From Paperwork to Rocket Science: AI-Based Tools for the Next Generation of Space Standardisation

RYAN OTT, Universiteit van Amsterdam (UvA) and European Space Agency (ESA)

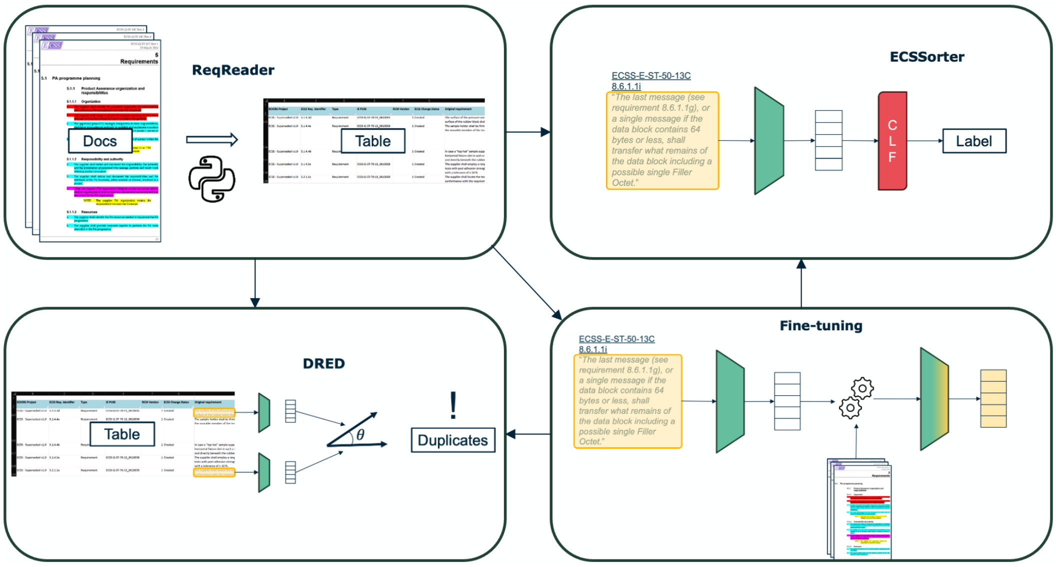
ReqReader
Docs
Table
ECSSorter
ECSS-E-ST-50-13C 8.6.1.1i
"The last message (see requirement 8.6.1.1g), or a single message if the data block contains 64 bytes or less, shall transfer what remains of the data block including a possible single Filler Octet."
C L F
Label
DRED
Table
θ
!
Duplicates
Fine-tuning
ECSS-E-ST-50-13C 8.6.1.1i
"The last message (see requirement 8.6.1.1g), or a single message if the data block contains 64 bytes or less, shall transfer what remains of the data block including a possible single Filler Octet."

Fig. 1. The four tools that were created as part of this project. A tool to read requirements from a directory of documents (top left). A text classification tool (top right). A duplicate requirement detector (bottom left). A fine-tuned text embedding model (bottom right). In combination, all parts form a pipeline to extract data, train and perform downstream tasks.

This paper presents the tools and experiments developed to assist the ECSS NextGen initiative in streamlining the space standards review process. Developed were the ReqReader for extracting and summarising requirements, the DRED for duplicate detection, and the ECSSorter for classifying requirements. These tools demonstrated a significant reduction the time and effort required for reviewing and updating standards, seeing quick adoption, and illustrating the benefits of integrating AI into traditionally manual processes. An investigation into domain-specific fine-tuning was also conducted, which although proving futile, provides insight into possible directions future AI endeavours under the ECSS should pursue.

1 Introduction

The European Space Agency (ESA) is an intergovernmental organisation that strives to shape the development of Europe's space capabilities and promotes the peaceful exploration of space and related technologies. The European Space Research and Technology Centre (ESTEC), located in Noordwijk, the Netherlands, serves as the technical heart of ESA, focusing on satellite testing, research, and development of advanced space technologies, as well as acting as the secretariat of the European Coordination for Space Standardisation (ECSS).

The ECSS defines and maintains standards that guide the production, assurance, and operation of space components across the space industry. It plays a critical role in ensuring the consistency and quality of European space missions by standardising technical and procedural requirements.

Author's Contact Information: Ryan Ott, ryan.ott@student.uva.nl, Universiteit van Amsterdam (UvA) and European Space Agency (ESA).

However, with the rapid pace of technological advancements and the rise of competitive private incumbents like SpaceX, these standards require modernisation to continue to meet the evolving needs of the space industry.

The ECSS NextGen project, launched in September 2024, is an initiative to update these standards to suit the needs of the upcoming decades of space exploration. It's focus is to reduce bureaucratic overhead, costs and to make the standards simpler to use. Key aspects of ECSS NextGen include simplifying requirements, identifying obsolete ones, and creating a digital platform that will make tailoring and accessing standards more intuitive for stakeholders.

Given the complexity of managing thousands of requirements, manually updating and refining them is highly cumbersome. This is where artificial intelligence (AI) tools can play a pivotal role. State-of-the-art techniques in Natural Language Processing (NLP) and neural language models have seen widespread use, and success, in a wide range of domains, especially since the emergence of transformer-based model architectures in 2017 [7].

