LDK 2023

Human-Machine Collaborative Annotation: A Case Study with GPT-3

Ole Magnus Holter
University of Oslo, Norway
olemholt@ifi.uio.no

Basil Ell
University of Oslo, Norway
Bielefeld University, Germany
basile@ifi.uio.no

Abstract

Within industry, it is vital to adequately commu-nicate the qualities and features of what is to be built, and requirements are important artefacts for this purpose. Having machine-readable re-quirements can enhance the level of control over the requirements, allowing more efficient requirement management and communication.

Training a semantic parser typically requires a dataset with thousands of examples. However, creating such a dataset for textual require-ments poses significant challenges. In this study, we investigate to what extent a large language model can assist a human annotator in creating a gold corpus for semantic parsing of textual requirements.

The language model generates a semantic parse of a textual requirement that is then corrected by a human and then added to the gold standard. Instead of incrementally fine-tuning the language model on the growing gold standard, we investigate different strategies of including examples from the growing gold standard in the prompt for the language model.

We found that selecting the requirements most semantically similar to the target sentence and ordering them with the most similar requirement first yielded the best performance on all the metrics we used. The approach resulted in 41 % fewer edits compared to creating the parses from scratch, – thus, significantly less human effort is involved in the creation of the gold standard in collaborative annotation. Our findings indicate that having more requirements in the gold standard improves the accuracy of the initial parses.

1 Introduction

Requirements describe the qualities that a physical product or a service must provide. They are an important part of industry communication, and often parts of contracts. Thus, the requirements legally bind the contractor and the supplier, and failing to comply with them can mean both legal and economic undesired consequences.

Having the requirements expressed in a computer-understandable format would be beneficial. Manual tasks, such as requirement retrieval and documentation could be automated. In addition, it can lay the foundation for automatic compliance checking of project descriptions with the requirements. Ideally, requirements should natively be formulated in a machine-readable format, i.e., when they are created. However, the reality is that the industry must work with a large number of existing requirements, most of them embedded in complex domain-specific documents written for subject-matter experts.

To address this challenge, semantic parsing offers a promising solution by transforming natural language text into a logical representation. To create a semantic parser, however, we need training data, and manually creating logical representations is a tedious and error-prone task. Moreover, the complexity of the documents and the language of these texts makes it difficult to use techniques such as crowd-sourcing. Since it requires a considerable amount of expert hours, it is an expensive undertaking. Automatic or semi-automatic methods that help us to create training data could result in substantial savings in both cost and labour.

Recent advances in large language models (LLMs) have resulted in generic models that can solve many NLP tasks without fine-tuning them on a task-specific corpus (Liu et al., 2019; Raffel et al., 2020). While the typical LLM benchmarks do not include semantic parsing, some works demonstrate that LLMs are capable of producing accurate semantic parses (Shin et al., 2021; Roy et al., 2022).

There has, however, been little focus on using LLMs for semantic parsing in complex domains such as industry standards or requirements. Furthermore, to the best of our knowledge, no work has addressed human-in-the-loop LLM-supported

semantic parsing or LLM-supported creation of semantic parsing gold standard datasets that can then be used to train semantic parsers.

While some attention has been given to sample selection and ordering for in-context learning (as a means of few-shot learning), most studies focus on common datasets where the approaches have full access to a gold standard. To the best of our knowledge, no study has investigated sample selection for in-context learning (few-shot learning) from an iteratively growing set of possible examples of industry requirements. In our scenario, the initial set of examples is empty and is populated via human-machine collaboration.

In this paper, we investigate the possibility to use GPT-3 to reduce the effort of creating a gold standard for semantic parsing of industry requirements to description logic. To conduct the study, we compile and annotate a dataset consisting of requirement sentences, all written in English, from various industry domains. The sentences are sampled from documents by Det Norske Veritas (DNV), a global risk management and classification corporation with a focus on standards and requirements.

We hypothesize that while a semantic parser, based on a large language model, may not consistently produce logically correct formalizations, the generated formalizations are often close to the desired form. Consequently, correcting them is easier for a human than creating logical formalizations from scratch. Our focus in this study is not to create a semantic parser for a particular application, but rather to demonstrate that this method can be used to quickly create high-quality training data.

Furthermore, we investigate how sample selection and ordering affect the performance on this specific task with technical, complex input texts and description logic as output and an iteratively increasing number of available examples. We then examine the decrease in human effort between manually creating logical representations vs. correcting LLM-generated logical representations.

The remainder of the paper is structured as follows. Section 2 gives an overview of related work. Section 3 describes the problem in more detail. In Section 4, we describe the method, while in Section 5 we present the results of the experiments. The discussion and the conclusion are found in Section 6 and Section 7, respectively. In Section 8 we describe the limitations of this study and sketch ideas for future work.

2 Related work

LLM prompting The transformer model, introduced by (Vaswani et al., 2017), was followed by (Devlin et al., 2019), who pretrained a bidirectional transformer model (BERT) on a large text corpus. The BERT model, together with its many variants, has been used to solve many different tasks in NLP. It has been shown that these models already contain a vast amount of knowledge (Petroni et al., 2019; Roberts et al., 2020). While fine-tuning to a specific task has been the preferred way of using such models (Raffel et al., 2020), prompting has more recently been suggested as an alternative approach (Petroni et al., 2019) and has been used for many tasks.

While many LLM prompts are manually created, several works have investigated the automatic generation or improvement of prompts. Haviv et al. (2021) propose to automatically rewrite queries to learn how to better query an LLM, while Jiang et al. (2020) propose to mine patterns from a corpus. Sample selection and ordering in a prompt can also have a large impact on performance. It is, however, hard to predict which order is better than another as this can change from task to task and from model to model (Lu et al., 2022). Liu et al. (2022) find that choosing examples semantically similar to the target task improves GPT-3's in-context learning performance on various tasks over a random baseline. They also observed that the ordering of the n most similar examples affects performance, but that different ordering performed best for different datasets. The impact of the ordering, however, was comparably small. Chang et al. (2021) propose to use clustering and select one element from each cluster to ensure good coverage of examples. They demonstrate that this strategy outperforms random selection. For a more detailed overview of prompting methods, strategies, and applications, see (Liu et al., 2023).

