation Satellite System (GNSS) to communicate the position of a train to an LU leasing unit. The natural language specification is as follows: R1: GNSS must send the train location to LU every 5 seconds

ChatGPT
Certainly! Let's translate the safety requirement for the railway system into CNL4DSA. Based on your example, here's how it would look:

```
vbnet Copy code  
R1: MUST GNSS send train_location to LU every_5_seconds
```

Fig. 2. Second prompt: giving an example requirement in NL and a translation attempt

You
Before you do the translation, I'll show you how I translated R1 and give you some more information on how to proceed. The information you might need is as follows: We name subjects 'subject' and objects 'object' followed by a number (e.g., subject1, subject2), while we name actions with a phrase, in slanted style, that reminds their doing (hasCategory). Actions are used in infix (e.g., subject1 Trigger objects1), with the exception of the action of s sending o to s', which we write in a mixed prefix-infix form as s sendTo(s', o)

Fig. 3. Third prompt: giving some explanations for the formation of correct conditions

You
 And here is my translation of requirement R1: IF c1 THEN MUST f1
 where c1 is a composite context and f1 is an atomic fragment, defined as follows:
 - c1 = IF subject1 hasRole 'GNSS' AND subject2 hasRole 'LU' AND object1
 hasCategory 'TrainPosition' AND object1 isProvidedBy subject1 AND ob-
 ject2 hasCategory 'ElapsedTime' AND object2 hasValue '5' and f1 = <subject1,
 sendTo(subject2), object1>

ChatGPT
 Got it. Based on your translation, here's the refined CNL4DSA representation of requirement R1:

Fig. 4. Fourth prompt: with our translation of the first requirement in CNL4DSA

Let us see what happens if we feed it another requirement, **R2**, and give it nothing as a hint. This is the result of the translation:

**R2: IF object1 hasCategory 'ElapsedTime' AND object1 hasValue
 'max_time_limit' AND NOT MUST GNSS sendTo(OBU, train_position)
 THEN MUST OBU transitTo(degraded_mode)**

As we can see, the first two conditions on Elapsed Time and Max Time Limit have been well expressed. The conditions on the other subjects and objects of the requirement (*i.e.*, on the fact that the subject that will then perform the action in the fragment is the OBU, on the fact that there are other entities representing the position of the train and the degraded mode) were not instantiated. Also, a syntax error was made (beyond the unclear semantics of **NOT MUST GNSS sendTo(OBU, train_position)**): the syntax does not allow **NOT** before a fragment, only before a context. We then tried to give ChatGPT a prompt that explains these things.

You
 Thanks for your translation. However, I see that you did not respect the syntax of the language because in CNL4DSA the NOT is not placed before an f fragment, but only before a c context. Can you reformulate the translation according to the right syntax?

ChatGPT
 Apologies for the oversight. Let's correct that. Here's the revised translation respecting the syntax of CNL4DSA:

```

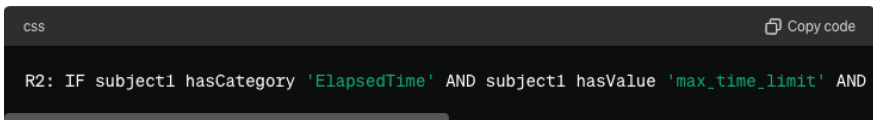
    CSS Copy code
    t' AND NOT IF GNSS sendTo(OBU, train_position) THEN MUST OBU transitTo(degraded_mode)
    
```

Fig. 5. Fifth prompt: explaining that there is a syntax error.

There are still a few things that need to be fixed. For instance, we notice that there was some confusion between the text of the first and second requirement, in fact, the screenshot in Fig. 5 shows the term GNSS, which is correct for Requirement **R1**, but does not appear in **R2**. We also point out that in the syntax there is only one **IF** with all necessary conditions following, while in the last translation they fulfill more than one **IF**. We go on like this for a while, trying to point out the errors. We do not show all the proofs so as not to bore the reader.