The aim of this work was therefore to assist the ECSS NextGen initiative by leveraging AI to simplify and automate parts of the ECSS standards review process. Specifically, models were developed to achieve the following objectives:

- Detection and handling of duplicate requirements.
- Classification of requirements based on relevance and technical content.
- Enhancing semantic representation of requirements through domain-specific language embedding model fine-tuning.

In addition, to create practical datasets to support the ECSS NextGen working groups and enable model training, tools were created to automatically extract and consolidate requirements from pools of ECSS standards documents.

The models and tools created as part of this project were intended to not only streamline existing workflows, but the insights gained from this research should also act as foundation and guidance for future work as part of the ECSS NextGen initiative. This report presents a summary of the project objectives, the technical implementation, and the outcomes achieved, along with an assessment of the tools created and their effectiveness regarding ESA's operational requirements.

2 Background

The related work section provides a detailed overview of the existing technologies and methodologies that informed the development of the tools created as part of this work. This includes a focus on several key areas of machine learning and natural language processing, particularly those relevant to embedding creation and classification tasks.

2.1 Embeddings

Given a sequence of for example words $\mathbf{x} = \{x_1, x_2, \cdots x_n\} \in \mathbb{V}^n$ from a vocabulary $\mathbb{V}$, an embedding model $\phi(x_i) : \mathbb{V}^n \rightarrow \mathbb{R}^d$ maps the discrete input items into continuous, high-dimensional vector representations. The magnitude and direction of these vectors capture semantic and syntactic properties, enabling the model to encode and compare meaning. Text passages with similar meaning should have embeddings that are close in vector space, facilitating tasks like clustering and classification [4].

2.2 Transformer Models

Transformers are the foundation for much of modern natural language processing. These models employ a self-attention mechanism that enables them to capture relationships between all tokens in an input sequence, regardless of distance, making them highly effective at understanding context and semantics [7]. The transformer architecture by Vaswani et al. [7] proposed an encoder module designed to encode a sequence of embeddings into a higher-dimensional vector representation, followed by a decoder to produce an output sequence from it. In our work, we specifically focus on using encoder-only models, as they are highly effective for creating embeddings that capture semantic information efficiently. Encoders excel at understanding and representing input text in a rich, contextualised manner, which is particularly useful for tasks that require high-quality semantic embeddings for downstream applications.

2.3 Sentence-Transformers

Sentence-Transformers are specialised models designed for creating efficient and meaningful sentence embeddings. Unlike general encoder-only transformers like BERT [1], which require computationally complex pooling strategies to derive fixed-length embeddings, Sentence-Transformers provide an end-to-end approach for generating embeddings optimised for semantic understanding [6], making them suitable for similarity comparison, clustering, and information retrieval tasks. For this project, we used the `all-mpnet-base-v2` Sentence-Transformer model, which encodes sentences into rather compact 768-dimensional vectors, striking a balance between computational efficiency while maintaining strong performance, evident by its high ranking in the Massive Text Embedding Benchmark (MTEB) [5] at the time of writing.

2.4 Fine-Tuning for Domain Adaptation

Pre-trained language models are trained on large-scale datasets to be effective in a wide range of domains; however, to perform better in highly specialised tasks, domain-specific adaptations are often employed. In this project, we applied fine-tuning to adapt the `all-mpnet-base-v2` transformer encoder model to the ECSS domain. This involved further training the model on ECSS-specific requirements data to better align the produced embeddings with the typical language used in ECSS standards. The fine-tuning process aimed to make the model more effective at understanding the nuances and terminologies unique to the ECSS requirements.

2.5 Masked Language Modelling

Masked Language Modelling (MLM) was used as part of the model training process to help the encoder learn contextual relationships within text. During MLM, random words in the input sequence are masked, and the model is trained to predict these masked words based on the surrounding context. This technique, commonly used in pre-training of encoder transformer models like BERT [1], helps the model understand both short-range and long-range dependencies within the text, resulting in embeddings that are more contextually aware and suitable for downstream tasks.

2.6 Contrastive Learning

Contrastive learning (CL) techniques were used to refine the embeddings by training the model to distinguish between similar and dissimilar pairs of data. Two main approaches were explored: pairwise and triplet learning. In the pairwise approach, the model learned to bring semantically similar sentences closer together in the high-dimensional vector embedding space while distancing dissimilar ones. The triplet approach further refined this by using an anchor, a positive sample, and a negative sample, ensuring that the positive sample is closer to the anchor than the negative one. These approaches are employed to make embeddings more discriminative, which is useful for tasks involving similarity detection and clustering [8].

2.7 Classical Classification Models

The embeddings generated by transformer models were used as inputs for trained classification models. We compared a number of different algorithms, including logistic regression, decision tree, random forest, support vector classifier (SVC), gradient boosting, and a four-layer neural network. Embeddings were generated using both the pre-trained `all-mpnet-base-v2` model and later the domain-specific fine-tuned version. The embeddings served as input features for the classifiers, which were tasked with identifying different categories of requirements, such as distinguishing between technical and non-technical ECSS requirements.