Prompt-based semantic parsing Several recent works on prompt-based semantic parsing have used constrained language models (Shin et al., 2021; Yang et al., 2022b). The models are constrained so that they will answer with a syntactically correct natural language equivalent of a semantic parse, i.e., a canonical form, that can be converted to a logical formalism by means of a synchronous context-free grammar. BenchCLAMP (Roy et al., 2022) was proposed as a benchmark specifically to evaluate se-

mantic parsing methods with constrained language models. A different approach was suggested by Rongali et al. (2022). The approach learns a mapping from natural language to a canonical form by jointly training a seq2seq model using masked prediction, denoising, and supervised semantic parsing examples using very little data.

While the constrained language model's output to a canonical format can be considered a form of paraphrasing, another way to use an LLM as part of a semantic parsing pipeline is to use an LLM to augment real datasets or to synthesize training data for semantic parsing by paraphrasing real examples or examples generated by a grammar (Yang et al., 2022a; Rongali et al., 2022).

As an extension to manually created prompts for semantic parsing, prompt tuning was proposed by Schucher et al. (2022). In their study, a trainable embedding is prepended at all layers of the language model, which is shown to outperform a fine-tuned T5 model (Raffel et al., 2020). In addition, the authors demonstrate that the performance gap between generating a logical representation directly and using a canonical form reduces as the size of the T5 model increases.

Regarding sample selection for semantic parsing, Shin et al. (2021) propose to use GPT-3 to select the n most relevant examples for a target sentence. They do not, however, show how it compares to other sample selection methods or consider the sample ordering.

Training data generation Wang et al. (2021) suggest that instead of using LLMs to directly produce a label (few-shot) to solve a classification task, one could use a couple of examples and a label as a prompt to generate "gold" data. They achieve better results when using the generated data to fine-tune T5 (Raffel et al., 2020) than using few-shot. Using generated gold data and real gold data in combination, they achieved state-of-the-art results on the SuperGLUE tasks (Wang et al., 2019).

In the construction of the Penn treebank, the authors use simple models to create initial syntactic parses which were then manually corrected (Marcus et al., 1993).

While in this paper we use an LLM for a particular case of semantic parsing, our study differs from prompt-based semantic parsing in that we do not intend to solve the task by prompting the LLM.It is also different from training data generation by prompting LLMs in that we do not use the LLM to generate synthetic data. It is similar to the approach taken by (Marcus et al., 1993), but we are not using heuristics or models pretrained for a particular task, but rather a generic large language model.

3 Preliminaries

3.1 Modelling of requirements

Klüwer and DNV GL (2019) proposed a logical framework for representing requirements using OWL 2 and description logic (DL) where a requirement is satisfied if and only if for every x that is a member of the class $\mathcal{S}$ and satisfies the condition $\mathcal{C}$ (which may be empty), x also satisfies the demand $\mathcal{D}$. The framework is appropriate for requirements because DL primarily deals with concepts rather than individuals. For an introduction to description logic see (Krötzsch et al., 2012). If $\mathcal{S}$, $\mathcal{C}$, and $\mathcal{D}$ are (possibly complex) ontological class expressions, the requirement can be expressed as:

$$\mathcal{S} \sqcap \mathcal{C} \sqsubseteq \mathcal{D} \tag{1}$$

This means that a thing that is an $\mathcal{S}$ needs to also be a $\mathcal{D}$ if it is $\mathcal{C}$. E.g., if something is a "steel pipe" $\mathcal{S}$ and it is "exposed to salt water" $\mathcal{C}$, it must have "corrosion protection" $\mathcal{D}$.

Ontological class expressions are either atomic classes, or expressions combining classes with conjunction $\sqcap$, disjunction $\sqcup$, negation $\neg$, or quantifiers with a property and a class expression (e.g., $\exists r.C$). We use square brackets after datatype to designate OWL 2 data ranges. E.g., $\exists$`hasSize.float[`$\geq$`50]` means that the concept has a `hasSize` relation to a float $f \in [50, \infty)$. We use expressions of the type $\exists$`hasDescription.string["a description"]` for expressions that are descriptive in nature or are either unnecessarily detailed or not expressible in DL.

The following requirement texts are taken from the document RU-Ship Pt4 Ch7 Sec 3 (Arrangements).[1] The DL statements are modelled by us.

Requirement [2.2.1] (sentence 2): *[. . .] the tank surfaces and bulkheads shall be insulated.*

$$\texttt{TankSurface} \sqcup \texttt{Bulkhead} \sqsubseteq \exists \texttt{hasFeature}.\texttt{Insulation}$$

[1] All documents are copyrighted ©DNV. DNV does not take responsibility for any consequences arising from the use of this content.

Requirement [2.2.2]: *Coamings for stairs, pipe openings, etc. shall be of ample height.*

```
Coaming ⊓ ∃usedFor.
  (Stair ⊔ PipeOpening)
⊑ ∃hasHeight.
  (PhysicalQuantitity
    ⊓ ∃hasDescription.
      string["of ample height"])
```

Requirement [3.1.1]: *The main inlet and main outlet pipes for thermal-oil at the fired heater and at the heater heated by exhaust gases shall have stop-valves, arranged for local manual and remote controlled operation from an easily accessible location outside the heater room.*

```
(MainInletPipe ⊔ MainOutletPipe)
⊓ ∃usedFor.ThermalOil
⊓ ∃connectedTo.(FiredHeater
  ⊔ ExhaustGasHeater)
⊑ ∃hasPart.(StopValve
  ⊓ ∃arrangedFor.ManualOperation
  ⊓ ∃arrangedFor.(RemoteOperation
    ⊓ ∃hasDescription.string["from an
      easily accessible location
      outside the heater room"])
```

3.2 Semantic parsing of requirements

To automatically find a logical representation of a sentence, we can use a semantic parser. In general, a semantic parser realizes a function $f : I \rightarrow O$ where the domain I is typically a set of utterances in natural language, such as in the form of sentences over an alphabet ($I \subseteq \Sigma^*$), and the co-domain O is the set of machine-readable representations that for some utterance express a subset of its meaning that is relevant for some task. The set of representations can be a language L generated via a grammar M, i.e., $L(M)$. For example, it can be the set of expressions in first-order logic over a predefined set of predicates P and class names C.

The functionality of the semantic parser will vary depending on the type of input, the logical formalism, and the needs of the particular application. Therefore, it is necessary to create a custom semantic parser for a new application and domain. In our case, the function $f : I \rightarrow O$ represents a mapping from a set I of textual requirements to the set of meanings expressed using description logic syntax as in Equation 1. One way to create a semantic parser is by fine-tuning a neural network pretrained on language generation, using models such as BART (Lewis et al., 2020) or T5 (Raffel et al., 2020). Training a neural network this way, however, requires a large annotated dataset, which can be very expensive to obtain.