However, we think it is interesting to show the last attempt that led us to give up. Among the various correction suggestions, at one point, as a result of the translation, two entities appear, *subject1* and *subject2*, which have never been introduced before. At this point, the prompt asks: *I still see some problems: You have never defined who subject1 and subject2 are.*

Here is the answer (Fig. 6). Basically, the term *subject1* was introduced in place of other terms that were included in one of the first answers, see the first translation of **R2** above.



```
css Copy code
R2: IF subject1 hasCategory 'ElapsedTime' AND subject1 hasValue 'max_time_limit' AND
```

Fig. 6. Sixth prompt: erroneous replacement

We conclude that simply showing the syntax of a language, with some explanations on the side, does not suffice to get meaningful translations: more training is needed to evaluate the capability of ChatGPT to perform more accurate translations.

5 Ask Copilot!

To improve the external validity of our exercise we repeated the exercise with Microsoft 365 Copilot⁶, Microsoft’s AI assistant that relies on OpenAI tools, mainly ChatGPT-4 and DALL-3. It also relies on Bing for scraping text from the Internet and Azure cloud, as well as on several other utilities developed by Microsoft for NLP and retrieval augmentation generation. We hold a licence, giving us access to the full power of the underlying AI models.

An interesting feature of Copilot is that it allows us to choose among three different tones: “creative”, “balanced”, or “precise”.⁷ We chose “precise” for obvious reasons since we intend to handle formal expressions where there is no room for creativity or hallucinations.

⁶ <https://copilot.microsoft.com>.

⁷ These parameters can be thought like the variable “temperature” in other LLMs regulates the degree of randomness used in producing the output, but what really is behind the scenes is still debated in *fora* like Reddit. Copilot itself, when questioned about it, replies as follows: “They are not directly related to an LLM’s “temperature” but they do affect the output in a somewhat similar way”.

Then, we followed similar steps, engaging the tool in learning the syntax of CNL4DSA first. That “training” was followed by prompting Copilot with the first paragraph in Sect. 2, which contains definitions and terms essential to build up the terminology used in the remainder of the dialogue. The scope of this training was to give the tool a context and to verify whether its output was intuitively meaningful. Copilot replied with a “simplified breakdown” of the text with keywords in bold, itemised text, and mathematical expressions in italics. showing that Copilot can process paragraphs that contain a mix of phrases and formulas. The same happened when we pasted a paragraph with the jargon used when talking about ERTMS L3 and railway systems. This pre-phase prompting is known as “context stuffing” We prompt with “knowledge” and vocabulary to ensure that Copilot, when generating text, relies on the information we gave it and not on that in its pre-trained model.

The next step was to show examples of CNL4DSA formula expressing requirements **R1–R2**. This was a tedious “just listen to me and learn” phase, where we disregarded the usual Copilot rephrasing. We omit the details.

With **R3**, we started to test Copilot’s capacity. Copilot’s answer, in Table 1, merits a discussion. What stands out at first is that Copilot is writing in a good CNL4DSA syntax and style, similar to the one we used in our exercise. About the phrasing and the semantics of the formalization, if we compare it to our formalization of **R3** (cf. Sect. 2), the difference that emerges is in the selection of subjects. For us, a subject is “the train”: it is the entity whose lack of integrity is confirmed (by whom is however unclear); another subject is the ‘OBU’, the unit that orders to brake. In Copilot, there is one subject: the ‘OBU’: when it confirms (to itself) that the property ‘TrainIntegrity’ does not hold, it gives the order to brake.

Table 1. Copilot’s output with **R3**: “If the train integrity cannot be confirmed within the maximum time limit, OBU shall order the brake activation”



In the CNL4DSA formal language, this requirement can be expressed as follows: **R3**: IF *c1* THEN MUST *f1*. Here


- *c1* is a composite context defined as follows:

```
subject1 hasRole 'OBU' AND
object1 hasCategory 'TrainIntegrity' AND NOT
object1 isConfirmedBy subject1 AND
object2 hasCategory 'ElapsedTime' AND
object2 hasValue 'maxtimelimit' AND
object3 hasCategory 'BrakeActivation'.
```

- *f1* is an atomic fragment defined as follows: $\langle \text{subject1, order, object3} \rangle$

In this context, *order* represents the action of ordering a particular operation. So, the requirement **R3** states that if the conditions in *c1* are met (*i.e.*, if the train integrity cannot be confirmed within the maximum time limit), then *subject1* (the OBU) must order the brake activation. This requirement ensures that the system can respond appropriately when it encounters issues with confirming the train’s integrity. ©

Table 2. Copilot’s output to **R4**: “If the connection between the RBC and OBU is lost, OBU must trigger an alarm”, and to **R5**: “Once OBU receives an alarm, it must send it to RBC” in CNL4DSA.