3 Python Requirements Reader Tool (ReqReader)

The Python Requirements Reader Tool (ReqReader) was developed to facilitate the extraction and organisation of highlighted requirements from ECSS-standard Word documents (.docx files). These files had previously been reviewed by different members of the ECSS working groups, who highlighted the requirements according to various categories: "contractual requirement," "means of compliance," "information," or "inconclusive." The goal of ReqReader was to automate the process of consolidating these annotations, which were distributed across multiple documents, into a unified and structured format.

ReqReader traverses a directory containing the highlighted ECSS-standard documents, systematically extracting the text of each requirement along with its associated headings, numbering, and type based on the provided highlights. The extracted information is then compiled into an Excel sheet, creating a centralised and searchable repository of requirements. This makes it easier for stakeholders to gain a comprehensive overview of the requirements within each standard and across multiple standards.

In addition to extraction and consolidation, ReqReader includes a summarisation feature. This feature tracks changes in the composition of requirement types over time, helping to identify trends or shifts in categorisation across different versions of the documents. This capability is particularly useful for tracking the evolution of requirements during the review process and provides an insightful overview of how different types of requirements change, both within individual standards and across the entire ECSS framework.

Listing 1. Usage of ReqReader tool from the terminal

```
# Read a directory of ECSS standards and produce an output sheet
python src/reader.py --input [path to input directory]

# Read the most recent .xlsx output and update a summary sheet
python src/reader.py --stats
```

While ReqReader was the least AI/ML-focused of the tools developed in this project, it quickly found widespread adoption within the ECSS working group procedures. Its ability to automate what was previously a laborious manual process significantly improved the efficiency of requirement management, helping teams to focus more on the content and quality of requirements rather than the mechanics of organising them. By making the requirements more accessible and easier to analyse, ReqReader has become a valuable tool for streamlining the standardisation workflow and enhancing collaboration across working groups.

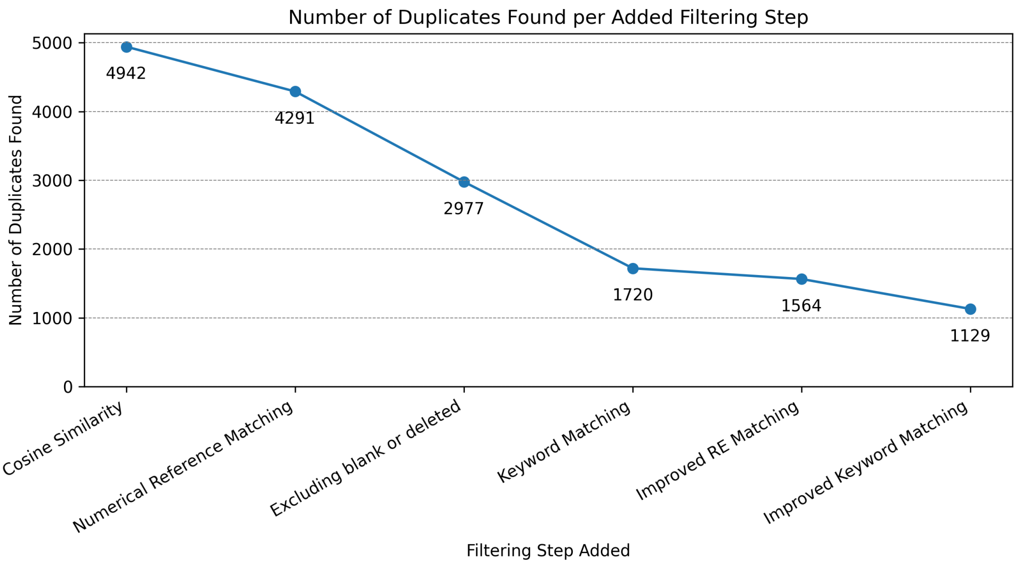

Fig. 2. Number of flagged duplicate requirements as filtering steps were added. Using just the cosine similarity of embeddings resulted in high recall, with more filtering steps improving precision and accuracy.

4 Duplicate Requirement Detector (DRED)

One of the primary goals of the ECSS NextGen initiative is to streamline and reduce the overall number of requirements. The ECSS standards contain thousands of requirements distributed across multiple documents, making consistency and avoiding duplication a significant challenge. While maintaining some duplicates across different standards is still desired for now before transitioning to a full database-first system, the focus was to identify and reduce redundant requirements down to a manageable set that is feasible for manual examination by domain experts. The Duplicate Requirement Detector (DRED) was developed to address the challenge of identifying redundant requirements within ECSS standards. DRED addresses this by encoding extracted requirements from the ECSS standards and applying a multi-step process to identify and flag potential duplicates.

DRED begins by reading the Excel sheet of requirements and processing each requirement's text using the `all-mpnet-base-v2` model to create fixed-length embeddings. It performs a pairwise cosine similarity comparison across all requirements. Cosine similarity is a metric that measures the cosine of the angle between two vectors in a multidimensional space, and it effectively quantifies how similar two texts are. A value closer to 1 indicates a high degree of similarity, while a value closer to 0 indicates dissimilarity. Requirements with a cosine similarity greater than 0.9 are flagged as potential duplicates. Given the absence of labelled data to indicate whether two requirements should indeed be considered duplicates, expert review was required to assess the performance of the system. According to the standardisation engineers who reviewed the flagged requirements, DRED achieved a high recall, meaning that many duplicate requirements were captured, though there were still many false positives (indicating lower precision).