3.3 A case study with GPT-3

Given the high cost of obtaining training data for semantic parsing in technical domains, we investigate the potential benefits of incorporating a large language model, specifically GPT-3, as part of human-computer collaboration, for constructing a gold standard dataset for semantic parsing of technical requirement sentences. Specifically, we want to find out the following: i) To what extent can a Hybrid Human-Machine collaborative annotation with GPT-3 reduce the effort needed for developing gold examples for semantic parsing as opposed to human annotation only? ii) Does using semantically similar requirements as examples improve effectiveness over random selection? iii) Will the ordering of the semantically similar requirement examples affect the effectiveness of the approach? iv) How does the number of examples influence the result? v) If we cluster the requirements and pick the most central requirement for each cluster, thus ensuring good coverage from the start, can that improve the performance a) over a random baseline, or b) over using the semantically most similar requirements?

4 Method

4.1 Corpus creation

To create the corpus, we obtained 2225 unlabelled requirement sentences from 23 PDF documents from DNV[2] that were accessible online[3] (see Table 3). To extract the text from the documents and create a semi-structured XML version of the PDF, we used Apache PDF box[4] and regular expressions. We limit our work to sentences containing the modal verb "shall," as DNV considers "shall" to be an indicator of a requirement (Det Norske Veritas, Ed. July 2022).

An annotation guideline was created and subsequently followed by the first author of the paper to produce the reference gold standard (RGS) consisting of 136 requirement sentences with a corresponding description logic formula. The second author of the paper verified the annotations to ensure the quality.

We do not make use of a predefined set of predicates or class names. However, by providing examples we implicitly specify the set of predicates and

[2]All documents are copyrighted ©DNV
[3]From https://rules.dnv.com/ 21.9.2022
[4]v2.0.1

the set of class names, so that a model could learn which class names and predicates are preferred.

4.2 Human-machine collaborative annotation

We propose a novel method for gold standard creation using a large language model together with a human expert. The approach involves iterating over a set of unlabelled requirement sentences (R) and generate a prompt p which consists of a brief task description (see Appendix A), n examples selected using a sample selection method (m), and the target sentence s. The examples are on the form:

```
Input: [requirement sentence]
Output: [logical representation]
```

We use a large language model, specifically GPT-3, to generate an initial semantic parse (r') for the target sentence. Subsequently, a human expert reviews and corrects the model's output to ensure accuracy and consistency (r''). The gold standard (G), which is initially empty, is extended with (s, r''). This iterative process continues until all examples are annotated, resulting in a complete gold standard. The process is outlined in Algorithm 1. If n exceeds the size of the set G ($n > size(G)$), we are unable to select n samples. In such cases, we utilize all the samples in G if G is non-empty, or non at all if G is empty.

Algorithm 1 Creating a gold standard

procedure CREATEGOLDSTANDARD(R, m)
  $G \leftarrow \varnothing$
  for $s \in R$ **do**
    $p \leftarrow \mathsf{createPrompt}(s, m, G)$
    $r' \leftarrow \mathsf{GPT}(p)$
    $r'' \leftarrow \mathsf{humanImprovement}(s, r')$
    $G \leftarrow G \cup \{(s, r'')\}$
  end for
return G
end procedure

The initial task description is part of all prompts. The samples, however, may be different for each target sentence. We use three general sample selection methods from the growing gold standard. The first general sample selection method (RandomN) is to randomly select n examples for each target sentence. To investigate how the number of examples in the prompt influences the quality of GPT-3's answer, we perform four experiments using this method, where n is 5, 10, 20, and 30, respectively. Since Random20, Clustering, and the MostSimilar requirements have the same number of examples, the Random20 can also serve as a baseline for the other sample selection methods.

The second general sample selection method (MostSimilar) is to use the n requirement sentences that are most semantically similar to the target sentence. To embed the sentences, we use the RoBERTa-large model from the sentence transformer library (Reimers and Gurevych, 2019) in Huggingface[5]. For each sentence s' in G, we calculate the cosine similarity between s' and the target sentence s. The sentences are sorted with the most semantically similar sentences first before we select the $k = 20$ most similar sentences. To investigate the impact of the order of the examples, we perform three experiments using this method, MostSimilarRandom, where the order of the n examples is randomized. MostSimilarFirst where we keep the original order of the n most similar sentences, and MostSimilarLast, where we sort the n most similar requirements from the least to the most similar.

The third general sample selection method (Clustering) is to use a fixed set of diverse requirements that ensure good coverage of topics. We used the KMeans clustering implementation in scikit-learn[6]. From each cluster k, we choose the data point that is closest to the cluster centroid. This gives us 20 sentences that, used as part of the prompt, will ensure high coverage of different types of requirements. This method will allow us to see if aiming for good coverage of different examples is better than random selection (RandomN) or selecting the most semantically similar sentences (MostSimilar). In the Clustering sample selection method, we label the sentences from the 20 clusters first.

4.3 Metrics

We estimate the effort, denoted by δ, of a human annotator to correct the logical representation with three metrics.

String Edit Distance Levenshtein Distance measures string similarity by counting the shortest edit sequence to transform one string into another. To compare DL formulas, however, we need a distance metric that considers their structure, thus we use a string edit distance metric that operates on the level of DL terms, operators, and individual string

[5] https://huggingface.co/sentence-transformers/all-roberta-large-v1

[6] v1.0.2

tokens counting the minimum number of insertions, substitutions, deletions, and transpositions.

For example, the difference between `Boiler` $\sqsubseteq$ $\exists$`hasFeature.Insulation` and `Compressor` $\sqsubseteq$ $\exists$`hasFeature.Insulation` is 1.

If the string edit distance exceeds the costs of turning an empty string into the reference parse, we return the edit distance of turning an empty string into the reference parse. This is reasonable because a human would discard a parse that would take more effort to correct than to create it from scratch.

Graph Edit Distance The string edit distance does not take into account that some binary operators, like conjunction and disjunction, are associative. For instance, the string edit distance between `A` $\sqcap$ `B` and `B` $\sqcap$ `A` is 2, even though the formulas are logically equivalent. To address this issue, we also use graph edit distance between the two DL formulas. Graph edit distance computes the minimum number of edits required to transform one graph g' into a graph isomorphic to another graph g.