In the CNL4DSA formal language, this requirement can be expressed as follows
R4: IF c1 THEN MUST f1. Here
 - **c1** is a composite context defined as follows:

```

subject1 hasRole 'RBC' AND
subject2 hasRole 'OBU' AND
NOT (subject1 isConnectedTo subject2)
AND object1 hasCategory 'Alarm'.

```

- **f1** is an atomic fragment defined as: $\langle \text{subject2, trigger, object1} \rangle$
 Here ‘trigger’ represents the action of triggering an operation. So, the requirement R4 states that if the conditions in **c1** are met (i.e., if the connection between the RBC and OBU is lost), then ‘subject2’ (the OBU) must trigger an alarm. This requirement ensures that the system can respond appropriately when it encounters issues with the connection between the RBC and OBU. ☺

(a) Copilot’s output to **R4**


In the CNL4DSA formal language, this requirement can be expressed as follows:
R5: AFTER f1 THEN MUST f2. Here:
 - **f1** is an atomic fragment defined as follows:

$$\langle \text{subject1, Receive, object1} \rangle$$

where **subject1** hasRole ‘OBU’ AND
object1 hasCategory ‘Alarm’.
 - **f2** is an atomic fragment defined as follows:

$$\langle \text{subject1, SendTo}(\text{subject2}), \text{object1} \rangle$$

where **subject2** hasRole ‘RBC’.
 Here ‘Receive’ and ‘SendTo(*s*)’ represent the actions of receiving an operation and sending to ‘*s*’, respectively. So, the requirement R5 states that once ‘subject1’ (the OBU) receives ‘object1’ (an alarm), it must send ‘object1’ (the alarm) to ‘subject2’ (the RBC). This requirement ensures that the system can respond appropriately when the OBU receives an alarm. ☺

(b) Copilot’s output to **R5**

The Copilot’s version is reasonable. We also notice that **R3**’s description could be clearer: the sentence “the train integrity cannot be confirmed” is, in fact, ambiguous: the word phrase “train integrity” can be interpreted as a property. To stress that “train” is a subject, the phrase should be written as “train’s integrity”. Copilot helped us to spot this source ambiguity.

Another remarkable observation is the natural language explanation of the expression. It may be considered as an explanation provided by Copilot. To us, it

serves as a cross-check for the correctness of the Copilot’s CNL4DSA expression.⁸ We continued with **R4**. We are still in the phase in which we intend to teach rather than test Copilot, but Copilot’s answer, this time, is almost as precise as the one we gave. Table 2a reports the answer. Copilot correctly recognizes both subjects, ‘OBU’ and ‘RBC’. Copilot models that the connection is lost as a predicate between the subjects, *i.e.*, the quality of them not being connected. Our modelling was more convoluted: the connection between ‘OBU’ and ‘RBC’ is an object whose status is ‘Lost’. We think that the version of Copilot is correct and more concise than ours.

With **R5**, Copilot shows its remarkable ability. Table 2b reports the output. Although there was no previous example about it, Copilot correctly uses the modality **AFTER** to express the causality of two events: “the alarm that is received by one subject, the ‘OBU’”, and “the alarm sent to the other subject, the ‘RBC’”. Copilot wrote in compliance with the CNL4DSA syntax, which requires **AFTER**’s being accompanied by two subjects.

We have to admit that Copilot’s formulation is as good as ours, if not slightly better. It was an astonishing discovery, and it was only after a handful of examples. It suggests that Copilot holds the potential to be used as a support for ‘translation’ checking. More evidence is, of course, necessary in support of this claim: “one swallow does not make a summer”. But it is an encouraging start.