To improve precision and accuracy, filtering steps were added to exclude requirements that were erroneously marked as duplicates. The first step involved extracting numerical references within the requirement text. Requirements referring to different tables, figures, or clauses were filtered out if the numerical references did not match exactly, ensuring that only requirements referring to the same entities were retained. Furthermore, requirements featuring any form of redaction like `<<deleted>>` were removed as these were trivial.

In addition, we used KeyBERT [3] with the `all-mpnet-base-v2` model as the underlying transformer to extract keywords from each requirement's text. For each requirement, up to a maximum of 10 keywords or the length of the requirement text were initially extracted, with the top five most confident keywords being retained. The keywords of potential duplicate pairs were then compared, and only those with matching keywords were kept in the final list of flagged duplicates.

Through iterative improvements in numerical reference matching and keyword filtering, DRED was able to reduce the initial list of over 5,000 flagged potential duplicates to just over 1,000 requirements out of approximately 45,000 total requirements. This significant reduction allowed the expert panel to focus on a manageable subset, ensuring that actual duplicates could be identified and subsequently removed from the next version of the ECSS standards. By automating much of the initial identification process, DRED reduces the burden on standardisation engineers and helps to improve the consistency and quality of the ECSS standards by ensuring that redundant requirements can systematically be identified and addressed.

5 Requirements Classification (ECSSorter)

In the next generation of ECSS standards, an effort is being made to focus on what requirements need to be met, rather than how they should be achieved, with the intention of reducing bureaucratic burden and simplifying the implementation of ECSS standards. For this, it helps to classify ECSS requirements as being either "contractual" or "means-of-compliance" and either "technical" or "non-technical". The former are requirements stating exact specifications of what a parts supplier, for example, must meet for a given product they deliver, whereas the latter category of requirements provides guidance on how a given requirement can be best met. With the large number of requirements, this task would be too laborious for manual review, which is why the ECSSorter was developed.

The ECSSorter takes as input multiple requirements texts from an Excel sheet or a single requirement as a string. For the "contractual vs. means-of-compliance" classification task, a set of 900 human-made labels created by ECSS standardisation engineers already existed from a previous attempt at this task. For the "technical vs. non-technical" classification, a different approach had to be taken as no dataset existed: requirements from standards that were considered to be mostly technical by ECSS standardisation engineers were labelled as technical, while requirements from generic "higher-level" ECSS standards were labelled as non-technical. Although these labels were noisy, they provided a starting point for training the classifiers.

The text of each requirement was encoded using the `all-mpnet-base-v2` Sentence-Transformer model. These embeddings were then fed as input features into six different classification algorithms of increasing complexity: logistic regression, decision tree, random forest, support vector classifier (SVC), gradient boosting, and a four-layer neural network. The intention behind comparing different classification algorithms was to determine which provided the optimal balance of performance metrics, namely precision, recall, accuracy, and F1 score, without assuming that more complex algorithms would necessarily yield the best results.

For both classification tasks, the datasets were split into training and validation sets (75% to 25% of total data, respectively), which for the "contractual vs. means-of-compliance" were the 900 labelled examples, and for the "technical vs. non-technical" were the around 45,000 currently in-use ECSS requirements.

The SVC model achieved the highest performance for the "technical vs non-technical" classification task, with an accuracy of 99.2% and an F1 score of 97.1% on the validation set (see Table 1). All models achieved an accuracy above 92%, but the F1-score differences proved to be more indicative

Task	Model	Precision	Recall	Accuracy	F1
Contractual vs Means-of-Compliance	SVC	**0.911**	**0.872**	**0.860**	**0.891**
	Gradient Boosting	0.910	0.800	0.840	0.851
	Neural Network	0.903	0.757	**0.860**	0.824
	Logistic Regression	0.807	0.794	0.812	0.800
	Random Forest	0.826	0.766	0.808	0.795
	Decision Tree	0.799	0.542	0.790	0.646
Technical vs Non-Technical	SVC	**0.980**	**0.963**	**0.992**	**0.971**
	Neural Network	0.959	0.959	0.989	0.959
	Gradient Boosting	0.959	0.907	0.983	0.932
	Logistic Regression	0.946	0.871	0.977	0.907
	Random Forest	0.890	0.715	0.953	0.793
	Decision Tree	0.719	0.708	0.928	0.714

Table 1. Performance metrics of classification algorithms for "contractual vs. means-of-compliance" and "technical vs. non-technical" classification. Sorted descending by F1 score for each task.

of performance, likely reflecting the high class imbalance in the dataset, where approximately 70% to 80% of requirements were non-technical. Notably, the SVC outperformed more complex algorithms, such as the neural network and gradient boosting.

For the "contractual vs means-of-compliance" classification, we have similar results, with the SVC model again showing the best performance, though with slightly lower metrics compared to the "technical vs non-technical" classification. It is notable that despite the small dataset size, strong performance in this task could be achieved.

The ECSSorter proved to be an effective tool for categorising ECSS requirements, providing a structured approach that simplified the otherwise labour-intensive process of manually distinguishing between different types of requirements. By testing a range of classification algorithms, our approach balanced both simplicity and accuracy, ultimately resulting in a tool that could be effectively utilised by the ECSS working groups, although performance can still very likely be increased with a larger, labelled dataset.