We parse the DL formula and transform it into a graph with terms on the nodes and the edges representing the relationships between the nodes. For the axiom ($\sqsubseteq$), we attach numeric labels to the edges because changing the order of the edges would change the meaning of the axiom. For the unary and binary operators (conjunction and disjunction where the order of the operands is not relevant), we do not add labels for the edges. An example of the graph structure is given in Figure 1.

We use the following operations to compute graph edit distance: node insertion, node deletion, node substitution, edge insertion, edge deletion, and edge substitution (in the case the edge has a label). The cost of each operation is set to 1. Like the string edit distance, if the graph edit distance exceeds the cost of turning a graph containing only one node with the $\sqsubseteq$ symbol into the graph of the reference parse, we return the edit distance between the graph of the reference parse and the graph containing only one node with the $\sqsubseteq$ symbol (used to align the graphs).

Computing the graph edit distance is an NP-hard problem (Zeng et al., 2009). Therefore we use a timeout of 20 seconds and return the best result. If no result was found within the timeout, we assume the distance is high, and use the maximum distance instead.

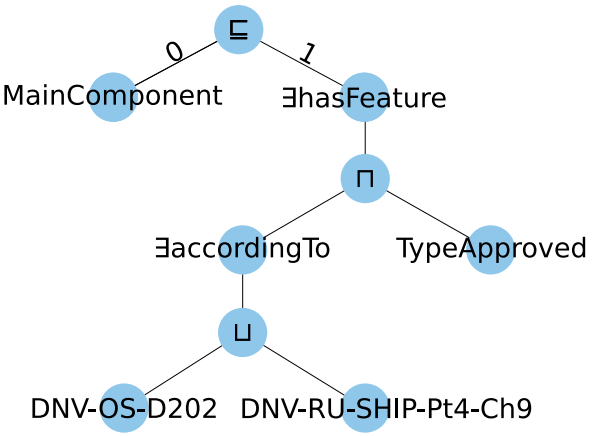

Figure 1: DL graph used for graph edit distance. Representing: `MainComponent` $\sqsubseteq$ $\exists$`hasFeature.(TypeApproved` $\sqcap$ $\exists$`accordingTo.(DNV-OS-D202` $\sqcup$ `DNV-RU-SHIP-Pt4-Ch9))`.

Jaccard distance Furthermore, to say something about how similar the terms, operators, and tokens in the GPT-3 parse are to the reference gold standard, we use Jaccard similarity. Jaccard similarity is the fraction of items shared between two sets to the union of the items in the sets two sets, and Jaccard distance is the complement of Jaccard similarity. We split the parse proposed by GPT-3 and the reference parse into their individual DL tokens, operators, and string tokens, remove duplicates, and calculate the Jaccard distance between the two.

4.4 Experimental setup

Hyperparameters For all the experiments, we use the model `text-davinci-3`.The temperature was set to 0 to eliminate randomness. We request the model to only return the most probable parse. Max token was set to 256, and the newline character was used as a stop symbol.

Experiments To quantify the effort for a human annotator to create a logical representation from scratch, without receiving anything proposed by GPT-3, we use *i*) Empty, where instead of a proposal from GPT-3, we use an empty string. To investigate to what extent the prompt (p) affects the effectiveness of the approach, and to answer the questions stated in Section 3.3, we use the following methods for choosing which examples to include in p (the sample selection methods are described in detail in Section 4.2). *ii*) Random5, *iii*) Random10, *iv*) Random20, *v*) Random30, *vi*) MostSimilarRandom, *vii*) MostSimilarFirst, *viii*) MostSimilarLast. All the experiments that

involve random sampling or ordering were run five times, and we report average values and the standard deviation.

We follow Algorithm 1 for each of the sample selection methods described above and for each of the 136 requirement sentences. We start with an empty list as the gold standard G from which we choose examples. G is incrementally extended as we prompt GPT-3 with new sentences. For each of the experiments not using the Clustering method, the order of the target sentences was the same. In the experiment using the Clustering method, the 20 central concepts from the clusters were used first, and the rest of the target sentences were used in the same order.

Evaluation To estimate the difference in human effort with and without the proposals by GPT-3, we evaluate the prediction by GPT-3 against the reference gold standard (RGS) with the metrics introduced in Section 4.3. For each requirement sentence s and reference parse r, we compute the difference between r and the parse generated by GPT-3 (r') with string edit distance (δ^s), Jaccard distance (δ^j), and graph edit distance (δ^g). Δ^s, Δ^j, and Δ^g are the sum of the string edit distances, Jaccard distances, and graph edit distances, respectively. The evaluation procedure is outlined in Algorithm 2. Note that, since we are evaluating against RGS, we extend G with the reference parse r directly.

Algorithm 2 Evaluation

procedure EVALUATE(RGS, m)
  $(\Delta^{\mathtt{s}}, \Delta^{\mathtt{j}}, \Delta^{\mathtt{g}}) = (0, 0, 0)$
  $G \leftarrow \varnothing$
  for $(s, r) \in RGS$ **do**
    $p \leftarrow \mathsf{createPrompt}(s, m, G)$
    $r' \leftarrow \mathsf{GPT}(p)$
    $\Delta^{\mathtt{s}} \leftarrow \Delta^{\mathtt{s}} + \delta^{\mathtt{s}}(r, r')$
    $\Delta^{\mathtt{j}} \leftarrow \Delta^{\mathtt{j}} + \delta^{\mathtt{j}}(r, r')$
    $\Delta^{\mathtt{g}} \leftarrow \Delta^{\mathtt{g}} + \delta^{\mathtt{g}}(r, r')$
    $G \leftarrow G \cup (s, r)$
  end for
return $(\Delta^{\mathtt{s}}, \Delta^{\mathtt{j}}, \Delta^{\mathtt{g}})$
end procedure

5 Results

We sum all the string edit distances, graph edit distances, and Jaccard distance, and report the totals and averages from the experiments described in Section 4.4 in Table 1.

GPT-3-assisted annotation The experiment using the empty string (Empty) gives a total string edit distance of $2,573$ edits. The best-performing sample selection method uses $1,506$ edits. This gives us a difference of 1067 edits. For graph edit distance, the numbers are $2,692$ and $1,681$, a reduction of 1011 edits. The average Jaccard distance decreases from 1 to 0.52.