6 Related Work

Since LLMs are about interpreting, transforming and generating text, it should not come as a surprise that an increasing number of researchers have engaged in studying the use of LLMs in requirements engineering, a discipline that includes defining and documenting requirements as concisely and clearly as one can.

The most efficient way to obtain a taste of the current research in this field is to refer to surveys of the state of the art. The ones we have found by browsing for “LLMs AND formal requirements” in the Google Scholar search engine discuss the use of LLMs in support of requirements specification. From different perspectives, they all recall the basic peculiarities and the different architecture of LLMs, a useful exercise to have realistic expectations about their capabilities. Beyond that common premise, they diversify their contribution. LLM-based generation is not always the more appropriate approach than a human-generated requirement, and the decision depends on the availability of datasets with which to train the model [30], and LLMs models (*i.e.*, Llama-2, CodeLlama, ChatGPT-4) are not equally performant in finding incongruities and ambiguities [8, 21]. Relevantly, LLMs’ output can be measured for clarity, conciseness, verifiability, and understandability [19].

Since the goal is to discuss whether LLMs help analyse unstructured documents, these works are only partially related to the task of reformulating require-

⁸ Also ChatGPT returned an informal explanation of the meaning of the CNL4DSA expressions it has produced; being these latter incorrect, the explanations were meaningless but still useful to confirm the presence of problems in the translation.

ments into formal languages. However, the authors of [5] do map requirements into temporal logic formulas, even if it does so for the sake of spotting linguistic ambiguities. The authors offer their own framework (called `n12spec`) but also analyse how well other LLMs (*i.e.*, Codex, ChatGPT-3.5-turbo, and Bloom) translate natural language into logic formulas before and after an interaction with human coders and with minimal prompting. Their experiment, involving students as reviewers, shows that only after a round of interactions the ratio of corrected translations improves from about 30% to about 87%, which suggests that the task of translation needs humans in the loop to achieve a higher level of quality. This outcome is consistent with our preliminary test: the translation by Copilot is superior to that by ChatGPT because of a more elaborate prompting and because of a round of interactions with the human teams.

The ability of LLMs to reason formally has been put to a test in [18]. The authors tested state-of-the-art LLMs, like ChatGPT 4, to translate natural language descriptions of propositional and first-order logic formulas into their formal form and back. Their experiment shows that accuracy quickly and disappointedly decreases with the length of formulas. However, their formulas were generated fully at random, which suggests a speculative hypothesis: contextualizing the translation into a specific domain (as we did by prompting our LLMs about railway system engineering jargon) is a necessary pattern to ensure that LLMs can correctly parse requirements that contain technical terms.

We have to stress that the majority of the works we have found on the subject of “LLMs and formal requirements” are published in `arxiv.org`. With the only exception of [5], published at the Conference on Computer Aided Verification (CAV), they do not come with verified and replicable artefacts (which we believe should be mandatory even for non-peer-reviewed works). Thus, any conclusion we may draft from this brief analysis of related work is inevitably partial and temporary. That premised, it seems that investigating LLMs and formal requirements is a field of research that is yet unexplored to its full extent. This encourages us to look into the matter more systematically, starting with defining a rigorous methodology of prompting, verifying the output for quality, and building a dataset of annotated samples of translations.

We conclude this section by referring the reader to [21]: it offers a comparative analysis of works that studied ChatGPT’s efficiency in eliciting requirements, offering insights into, we quote, “the effectiveness of ChatGPT, the importance of human feedback, prompt engineering techniques, technological limitations, and future research directions in using LLMs in software requirements engineering”.

7 Conclusion

In previous work [20] a particular Controlled Natural Language called CNL4DSA—praised for precision, expressiveness, naturalness, and simplicity—was proved to be apt to express railway-related cybersecurity properties, such as those found in a subset of real control rules about the “ERTMS L3 moving block” next-generation railway signalling systems.