Similarly to ReqReader and DRED, a simple terminal interface was created for ECSSorter to classify either a single requirement text or a suitably formatted ECSS `.xlsx` sheet with multiple requirements. Additionally for this project, a GUI demo app was created to classify into "technical vs non-technical" (see Appendix B).

6 Fine-Tuning the Sentence-Transformer Model

The final undertaking of this project was to create a fine-tuned version of the `all-mpnet-base-v2` Sentence-Transformer model trained specifically on ECSS standards and requirements. The motivation for fine-tuning was driven by the highly specialised language used in the ECSS, which is unlikely to match the data on which our embedding models was trained. By developing a domain-specific model, the aim was to improve performance for downstream tasks such as clustering and classification, ensuring the embeddings capture the unique nuances of ECSS content. A link to the Weigths & Biases workspace used to log training and evaluation can be found under Appendix A.

6.1 Fine-Tuning Methodologies

Two main fine-tuning approaches were employed for the Sentence-Transformer model: Masked Language Modelling (MLM) and Contrastive Learning (CL), a setup which closely resembles the training regime for the BERT and initial Sentence-Transformer models [1] [6].

6.1.1 Masked Language Modelling (MLM). This fine-tuning step involved randomly masking 15% of the tokens in the input sequences and training the model to predict them, thereby training it to improve its contextual understanding of domain-specific text. MLM training was conducted using a combination of all ECSS and CCSDS standards, extracted using the ReqReader tool described in Section 3. The CCSDS (Consultative Committee for Space Data Systems) standards represent another highly specialised domain in the space sector. The rationale behind combining these two sources of standards was that it would provide a richer learning context to the model, given their complementary focuses on space data and requirements, allowing the model to capture a more diverse range of technical terminology and nuances, ultimately resulting in better embeddings for the target downstream tasks. Ultimately, the combination of ECSS and CCSDS data led to marginally better MLM validation performance compared to just using ECSS standards, so any reference made to a MLM fine-tuned embedding model refers to the model trained on ECSS + CCSDS.

6.1.2 Contrastive Learning (CL). To enhance the model's ability to create more discriminative embeddings, contrastive learning was applied, where two variants were experimented with. The first involved pairwise contrastive learning (shorted to CLP for Contrastive Learning Pairwise). For this, a labelled dataset of ECSS requirements was created whereby requirements coming from the same section of the same standard were marked as positive examples. Negative examples were requirements from different sections within the same standard, as to not make the task of differentiating between positive and negative pairs too trivial. For example: requirements *5.1a* and *5.1c* from standard *ECSS-E-ST-10-02C* were marked as positive, while *5.1a* and *6.3e* were a negative example. A total of 100k examples were sampled to create the final dataset, with an even split of positive and negative examples. The model was then trained using `CosineSimilarityLoss` from the Sentence-Transformer package's `losses` class.

The second approach, Contrastive Learning Triplets (CLT), involved a similar set up, where given a triplet of (anchor, positive, negative) requirements, the model is taught to minimizes the distance between anchor and positive while it maximizes the distance between anchor and negative, known as `TripletLoss`. Despite extensive efforts in hyperparameter tuning and different mining strategies for negative samples, the triplet-based contrastive learning approach did not converge satisfactorily. Mining strategies included sampling negatives from different sections within the same standard and sampling from entirely different standards, but neither approach led to stable convergence.

6.2 Evaluating the Fine-Tuned Embeddings

The primary goal of fine-tuning was to create embeddings that could replace those generated by the original `all-mpnet-base-v2` model, leading to improved performance in downstream tasks, specifically duplicate detection, classification and clustering.

Four versions of the model were evaluated:

- **Base**: The original `all-mpnet-base-v2`.
- **Base + MLM**: Fine-tuned with MLM using both ECSS and CCSDS standards.
- **Base + MLM + CLP**: Fine-tuned with MLM, followed by pairwise contrastive learning.
- **Base + CLT**: Fine-tuned with MLM, followed by triplet-based contrastive learning (included for completeness despite convergence issues).

6.2.1 Duplicate Detection. The models were evaluated on the task of detecting duplicate requirements using the DRED tool (see Section 4). A significant increase in the number of duplicates detected was observed when using embeddings from models fine-tuned with contrastive learning, shown in Figure 3. This increase may indicate that the contrastive learning process enhanced the model's sensitivity to subtle similarities, leading to more aggressive duplicate detection. However, it also resulted in increased false positives, something which could be alleviated by re-tuning hyperparameters such as the cosine similarity threshold, to balance recall and precision effectively.

