Req-BERTA and Req-LLaMA: Finetuning Large Language Models for Requirements Management in Multidisciplinary Integration Projects.

Alan Ramirez[1].

[1] University of Leeds, LS2 9JT, UK
od20aprg@leeds.ac.uk

Abstract. This study explores fine-tuning approaches of Large Language Models (LLM) for Requirements Management (RM) in a large multidisciplinary integration project and contributes a new multi-label requirements dataset. Particularly we investigate Transformers with Encoder-Classification Head type architectures as well as generative Encoder-Decoder Transformer architectures to classify textual requirements according to their semantic and syntactical properties. We present Req-BERTA and Req-LLaMA, two multilabel LLMs capable of classifying requirements according to the syntax - Functional Requirements (FR) from Non-Functional Requirements (NFR) - and the semantic properties of the Requirement - Discipline- with state-of- the-art performance, beating the existing benchmark of FR vs NFR classifiers with an F1 score of 0.984. We further demonstrate that simpler architectures not only reduce inference time in production but can offer the same if not better performance than more complex models trained on the same dataset for task specific applications, and emphasize the importance of data quality over model complexity.

Keywords: Req-BERTA, Req-LLaMA, LoRA.

1 Introduction and Background Research

A high-quality requirement is defined as necessary, clear, concise, measurable and verifiable (J. M. Lanza-Guiterrez, 2017) and can enhance organisational learning and promote innovation. (Klaus Pohl, 2009) provided a comprehensive overview of the requirements engineering processes and emphasized the importance of a structured approach to requirements elicitation, analysis, specification, validation, verification and management. The classification of Functional Requirements (FR) from Non-Functional Requirements (NFR) lead to SO/IEC 25010 and is a classic problem in the field of Natural Language Processing (NLP).
Task-specific architectures have been deprecated in favour of fine-tuning pre-trained models. Word Vectors - specifically Support Vector Machines (SVMs)- where one of the first steps towards this path, evolving to Recurrent Neural Networks (RNNs). Finally, the arrival of the transformer architecture (Vaswani, 2017) and its variants such as BERT (Jacob Devlin, 2018) and the popular GPT (OpenAI, 2023) cemented the fine-tuning approach. Below we explore the most significant studies and benchmark their performance.

1.1 Task Specific Architectures

Task-specific architectures can achieve competitive performance metrics but they are expensive to develop and prone to maintenance, specially the case of Decision Trees (DT) which require heavy manual labour and the support of SMEs. In the quality classification of requirements, the most notables DT architectures developed are (F Fabbrini, 2001) who developed the QuARS tool for requirements quality, (Benedikt Gleich, 2010) who studied ambiguity detection with effective precision, (Sri Fatimah Tjong, 2013) who provided a quality classifier with 100% recall at the cost of precision arguing that designers must detect all ambiguities in safety-critical designs. . DT have also been used for quality classification in Railway Engineering (B. Rosadini, 2017). Similarly, Smella (Henning Femmera, 2017) adhered to the ISO 29148 standards for parsing requirements into a quality analysis. Ontological approaches on the PROMISE dataset to detect vagueness of requirements were explored in (Feng-Lin Li, 2014) using qualities as mapping adhering to the quality definition in ISO/IEC 25010. (Accenture, 2017) investigated vagueness in multi-lingual requirement environments, curating a tool containing a database of key words typically found in vague requirements. DTs have also been used for the the binary FR vs NFR problem (Cleland-Huang, 2006) (Jane Cleland-Huang, 2007) (Ishrar Hussain, 2008) and the multilabel problem (Prateek Singh, 2016) (Zijad Kurtanovic, 2017) (Fabiano Dalpiaz, 2019) reducing the task to linguistic-based rules for feature engineering. The tokenisation of strings plays a crucial role in the performance of the model, regardless of the architecture. Various requirement classification studies used feature techniques (Zahra Shakeri Hossein Abad, 2017) (Manal Binkhonain, 2019) achieving reasonable performance metrics for the binary class. (Osamah AlDhafer, 2022) explored the multilabel problem by tokenising both words and characters and found that word-based tokens yielded the highest result whilst proving the advantages of a no-feature-engineering approach. The architecture employed was a Bidirectional Gated Recurrent Neural Network (BiGRU), which showed promising results in other NLP exercises but had not been explored before for requirements classification. Other notable feature techniques are found in (Alhindawi, 2018), who proposed the Latent Dirichlet Allocation for Requirements Classification (LDARC) approach – an information retrieval-based system.