The edit distance metrics depend on the size of the formula; short parses can have at most small edit distances, while long parses can have large edit distances. Therefore, to be able to observe a trend over time, we need to factor out the size of the formula. Consequently, we normalize the sting edit distance by dividing the number of edits by the number of tokens in the correct parse. A normalized edit distance of 1 indicates that the entire formula needs to be changed. Although the metric shows much variation, we can observe a downward trend in string edit distance from the first to the last target sentence (see Figure 2). This trend is also visible for graph edit distance, as shown in Figure 3. Similarly, in Figure 4, we can see a comparable trend for Jaccard distance.

Sample selection methods We found that the MostSimilarFirst sample selection method obtained the shortest distance on all metrics. Specifically, it achieved a total string distance of 1,506, a total graph edit distance of 1,681, and an average Jaccard distance of 0.52. The MostSimilarLast method, however, was found to perform worse than the random ordering of the most similar examples on average.

All the experiments with the MostSimilar method yielded smaller string edit distances, graph edit distances, and Jaccard distances than the experiments with RandomN. The experiment with the Clustering method, however, obtained a better string edit distance than the experiment with the MostSimilarLast method, while MostSimlarLast performed better on the other metrics. The experiment with the Clustering method has a smaller string edit distance and graph edit distance than all the experiments with RandomN on average. The experiments with Random20 and Random30 performed better than the experiment with Clustering on Jaccard distance. The experiment with Random30 was better than all the other experiments

with RandomN on average, and the experiment with Random5 obtained the largest distance on all the metrics on average.

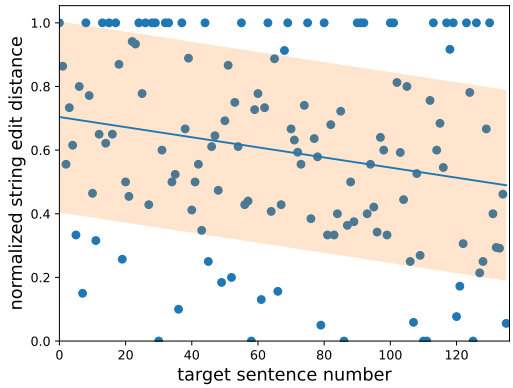

Figure 2: Normalized string edit distance from the first to the last target sentence using the MostSimilarFirst method.

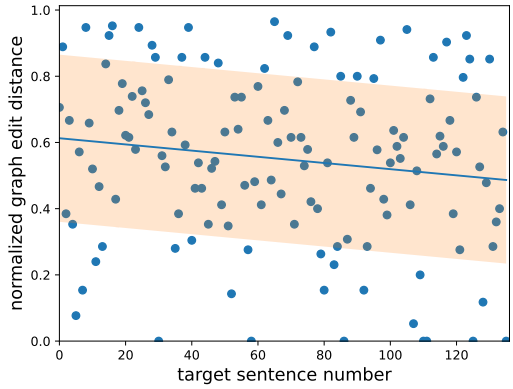

Figure 3: Normalized graph edit distance from the first to the last target sentence using the MostSimilarFirst method.

5.1 Examples of GPT-3 mistakes

First, we discuss the different types of errors we encounter. Often, multiple errors occur in one parse provided by GPT-3. Furthermore, we analyzed the frequencies of these errors on the same 20 requirements using three sample selection methods: Clustering, Random20, and MostSimilarLast. We randomly selected one of the experiments with Random20 for this analysis, and the error counts are presented in Table 2.

i) **Wrong DL syntax** Although rare, this type of error typically affects the first one or two sentences. Examples include the use of variables (e.g., $\exists$`c.Component` $\sqsubseteq$ `Type-Approved(c)`) and multiple subclass axioms ($A \sqsubseteq B \sqsubseteq C$), neither of which is permitted in DL. We observed syntax mistakes both in Clustering and Random20.

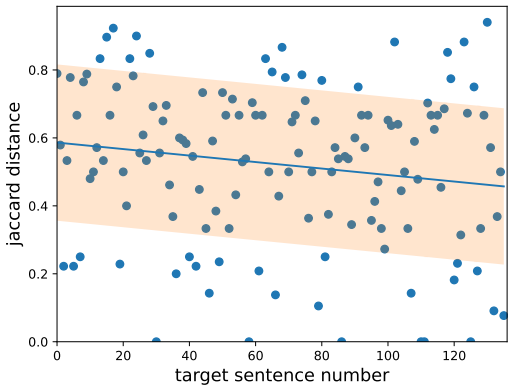

Figure 4: Jaccard distance from the first to the last target sentence using the MostSimilarFirst method.

ii) **Different modelling choice** This type of error is not necessarily incorrect, but it affects the edit distance metrics. Different modelling choice is relatively frequent and takes many forms, such as using a concept as a property instead of a class or breaking down a requirement differently than we do, but in a plausible way. For example, we model accordance as `∃inAccordanceWith...`, but we have observed instances where GPT-3 models it as `∃fitted.(InAccordanceWith...)`.

iii) **Element on the wrong side of axiom** Another common type of modelling mistake made by GPT-3 is to model a condition as a mandatory feature or create an axiom where the left side is not what the requirement is about. In these cases, the proposed axiom is often substantially different from the reference gold standard. For example, if a requirement says *There shall be a portable foam applicator in each boiler room*, modelling `PortableFoamApplicator` $\sqsubseteq$ `∃hasLocation.BoilerRoom` would be incorrect as it implies that all portable foam applicators must be located in boiler rooms.

iv) **Too much or too little information as a string** Another type of mistake is including either too much or too little information

Method	String Distance		Graph Distance		Jaccard	
	Δ	μ	Δ	μ	Δ	μ
Empty	2,573	18.92	2,692	19.79	135.49	1.00
Clustering	1,640	12.06	1,821	13.39	80.18	0.59
Random5	1,809±47	13.30±0.34	1,909±21	14.04±0.15	84.45±1.74	0.62±0.01
Random10	1,744±42	12.82±0.31	1,903±29	13.99±0.21	80.94±1.86	0.60±0.01
Random20	1,724±17	12.68±0.13	1,843±1	13.55±0.01	79.77±1.00	0.59±0.01
Random30	1,683±27	12.38±0.20	1,848±29	13.59±0.22	77.89±1.39	0.57±0.01
MostSimilarRandom	1,569±29	11.54±0.21	1,759±25	12.93±0.18	72.64±1.47	0.53±0.01
MostSimilarFirst	**1,506**	**11.07**	**1,681**	**12.36**	**70.96**	**0.52**
MostSimilarLast	1,658	12.19	1,748	12.85	74.45	0.55