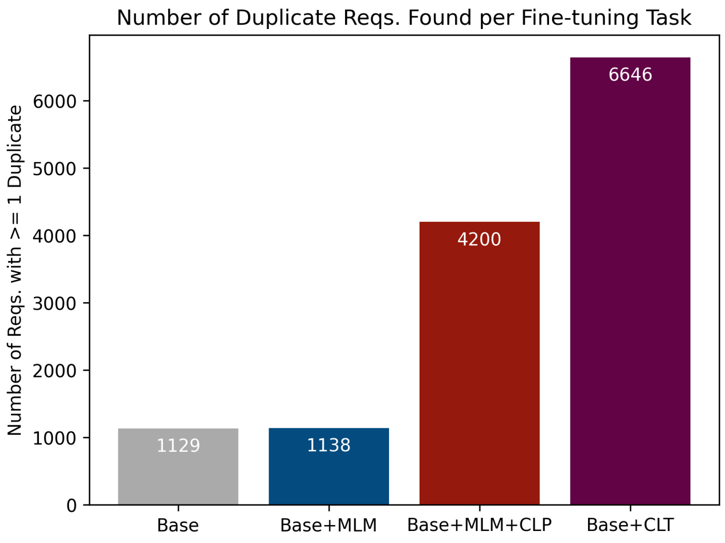

Fig. 3. Number of flagged duplicates per embedding model used. Subjecting the embedding model to some form of contrastive learning during fine-tuning resulted in a large increase in number of requirements showing up as having one or more duplicates based on its text's embedding.

6.2.2 Classification Performance. : To also evaluate downstream classification performance, we focused on the "contractual vs means-of-compliance" classification task, for which expert labels actually exist. The best-performing trained SVC from the ECSSorter (Section 5) was reused. The performance on the validation subset (225 examples) showed that the base model consistently outperformed all fine-tuned versions, with each additional fine-tuning step slightly decreasing classification accuracy and F1 score. The order of performance was: Base model > Base + MLM > Base + MLM + CLP > Base + CLT as visible in Figure 4. This suggests that while fine-tuning improved the model's familiarity with domain-specific language, it may have led to overfitting or reduced generalisability, negatively impacting classification effectiveness.

6.3 UMAP Visualisation of Embeddings

To better understand how the different fine-tuning approaches affected the embedding space and to gain an initial insight into clustering performance, UMAP was used to visualise the embeddings generated by each model. An ECSS standard can be within one of the five branches of ECSS (see Appendix C). Requirements were colour-coded according to branches of the standard they are found in and projected into 3D space using UMAP.

For the base model, a relatively compact grouping of all requirements was observed, with a distinct clustering of Space Engineering (M, green) and Space Product Assurance (Q, blue) requirements.

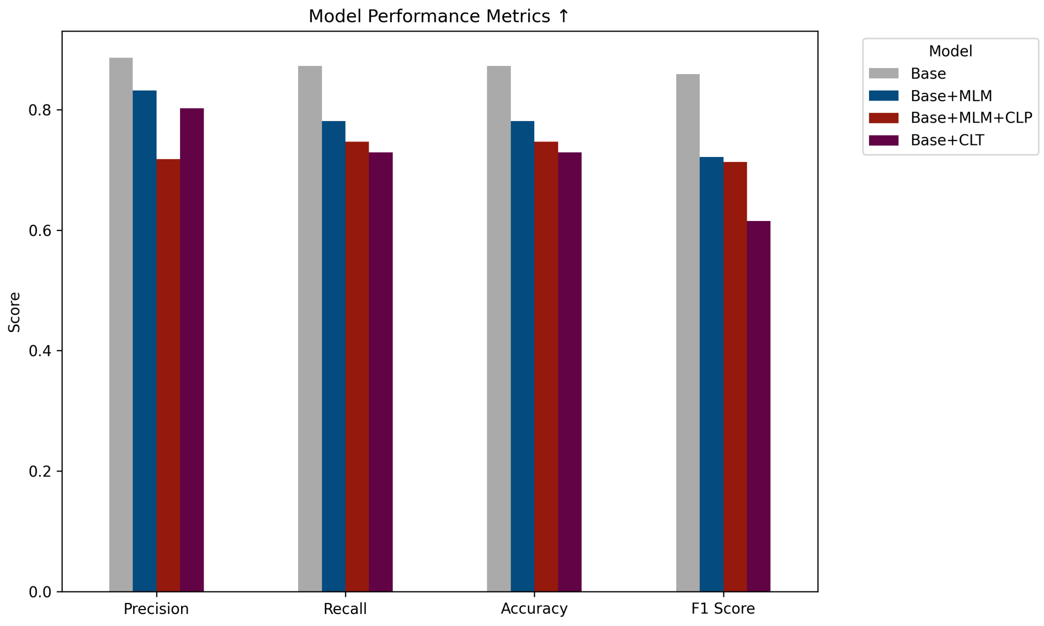

Fig. 4. Performance on the "contractual vs means-of-compliance" classification task per embedding model. The more complex the fine-tuning training regime, the worse the performance.

The Base + MLM model showed a similar clustering structure but with a different shape in the 3D UMAP space, suggesting a change in how the model perceived relationships between requirements.

With the Base + MLM + CLP model, the embeddings of "E" (Space Engineering) requirements were more clearly separated from others, forming a distinct sphere around a central cluster of other requirements. This indicates that contrastive learning might have helped to differentiate these requirements more effectively.

The most intriguing visualisation came from the Base + CLT model, where the embedding landscape exhibited a complex, snake-like structure. Requirements from the "E" branch were more separated, and there was a clear clustering of "M" branch requirements, while a mix of "E" and "Q" requirements formed larger mixed clusters. This unique structure, while visually interesting, may also reflect the instability of the triplet-based training that failed to converge.