1.2 Vectorisation Methods

Vectorisation methods have also been heavily explored, such as Support Vector Machines (SVMs) (Zijad Kurtanovic, 2017) (Sousuke Amasaki, 2018). A re-tagged version of the PROMISE Dataset to address the multilabel problem was used in (Abderahman Rashwan, 2013), who studied SVM, K-Nearest Neighbour (KNN), and Naïve Bayes (NB) techniques, including the identification of FR and NFR in unstructured texts. (Osamah AlDhafer, 2022) also used the PROMISE dataset to further classify NFR into different types, making the requirements classification a multi-label problem. Other significant SVM techniques (Zijad Kurtanovic, 2017) (Sousuke Amasaki, 2018) were performed on the multilabel case, revealing challenges in classifying certain categories of NFR.

1.3 Convolutional Neural Networks

(Jonas Winkler, 2016) proposed CNNs for the binary classification problem. (Alex Dekhtyar, 2017) also used CNNs with pre-trained embeddings (Word2Vec) for the binary problem and compared it against a Naïve Bayes with TF-IDF representations. (Zahra Shakeri Hossein Abad, 2017) also explored supervised techniques (Binary Naïve Bayes). However NB and SVM often use the bag-of-words conversion, which prevents the model from understanding patterns in the order of words. A more desirable approach is recognised in (Jonas Winkler, 2016) which recommend the Word2Vec or GloVe vectorisation methods. Similarly as with (Alex Dekhtyar, 2017) , CNNs with Word2Vec have also been explored in the multiclass problem (Raul Navarro-Almanza, 2017). Word2Vec was also used in (Muhammad Younas, 2019) , which used Wikipedia corpus to train the vectorization before feeding in the key words of requirements. (Md. Abdur Rahman, 2019) compared GRU & LSTM with Word2Vec, with LSTM yielding the best results for the multilabel. Artificial Neural Networks (ANN) have also been compared against CNN with CNN outperforming ANN (Cody Baker, 2019) for the multiclass case.

1.4 Transformers: The rise of Large Language Models

(Vaswani, 2017) introduced the Transformer architecture, cementing itself as the leading architecture for NLP tasks, with many state-of-the-art variants developed (Jacob Devlin, 2018) (Deep Mind, 2022) (Google, 2022) (OpenAI, 2023). Unlike recurrent units such as Long Short Term Memory (LSTM) or Gated Recurrent Units (GRUs) where the current timestep's hidden state depends on the previous timestep's state, attention heads compute the hidden states of all time steps simultaneously enabling complex context understanding, a prime reason why they have become the default architecture of choice for NLP tasks. BERT (Jacob Devlin, 2018) introduced the possibility of fine-tuning a pre-trained transformer model using deep bidirectional representations of unlabelled text by adding a single additional output layer to create models finetuned to a task specific application. Google's BERT algorithm has had a requirement version developed called NoRBERT (Tobias Hey, 2020) and in a parallel thesis with (Hanisch, 2020) who studied the use of transformers for detecting quality issues but obtained poor results, acknowledging the dataset quality proved a threat to validity but still managed to outperform all models which had not done manual pre-processing of the dataset. Google also released LaMDA (Google, 2022), a 137B parameter model optimised for Dialogue Applications. Fine-tuning is used with annotated data and the model is allowed to consult external knowledge sources to improve dialogue quality. The scaling of the model demonstrated promising results in the quality of the response, yet fell behind on safety and groundeness when compared to a human response. When combined with fine-tuning and information-retrieval based systems these two fields were marginally improved. LaMDA supported the view that performance is improved with model size. (Deep Mind, 2022) debunked the assumption that the performance of Large Language Models (LLM) are improved by scaling up the number of parameters of the model, but rather by increasing the amount of training data. The focus on complexity of NN often meant that most complex model were underfitted from their optimal weight assignment and have a higher memory footprint and inference cost. Their conclusion debunked that of (J. Kaplan, 2020), and concluded that given a compute budget, the model parameters and training tokens ought to be scaled in equal proportions. Deep Mind investigated 400 language models ranging from 70M to 16B parameters with 5B to 500B tokens to develop Chinchilla - a 70B parameter model trained on x4 more data than its predecessor Gopher (Deep Mind, 2022) with 1.4 Trillion tokens. Indeed, Chinchilla outperformed its predecessor Gopher (Deep Mind, 2022) on the Massive Multitask Language Understanding (MMLU) benchmark by 7.6%

and GPT-3 by 23.7%, despite having 105B parameters less than the latter. Shortly after, LLaMa (Meta AI, 2023) introduced an open-source generative LLM based on the Transformer architecture presented by Google (Vaswani, 2017) but focused on scaling the amount of training data used to train the models. (Meta AI, 2023) proved that a 7B parameter model continued improving performance after feeding a larger amount of data than what (Deep Mind, 2022) recommended. Ultimately this further reinforces the hypothesis that it is advantageous to train simpler foundation models with relatively larger tokens since it demands lower computational resources and power.