Nevertheless, expressing requirements in a controlled natural language requires specific skills, and years of experience in working with requirements and formal languages. It was thus a valid question to explore whether AI could help with the task. In this work, we started collecting evidence to test this hypothesis. We experimented with the capability of AI, in particular of LLMs, to replace the human “formalizer” in the task of expressing NL requirements in the CNL4DSA. We did so through a simple proof-of-concept experiment to get a first idea of how AI performed when entrusted, *i.e.*, prompted, with the task.

We independently tested two different, yet related, AI tools: ChatGPT, a free version, and Microsoft 365 Copilot, a licensed version. We fed the models with some “contextual information” (*i.e.*, the syntax of the CNL4DSA, some text exemplifying the terminology used in the railway ERTMS jargon) and with examples of correct translation before testing their quality of translation into CNL4DSA with five requirements expressed in natural language. Here, quality is assessed, informally, as structure similarity and semantic consistency when compared with the translations we came out with as experts. We, human experts in formal languages and in CNL4DSA, acted as verifiers.

The preliminary results with ChatGPT reveal insights that are somehow surprising. ChatGPT was asked to provide a translation immediately after the “contextual information”. The output showed that, despite some basic ability, ChatGPT apparently needs more supervision to produce accurate translations.

The experiment with Microsoft 365 Copilot, more structured, turned out promising. After the common prompting about the context, we followed with a phase where we gave examples one by one, letting the tool learn our translations, and correcting them when they were inconsistent with ours. Copilot’s quality of translation improved considerably during the dialogue to a point where we believe one of its translations was actually superior to ours.

This sneak-preview exploration into AI’s capabilities to translate requirements from NL into CNL has no significant validity to allow us to draw any solid conclusion. Indeed, we have tested the translation capability on a small number of security requirements. Other LLMs and their versions, both free and premium, would need to be evaluated, and one could explore in depth a refinement for the prompt engineering phase. However, our preliminary results suggest that it is worth setting up some future work to explore up to which point LLMs can support human “formalizers” in their work. It also suggests, assuming to have the ability to build up a sufficiently large set of requirements with golden datasets of correct translations, that a dedicated and personalized AI model might greatly help the community, considering that already a general-purpose one has been able to give meaningful suggestions. One of the challenges we foresee is quantifying the “quality of translation” as a metric to assess AI’s output quality: in this work, our assessment was simplistically based on our expert’s yet subjective judgement. Previous work on measuring AI’s output quality in terms of false positives and negatives can be referred to as a starting point for this future work [5,8]. Explainable AI models would help, too [25].

Acknowledgements. This work has been written for the Colloquium in honour of Professor Rocco De Nicola, Rector of IMT School for Advanced Studies Lucca, in Lucca, Italy, held at ISoLA 2024 in late October 2024. All the authors of this paper have fruitfully worked with Rocco over the years and wish to express their professional and personal gratitude to him for the years spent together. We prepared this unpretentious paper, which brings together the theme of languages, a great passion of Rocco since his PhD years, with trendy topics of the moment, LLMs and their potential ability to support human reasoning, but also to misguide it if not appropriately prompted. The experiment was not very elaborate in terms of prompting, we mostly used common sense, but it was insightful. With curiosity, the seed of science, we are now keen to discover whether our experiment can be reproduced in a reliable enough way so as to generalize our findings or if, instead, different prompts inevitably lead to different results, thus falsifying our hypothesis that AI holds great potential in helping to formalize (cybersecurity) requirements. We hope that Rocco appreciates this little tribute.