6.4 Conclusions and Future Work

The fine-tuning experiments highlighted both the potential benefits and challenges of adapting pre-trained models to highly specialised domains. Additional fine-tuning with contrastive learning did not yield the expected gains in downstream task performance. Perhaps experimenting with other contrastive learning techniques or better initialisation strategies for self-supervised learning objectives as suggested by Draganov et al. [2] could have helped to facilitate better training. Future work should perhaps focus on acquiring more labelled training data to mitigate overfitting and improve the robustness of the model, rather than investing in compute-heavy fine-tuning.

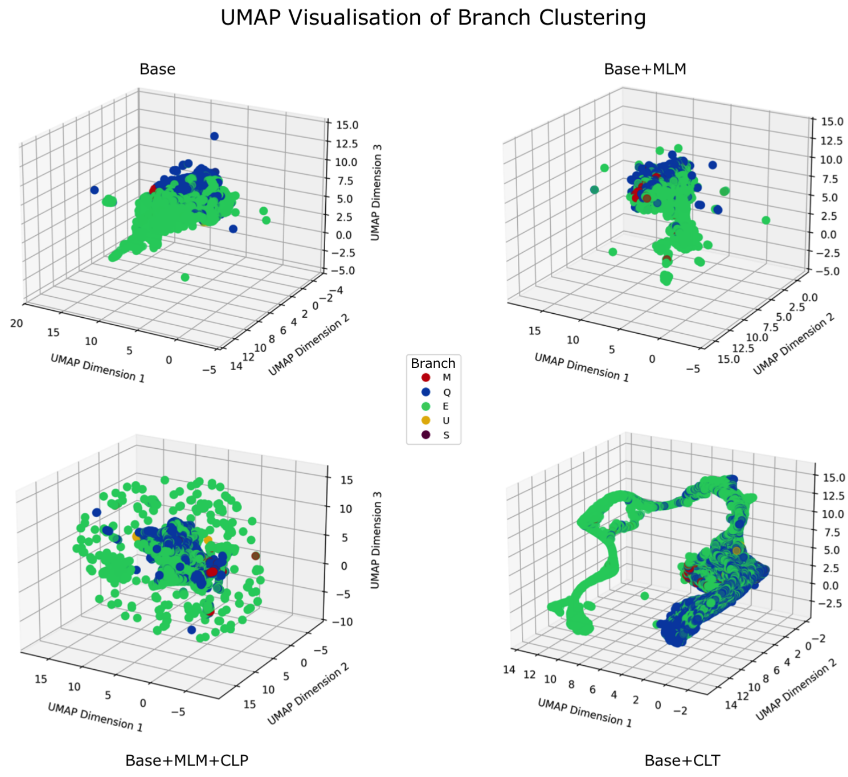

Fig. 5. Visualisation of requirements clustering based on their standard's ECSS branch after projecting down to three dimensions using UMAP.

7 Discussion

7.1 Challenges

The development and implementation of tools for the ECSS standards review process presented several challenges. One of the primary obstacles was the difficulty in obtaining labelled data for both duplicate detection and classification tasks. The scarcity of such labelled data made it challenging to train models effectively, especially in specialised domains where the availability of high-quality annotations is limited. The specialised nature of the ECSS language further exacerbated these challenges, as it is not well-covered by general pre-trained models. Pre-trained language models are typically trained on large-scale general-purpose corpora, which often lack the specific technical jargon and nuanced language commonly used in the space sector.

7.2 Tool Adoption

Despite these challenges, the tools developed in this project achieved decent performance and were well-received by the ECSS working group. The ReqReader and DRED tools proved particularly valuable in streamlining the standards review process. Engineers within the ECSS working group provided positive feedback, highlighting how these tools significantly reduced manual effort and enhanced the overall accuracy of their review processes. By automating key aspects of the standards review workflow, these tools allowed experts to focus more on critical decision-making rather than repetitive, labor-intensive tasks. The positive reception of these tools underscores their practical utility and the tangible benefits of applying AI-driven solutions to improve complex engineering processes.

7.3 Future Work

There are several avenues for future work that could enhance the current system. One major area of improvement involves obtaining more labelled data, which would directly enhance downstream model performance. Additionally, exploring unsupervised learning methods may help overcome the limitations posed by the scarcity of labelled data. Another promising direction is the investigation of other AI techniques, such as reinforcement learning or generative models. These approaches could potentially generate missing standards or assist in tailoring requirements to better suit specific space missions. As the space industry continues to evolve, these tools could provide a foundation for future advancements in automating and optimising the management of space standards, ultimately supporting more efficient and robust space exploration.