Table 1: The sum (Δ) and average (μ) values of edit distance, graph edit distance, and Jaccard distance for each of the experiments. For the RandomN experiments, we show the average of 5 runs with one standard deviation.

in a `hasDescription.String[]`. In some cases, GPT-3 may provide redundant information or use this construct for things that are easy to express in DL. In other cases, it may try to model something using DL that is not possible. We found this mistake to be most frequent in Random20, and least frequent in the experiment with the Clustering method.

v) **Different terminology (plausible)** The use of different terminology is another common mistake, which can include synonyms, spelling differences, or using the plural instead of a single form, compared to the reference gold standard. For instance, `Fail-SafeFunctionality` instead of `FailSafeFunctionality`, `NewDesigns` instead of `NewDesign` are simple differences in spelling, and `Emergency` may be as good as `StateOfEmergency`. While similar terms could be interchangeable, they are all counted as equally different using our metrics. This type of error was found to be less frequent using the MostSimilarFirst method and the Random20 method and most frequent in the experiment with the Clustering method.

vi) **Different terminology (not plausible)** Generating very long and complicated concepts or properties like `Within3MetersFromHazardousAreas` and `TwoIndependentAlternativesForPressurization` is another mistake GPT-3 makes. Although these names may be technically correct, they are unlikely to be found in a typical ontology. Instead of trying to break down complex concepts into more atomic ones, GPT-3 captures everything in a single concept or property. This type of mistake was found to be most frequent in the experiment with Clustering, and least frequent with MostSimilarFirst.

vii) **Confusing disjunction and conjuction** Another mistake GPT-3 makes is confusing conjunction and disjunction. This often occurs when using only one feature relation instead of multiple. For example, GPT-3 may model the requirement of having the two features A and B using a disjunction, as in $\exists$`.r(A` $\sqcup$ `B)`. However, the correct representation should use a conjunction, as in $\exists$`r.A` $\sqcap$ $\exists$`r.B`. This mistake was found to be most frequent in the experiment using the Clustering method.

viii) **Missing or extra elements/clauses** Adding too much information or missing important details are also mistakes seen in GPT-3's parses. For instance, GPT-3 may add explanations and reasons behind a requirement, even though they are not needed in our framework. It may also miss some important details.

6 Discussion

GPT-3 assisted annotation The difference between creating the 136 parses from scratch and with the help of the best method using GPT-3 is 1067 edits, a reduction of the effort of about 41% in the number of string edits. For graph edit distance the reduction is 1011 edits, about 38 %. This shows that the method is effectively reducing the human effort of creating the gold standard. Figures 2 and 3 indicate that the accuracy of the parses improves with more examples in the gold standard.

Considering Jaccard distance, we observe that, on average, there are differences between 52 % of

Method	i) Wrong DL syntx	ii) Different modelling	iii) Wrong side of axiom	iv) Too much/little in string	v) Different terms (plausible)	vi) Different terms (implausible)	vii) Confusing $\sqcap$ and $\sqcup$	$viii$) More/less elements
MostSimilarFirst	0	6	3	7	8	2	2	7
Clustering	1	6	3	2	11	4	4	8
Random20	1	6	3	8	8	3	2	9

Table 2: Counts of different GPT-3 mistakes on the same 20 sentences. i) is wrong DL syntax, ii) different modelling choice, iii) element on the wrong side of axiom, iv) too much or too little information as a string, v) different terminology (plausible), vi) different terminology (not plausible), vii) Confusing disjunction and conjunction, $viii$) missing or extra clauses or elements.

the terms, symbols, and tokens. Hence, there is an overlap of 48 % between the terms in the predicted parses and the reference parses. This distance also decreases with more examples.

To create a correct formula from scratch, one needs more than just to write down the components, one has to identify good terms (the correct terms) to express this in a logical format and then structure it correctly. If we have many of the correct terms and parts of the structure, this is already helpful.

The evaluation metrics do not take into account the lexical and semantic similarity of DL terms. The metrics will, for example, regard a term as wrong if the term was written in plural form instead of in singular form. This is, however, easy to correct as opposed to identifying and using a new term. It may also be easier to substitute a semantically similar term with another if the annotator knows which is the correct one. Edit distance can also overestimate the human effort of deleting a series of tokens in a `∃hasDescription.String[]`-construct. If the model suggests making a long string literal which should not be included, it requires deleting multiple tokens, while a human can typically do this in one operation. If, however, both the proposed parse and the parse in the reference gold standard contain such a string literal, then the deletion of individual tokens would correspond to the actual effort.

Hence, we argue that string edit distance, graph edit distance, and Jaccard distance overestimate the human effort because to change a term into something completely different is more effortful than to change spelling or use a synonym. However, our metrics treat all changes as equally different. As seen in GPT-3 mistake v) in Section 5.1, many of the mistakes with terms involve substituting plausible but incorrect terms.

Sample selection methods As expected, we find that selecting the examples that are most semantically similar to the target sentence is the most effective strategy which is confirmed by all the metrics. We also find that the ordering impacts performance, which is consistent with the results presented in (Liu et al., 2022). Specifically, we find that ordering the examples with the most semantically similar examples first achieved the best results. The Clustering method also yields better results than random sampling similar to what was found by (Chang et al., 2021). All sample selection methods, however, yield a reduction in the work needed to create the gold standard.

With the MostSimilarFirst method, the Jaccard distance was found to decline over time (see Figure 4). This trend can be attributed to the fact that as we accumulate more examples and consequently have access to more examples with similar topics and terms to the target sentence, the model will be increasingly exposed to sentences with similar terms and how these terms are represented in the DL parses. Our error counts support the observation that when creating the prompt using the MostSimilarFirst method it produces fewer terminology-related mistakes, indicating a better understanding of the DL vocabulary.

7 Conclusion

In our study, we propose a systematic approach to gold standard creation based on the concept of Human-Machine collaborative annotation. To evaluate the effectiveness of our approach, we con-

ducted a case study on a small corpus of industry requirements. Our results indicate that the best method reduced the annotation effort over manual annotation by about 41 % and 38 % using the string edit distance, and graph edit distance respectively. We argue that the actual reduction in effort is even greater, as the metrics we use overestimate the effort required to correct terms.