References

1. Bartholomeus, M., et al.: The use of formal methods in specification and demonstration of ERTMS Hybrid Level 3. *IRSE News* **260**, 14–17 (2019). <https://www.irse.org/LinkClick.aspx?fileticket=pKNpGWY33CA%3d&portalid=0>
2. Basile, D., ter Beek, M.H., Ferrari, A., Legay, A.: Modelling and analysing ERTMS L3 moving block railway signalling with Simulink and UPPAAL SMC. In: Larsen, K.G., Willemse, T. (eds.) *FMICS 2019*. LNCS, vol. 11687, pp. 1–21. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-27008-7_1
3. Basile, D., ter Beek, M.H., Ciancia, V.: Statistical model checking of a moving block railway signalling scenario with UPPAAL SMC. In: Margaria, T., Steffen, B. (eds.) *ISoLA 2018*. LNCS, vol. 11245, pp. 372–391. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03421-4_24
4. Brown, T.B., et al.: Language models are few-shot learners. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) *Advances in Neural Information Processing Systems 33: Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS 2020)*, pp. 1877–1901 (2020). <https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfc4967418bfb8ac142f64a-Abstract.html>
5. Cosler, M., Hahn, C., Mendoza, D., Schmitt, F., Trippel, C.: nl2spec: interactively translating unstructured natural language to temporal logics with large language models. In: Enea, C., Lal, A. (eds.) *CAV 2023*. LNCS, vol. 13965, pp. 383–396. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-37703-7_18
6. Costantino, G., Martinelli, F., Matteucci, I., Petrocchi, M.: Efficient detection of conflicts in data sharing agreements. In: Mori, P., Furnell, S., Camp, O. (eds.) *ICISSP 2017*. CCIS, vol. 867, pp. 148–172. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93354-2_8
7. Fantechi, A., Gnesi, S., Passaro, L.C., Semini, L.: Inconsistency detection in natural language requirements using ChatGPT: a preliminary evaluation. In: Schneider, K., Dalpiaz, F., Horkoff, J. (eds.) *Proceedings of the 31st IEEE International Requirements Engineering Conference (RE 2023)*, pp. 335–340. IEEE (2023). <https://doi.org/10.1109/RE57278.2023.00045>

8. Fantechi, A., Gnesi, S., Semini, L.: Rule-based NLP vs ChatGPT in ambiguity detection, a preliminary study. In: Ferrari, A., et al. (eds.) Joint Proceedings of REFSQ-2023 Workshops, Doctoral Symposium, Posters & Tools Track and Journal Early Feedback: 6th Workshop on Natural Language Processing for Requirements Engineering (NLP4RE 2023). CEUR Workshop Proceedings, vol. 3378. CEUR-WS.org (2023). <https://ceur-ws.org/Vol-3378/NLP4RE-paper1.pdf>
9. Ferrari, A., ter Beek, M.H.: Formal methods in railways: a systematic mapping study. *ACM Comput. Surv.* **55**(4), 69:1–69:37 (2023). <https://doi.org/10.1145/3520480>
10. Ferrari, A., Lipari, G., Gnesi, S., Spagnolo, G.O.: Pragmatic ambiguity detection in natural language requirements. In: Bencomo, N., Cleland-Huang, J., Guo, J., Harrison, R. (eds.) Proceedings of the 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE 2014), pp. 1–8. IEEE (2014). <https://doi.org/10.1109/AIRE.2014.6894849>
11. Ferrari, A., Mazzanti, F., Basile, D., ter Beek, M.H.: Systematic evaluation and usability analysis of formal methods tools for railway signaling system design. *IEEE Trans. Softw. Eng.* **48**(11), 4675–4691 (2022). <https://doi.org/10.1109/TSE.2021.3124677>
12. Ferrari, A., Spoletini, P., Gnesi, S.: Ambiguity cues in requirements elicitation interviews. In: Proceedings of the 24th International Conference on Requirements Engineering (RE 2016), pp. 56–65. IEEE (2016). <https://doi.org/10.1109/RE.2016.25>
13. Furness, N., van Houten, H., Arenas, L., Bartholomeus, M.: ERTMS level 3: the game-changer. *IRSE News* **232**, 2–9 (2017). <https://www.irse.nl/resources/170314-ERTMS-L3-The-gamechanger-from-IRSE-News-Issue-232.pdf>
14. Gewirtz, D.: How to write better ChatGPT prompts in 5 steps. *ZDNET* (online magazine) (2024). <https://www.zdnet.com/article/how-to-write-better-chatgpt-prompts-in-5-steps/>
15. Gnesi, S., Matteucci, I., Moiso, C., Mori, P., Petrocchi, M., Vescovi, M.: My data, your data, our data: managing privacy preferences in multiple subjects personal data. In: Preneel, B., Ikonomidou, D. (eds.) APF 2014. LNCS, vol. 8450, pp. 154–171. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06749-0_11
16. Gnesi, S., Petrocchi, M.: Towards an executable algebra for product lines. In: Proceedings of the 16th International Software Product Line Conference (SPLC 2012), vol. 2, pp. 66–73. ACM (2012). <https://doi.org/10.1145/2364412.2364424>
17. International Union of Railways (UIC): ERTMS/ETCS Systems Requirements Specification (1999)
18. Karia, R., Dobhal, D., Bramblett, D., Verma, P., Srivastava, S.: Can LLMs Converse Formally? Automatically Assessing LLMs in Translating and Interpreting Formal Specifications (2024). <https://arxiv.org/abs/2403.18327>
19. Krishna, M., Gaur, B., Verma, A., Jalote, P.: Using LLMs in Software Requirements Specifications: An Empirical Evaluation (2024). <https://arxiv.org/abs/2404.17842>
20. Lenzini, G., Petrocchi, M.: Modelling of railway signalling system requirements by controlled natural languages: a case study. In: ter Beek, M.H., Fantechi, A., Semini, L. (eds.) From Software Engineering to Formal Methods and Tools, and Back. LNCS, vol. 11865, pp. 502–518. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30985-5_29
21. Marques, N., Silva, R.R., Bernardino, J.: Using ChatGPT in software requirements engineering: a comprehensive review. *Future Internet* **16**(6) (2024). <https://doi.org/10.3390/fi16060180>