Acknowledgments

To Alexandru Mancas and Jaap Kamps for kindly offering their supervision in this project, and the entire TEC-QES section for welcoming me with open arms and sharing many fun conversations.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL] https://arxiv.org/abs/1810.04805

[2] Andrew Draganov, Sharvaree Vadgama, and Erik J. Bekkers. 2024. The Hidden Pitfalls of the Cosine Similarity Loss. arXiv:2406.16468 [cs.LG] https://arxiv.org/abs/2406.16468

[3] Maarten Grootendorst. 2020. KeyBERT: Minimal keyword extraction with BERT. https://doi.org/10.5281/zenodo.4461265

[4] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. arXiv:1310.4546 [cs.CL] https://arxiv.org/abs/1310.4546

[5] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. MTEB: Massive Text Embedding Benchmark. *arXiv preprint arXiv:2210.07316* (2022). https://doi.org/10.48550/ARXIV.2210.07316

[6] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv:1908.10084 [cs.CL] https://arxiv.org/abs/1908.10084

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. https://arxiv.org/abs/1706.03762

[8] Lilian Weng. 2021. Contrastive Representation Learning. *lilianweng.github.io* (May 2021). https://lilianweng.github.io/posts/2021-05-31-contrastive/

A Fine-tuning Training and Evaluation Weights & Biases Workspace

Link to the public wandb workspace

B ECSSorter GUI App

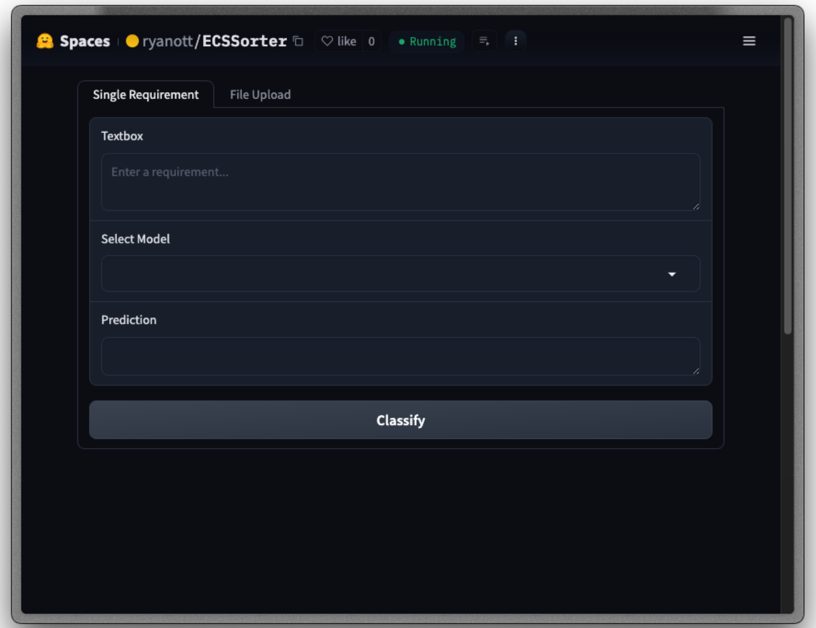

Fig. 6. GUI of ECSSorter app to classify requirements into "technical vs non-technical". Available on Huggingface Spaces. Note that for the file upload feature the correct ECSS EARM formatting is required. Source code for the app is available under Files.

C ECSS Standards Structure

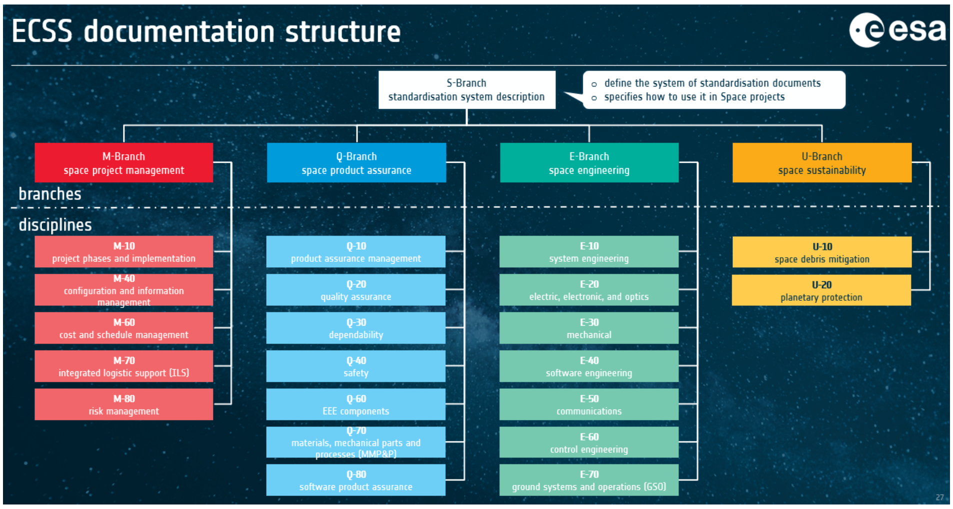
ECSS documentation structure
esa
S-Branch
standardisation system description
define the system of standardisation documents
specifies how to use it in Space projects
M-Branch
space project management
Q-Branch
space product assurance
E-Branch
space engineering
U-Branch
space sustainability
branches
disciplines
M-10
project phases and implementation
M-40
configuration and information management
M-60
cost and schedule management
M-70
integrated logistic support (ILS)
M-80
risk management
Q-10
product assurance management
Q-20
quality assurance
Q-30
dependability
Q-40
safety
Q-60
EEE components
Q-70
materials, mechanical parts and processes (MMP&P)
Q-80
software product assurance
E-10
system engineering
E-20
electric, electronic, and optics
E-30
mechanical
E-40
software engineering
E-50
communications
E-60
control engineering
E-70
ground systems and operations (GSO)
U-10
space debris mitigation
U-20
planetary protection

Fig. 7. Structuring of ECSS standards into branches and disciplines.