In our study, we find that selecting the semantically most similar requirements as examples and ordering them with the most similar example first was most effective. Additionally, we found that using 30 examples was better, on average, than 5, 10, and 20. It is worth noting, however, that the effectiveness of the model depends more on which examples it sees than the number of examples, demonstrated by the fact that both Clustering and MostSimilar resulted in fewer edits than all the experiments with RandomN even Random30, which use more examples.

8 Limitations and future work

Limitations The metrics we use to estimate the human effort to correct an initial parse, i.e., string edit distance, graph edit distance, and Jaccard distance, all assume that each operator, term, and token are equally difficult to change and thus overestimate the real effort as discussed in Section 6. The distance is measured between the parse proposed by the LLM and the parse in the reference gold standard. However, as there may exist multiple ways to represent one and the same requirement, it is possible that the proposed parse is equally valid as the reference parse, but simply on a different form. A human annotator could have accepted this parse (with or without modifications), however, our metrics are unable to capture such cases.

We were not able to measure how the approach affects the actual time it takes for a human to create the parses from scratch as opposed to correct the proposals by the LLM. This would have been a better measure than edit distance measures and Jaccard distance. To be able to estimate the actual time it takes for a human to create the parses, we would have needed to conduct all the experiments several times with multiple domain experts doing the corrections (to account for individual differences), something we did not have access to.

In addition, creating a consistent reference gold standard was challenging due to the many different topics and the lack of an ontology to ensure consistent modelling of terms and constructs. The possibility of modelling the same requirements in different ways further complicated the process. Using a more narrow domain or having access to a concrete ontology and application could have facilitated the creation of the reference gold standard. In the future, however, we want to use our approach to create a gold standard for a real application.

Since this is a case study, we have focused on only one language model. However, it is important to notice that other models are likely to demonstrate different performances. Furthermore, we could have compared how a human subject performs compared to a language model on the task. It is possible that human performance also is suboptimal.

Moreover, one may argue that a wrongly parsed requirement by GPT-3 may mislead the human annotator into creating a parse that is incorrect but looks plausible. It is, therefore, important to have annotators with both domain and modelling knowledge. To see if this is the case, one would have to have several groups of people annotate the same requirements with and without collaboration with GPT-3.

Future work It would be interesting to carry out similar studies with existing semantic parsing datasets and compare how the performance on this particular dataset differs from standard datasets. Working with several models and several datasets could provide insight into how effective this method is for gold standard creation for semantic parsing in general, and how the domain specificity affects the effectiveness in particular.

Another interesting direction for future work is to explore the possibility of including an existing vocabulary as part of the prompt. Since many of the mistakes come from using incorrect vocabulary or different concept breakdowns than the one proposed in the reference gold standard, a two-phase prompting approach, where one can make use of vocabulary from an existing ontology, could improve the performance of the method.

Finally, the correct understanding of a requirement often relies on factors such as domain knowledge, the surrounding context and the interplay with other requirements. Therefore, taking into account larger structures, such as paragraphs, sections or entire documents can provide essential information that could enhance parsing accuracy.

9 Acknowledgement

The research is funded by the SIRIUS centre[7]: Norwegian Research Council project number 237898. It is co-funded by partner companies, including DNV. We thank the reviewers for their valuable feedback.

References

Ernie Chang, Xiaoyu Shen, Hui-Syuan Yeh, and Vera Demberg. 2021. On training instance selection for few-shot neural text generation. In *ACL 21 / IJCNLP 21 (Volume 2: Short Papers)*, pages 8–13.

Det Norske Veritas. Ed. July 2022. RULES FOR CLASSIFICATION: Ships. Technical report, DNV-RUSHIP. ©DNV GL.

Jacob Devlin, Ming-Wei Chang, et al. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Adi Haviv, Jonathan Berant, and Amir Globerson. 2021. BERTese: Learning to speak to BERT. In *EACL 2021: Main Volume*, pages 3618–3623, Online.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know? *TACL*, 8:423–438.

Johan W Klüwer and DNV GL. 2019. OWL Upper Ontology for Reified Requirements. `https://data.dnv.com/ontology/requirement-ontology/documentation/req-ont.pdf` accessed: 2023-01-13.

Markus Krötzsch, Frantisek Simancik, and Ian Horrocks. 2012. A description logic primer. *arXiv:1201.4089*.

Mike Lewis, Yinhan Liu, et al. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL 2020*, pages 7871–7880.

Jiachang Liu, Dinghan Shen, et al. 2022. What makes good in-context examples for GPT-3? In *DeeLIO 2022 Workshop*, pages 100–114.

Pengfei Liu, Weizhe Yuan, et al. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

Yinhan Liu, Myle Ott, et al. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv:1907.11692*.

[7] `http://sirius-labs.no`

Yao Lu, Max Bartolo, et al. 2022. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *ACL 2022 (Volume 1: Long Papers)*, pages 8086–8098.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Fabio Petroni, Tim Rocktäschel, et al. 2019. Language models as knowledge bases? In *EMNLP-IJCNLP 2019*, pages 2463–2473.

Colin Raffel, Noam Shazeer, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21(1):5485–5551.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *EMNLP-IJCNLP 2019*, pages 3982–3992.

Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. How much knowledge can you pack into the parameters of a language model? In *EMNLP 2020*, pages 5418–5426.

Subendhu Rongali, Konstantine Arkoudas, Melanie Rubino, et al. 2022. Training naturalized semantic parsers with very little data. *arXiv:2204.14243*.

Subhro Roy, Sam Thomson, et al. 2022. Benchclamp: A benchmark for evaluating language models on semantic parsing. *arXiv:2206.10668*.

Nathan Schucher, Siva Reddy, and Harm de Vries. 2022. The power of prompt tuning for low-resource semantic parsing. In *ACL 2022 (Volume 2: Short Papers)*, pages 148–156.

Richard Shin, Christopher Lin, et al. 2021. Constrained Language Models Yield Few-Shot Semantic Parsers. In *EMNLP 2021*, pages 7699–7715.

Ashish Vaswani, Noam Shazeer, et al. 2017. Attention is all you need. *NIPS 2017*, 30.

Alex Wang, Yada Pruksachatkun, et al. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *NeurIPS 2019*, 32.

Zirui Wang, Adams Wei Yu, Orhan Firat, and Yuan Cao. 2021. Towards zero-label language learning. *arXiv:2109.09193*.

Jingfeng Yang, Haoming Jiang, et al. 2022a. SEQZERO: Few-shot compositional semantic parsing with sequential prompts and zero-shot models. In *NAACL 2022*, pages 49–60.