22. Martinelli, F., Matteucci, I., Petrocchi, M., Wiegand, L.: A formal support for collaborative data sharing. In: Quirchmayr, G., Basl, J., You, I., Xu, L., Weippl, E. (eds.) CD-ARES 2012. LNCS, vol. 7465, pp. 547–561. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32498-7_42
23. Matteucci, I., Petrocchi, M., Sbodio, M.L.: CNL4DSA: a controlled natural language for data sharing agreements. In: Shin, S.Y., Ossowski, S., Schumacher, M., Palakal, M.J., Hung, C. (eds.) Proceedings of the 25th Symposium on Applied Computing (SAC 2010), pp. 616–620. ACM (2010). <https://doi.org/10.1145/1774088.1774218>
24. Sahoo, P., Singh, A.K., Saha, S., Jain, V., Mondal, S., Chadha, A.: A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications (2024). <https://doi.org/10.48550/arXiv.2402.07927>
25. Seisenberger, M., et al.: Safe and secure future AI-driven railway technologies: challenges for formal methods in railway. In: Margaria, T., Steffen, B. (eds.) ISoLA 2022. LNCS, vol. 13704, pp. 246–268. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-19762-8_20
26. Soderi, S., Masti, D., Hämäläinen, M., Iinatti, J.H.: Cybersecurity considerations for communication based train control. *IEEE Access* **11**, 92312–92321 (2023). <https://doi.org/10.1109/ACCESS.2023.3309005>
27. Soderi, S., Masti, D., Lun, Y.Z.: Railway cyber-security in the era of interconnected systems: a survey. *IEEE Trans. Intell. Transp. Syst.* **24**(7), 6764–6779 (2023). <https://doi.org/10.1109/TITS.2023.3254442>
28. Straach, J., Truemper, K.: Learning to ask relevant questions. *Artif. Intell.* **111**(1–2), 301–327 (1999). [https://doi.org/10.1016/S0004-3702\(99\)00037-5](https://doi.org/10.1016/S0004-3702(99)00037-5)
29. Vaswani, A., et al.: Attention is all you need. In: Guyon, I., et al. (eds.) Advances in Neural Information Processing Systems 30: Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS 2017), pp. 5998–6008 (2017). <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>
30. Vogelsang, A., Fischbach, J.: Using Large Language Models for Natural Language Processing Tasks in Requirements Engineering: A Systematic Guideline (2024). <https://arxiv.org/abs/2402.13823>
31. Vv.Aa.: Effective Prompts for AI: The Essentials. MIT Sloan Teaching & Learning Technologies (online magazine) (2023). <https://mitsloanedtech.mit.edu/ai/basics/effective-prompts/>