Kevin Yang, Olivia Deng, et al. 2022b. Addressing resource and privacy constraints in semantic parsing through data augmentation. In *ACL 2022*, pages 3685–3695.

Zhiping Zeng, Anthony KH Tung, et al. 2009. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36.

A Prompt

We used the following fixed prompt with GPT-3.

```
Below are some inputs and the outputs of a semantic parser of industry standards. It always transforms a sentence into its correct corresponding logical representation. The input is a requirement from an industry standard. The output is a logical representation in description logic (DL) format. The output represents classes, properties, individuals and restrictions. The symbols used in the DL syntax are: ∃, ⊑, ⊓, ⊔, and ¬. On the left-hand side of the ⊑ is most often a physical object and possibly a condition on the object. The right-hand side of the ⊑ is what is demanded of the object on the left side.
```

B Examples of modelling by GPT-3

DNV-RU-HSLC-Pt4 [1.2.2] (sentence 2) *Flexible pipes shall have suitable connections, be resistant to salt, water, oil and vibration, be visible, easily accessible and are not to penetrate watertight bulkheads.* GPT-3 proposes

```
FlexiblePipe ⊑
∃hasFeature.(SuitableConnection
  ⊓ ResistanceToSalt
  ⊓ ResistanceToWater
  ⊓ ResistanceToOil
  ⊓ ResistanceToVibration
  ⊓ Visible
  ⊓ EasilyAccessible)
⊓ ¬∃permits.(PenetrateWatertightBulkhead)
```

There are some problems with this. First, this formula states that the FlexiblePipe shall have one of these features, but the requirement states that it needs to have all the features. In addition, GPT-3 changes *resistant to water* to `ResistanceToWater`, while being a reasonable proposal it is different from the reference parse, thus it affects distance.

The reference parse is:

```
FlexiblePipe ⊑
∃hasFeature.SuitableConnection
⊓ ∃hasFeature.ResistantToSalt
⊓ ∃hasFeature.ResistantToWater
⊓ ∃hasFeature.ResistantToOil
⊓ ∃hasFeature.ResistantToVibration
⊓ ∃hasFeature.Visible
⊓ ∃hasFeature.EasilyAccessible
⊓ ¬∃permits.
  PenetrateWatertightBulkheads
```

This formula comes out with a string edit distance of 19 and a graph edit distance of 30, a Jaccard distance of 0.5. The edit distance is high, but in reality, this formula helps the annotator much as the basic structure is there. One has to remove the parenthesis, and add `⊓ ∃hasFeature.` before each of the features, change the spelling from `Resistance` to `Resistant` (and similar for the others).

DNV-RU-HSLC-Pt4 [1.5.1] (sentence 1) *Fuel with a flashpoint below 35°C shall not be used.* In this example, GPT-3 proposes the same as the reference parse:

```
Fuel ⊑
¬∃hasFlashpoint.(PhysicalQuantity
  ⊓ ∃hasValue.float[<35]
  ⊓ ∃hasUnit.string['C'])
```

DNV-OS-C103 [1.3.7] (sentence 1) *For new designs, and/or unproved design applications of designs where limited or no direct experience exists, relevant analyses and model testing, shall be performed in order to demonstrate that an acceptable level of safety is obtained.* GPT-3 proposes

```
NewDesigns ⊔ UnprovedDesignApplications
⊑ ∃hasFeature.(RelevantAnalyses
  ⊔ ModelTesting)
  ⊓ ∃permits.AcceptableLevelOfSafety
```

The reference parse is:

```
∃hasFeature.(NewDesign
  ⊔ (Design ⊓ ∃hasFeature.
    (LimitedExperience ⊔ NoExperience)))
⊑ ∃hasFeature.RelevantAnalysis
⊓ ∃hasFeature.ModelTesting
⊓ ∃permits.AcceptableLevelOfSafety
```

This solution gives a string edit distance of 14, a graph edit of 20, and a Jaccard distance of 0.5. Here we observe that GPT-3 has broken down the requirement differently from what the reference parse does. It puts the demand on the concept `NewDesigns ⊔ UnprovedDesignApplications`. We consider, however, that the requirement is not so much about the design, but the object that is being designed. On the right side of ⊑, it requires only one feature for something that is either a relevant analysis or a model testing, which is wrong. It should be two (different) features. The use of plural in NewDesigns, RelevantAnalyses is easy to correct, but affects the edit distances and Jaccard distance.

C Documents

Document code	Name
DNV-CG-0051	Non-destructive testing (January 2022)
DNV-CP-0231	Cyber security capabilities of systems and components (September 2021)
DNV-CP-0507	System and software engineering (September 2021)
DNV-OS-A101	Safety principles and arrangements (July 2019/August 2021)
DNV-OS-C101	Design of offshore steel structures, general - LRFD method (July 2019/August 2021)
DNV-OS-C102	Structural design of offshore ship-shaped and cylindrical units (July 2020/August 2021)
DNV-OS-C103	Structural design of column stabilised units - LRFD method (July 2020/August 2021)
DNV-OS-D101	Marine and machinery systems and equipment (July 2021)
DNV-OS-D201	Electrical installations (July 2022)
DNV-OS-D202	Automation, safety and telecommunication systems (July 2019/August 2021)
DNV-OS-D301	Fire protection (July 2019/August 2021)
DNV-OS-E301	Position mooring (July 2021)
DNV-OS-E402	Diving systems (July 2019/August 2021)
DNV-RU-HSLC-Pt3	High speed and light craft Part 3 Structures, equipment (August 2021)
DNV-RU-HSLC-Pt4	High speed and light craft Part 4 Systems and components (July 2022)
DNV-RU-NAVAL-Pt3	Naval vessels Part 3 Surface Ships (December 2015)
DNV-RU-NAVAL-Pt4	Naval vessels Part 4 Sub-surface ships (January 2018)
DNV-RU-NAV-Pt7	Naval vessels Part 7 Fleet in service (July 2022)
DNV-RU-OU-0101	Offshore drilling and support units
DNV-RU-OU-0104	Self-elevating units, including wind turbine installation units and liftboats (July 2022)
DNV-RU-SHIP-Pt4	Ships Part 4 Systems and components (July 2021)
DNV-SI-0166	Verification for compliance with Norwegian shelf regulations (January 2022)
DNV-ST-0111	Assessment of station keeping capability of dynamic positioning vessels (December 2021)

Table 3: The documents used in this study