1.5 Performance Benchmark

Table 1: Performance of Requirement Classifiers benchmark summary.

Study	Type	Architecture	Precision	Recall	F1
(Ishrar Hussain, 2008)	Binary	DT	0.978	1	0.990
(Benedikt Gleich, 2010)	Binary (Quality)	DT	0.95	0.86	0.86
(Sri Fatimah Tjong, 2013)	Binary (Quality)	DT	0.68	1	0.810
(Jonas Winkler, 2016)	Binary	CNN, Word2Vec	0.814	0.820	0.810
(Zijad Kurtanovic, 2017)	Binary	SVM, bag of words	0.925	0.925	0.925
(Zahra Shakeri Hossein Abad, 2017)	Binary	DT, Feature engineering	0.944	0.94	0.94
(Alex Dekhtyar, 2017)	Binary	NB, WC	0.822	0.888	0.850
		NB, TF-IDF	0.850	0.793	0.802
		CNN, Word2Vec	0.879	-	0.914
(B. Rosadini, 2017)	Binary (Quality)	DT	0.643	0.883	0.744
(Henning Femmera, 2017)	Binary (Quality)	DT	0.590	0.82	0.686
(Tobias Hey, 2020).	Binary	NoRBERT	0.92	0.92	0.925
(Hanisch, 2020)	Binary (Quality)	BERT	0.45	0.53	0.48
		DistilBERT	0.33	0.71	0.48
		ERNIE2.0	0.35	0.85	0.50
(Abderahman Rashwan, 2013)	Muti-Class	SVM	0.800	0.490	0.535
(Prateek Singh, 2016)	Muti-Class	Rule-based	0.950	0.965	0.955
(Zahra Shakeri Hossein Abad, 2017)	Muti-Class (Unsupervised)	LDA	0.62	0.31	0.414
		BTM	0.08	0.03	0.044
		Hirearchical	0.21	0.16	0.182
		K-means	0.20	0.15	0.170
		NB	0.22	0.19	0.204
(Zahra Shakeri Hossein Abad, 2017)	Muti-Class	BNB	0.90	0.91	0.905
(Zijad Kurtanovic, 2017)	Multi-Class	SVM, Feature selection	0.655	0.637	0.618
(Raul Navarro-Almanza, 2017).	Muti-Class	CNN, skipgram	0.81	0.785	0.77
(Alhindawi, 2018)	Muti-Class	LDA	0.45	0.96	0.613
(Sousuke Amasaki, 2018)	Muti-Class	SVM, NB, LR, RF Vectorisation: Word2Vec, Doc2Vec, SCDV, TF, TF-IDF.	0.774	0.610	0.677
(Md. Abdur Rahman, 2019)	Muti-Class	CNN	0.813	0.785	0.77
		GRU	0.961	0.961	0.961
		LSTM	0.973	0.967	0.966
(Muhammad Younas, 2019)	Muti-Class		0.51	0.41	0.42
(Fabiano Dalpiaz, 2019)	Muti-Class	SVM, Feature selection	0.79	0.765	0.770
(Tobias Hey, 2020)	Muti-Class	NorBERT	0.825	0.88	0.845
(Tobias Hey, 2020).	Muti-Class	NoRBERT	0.825	0.88	0.845
(Osamah AlDhafer, 2022)	Muti-Class	BiGRU	0.82	0.8	0.797

2 Design and Methodology

This study contributes to the existing benchmark of the FR vs NFR problem in Table 1 by fine-tuning pre-trained transformers, further advancing the research presented in (Tobias Hey, 2020) (Hanisch, 2020). Requirements typically contain propriety information, which has greatly limited the amount of open source datasets available. Additionally, different organizations have different processes and styles for eliciting requirements. We present a methodology for the development of no-feature classifier models tailored towards the specific requirements of an organisation. We present a binary multilabel problem, with the objectives of categorising requirements according to its type (FR vs NFR) and discipline (Construction, Railway) in an industrial railway and construction integration project following the CRISP-DM Process and release the models source code and weights on github. It is not straightforward to postulate if a false negative is costlier than a false positive. In the case of the FR vs NFR problem, the first case would imply misclassifying a FR as a NFR, introducing a gap in the technical specifications. However conversely flagging a NFR as a FR could lead to misinterpretation and potentially delivering unverifiable evidence. A similar argument could be made for the Discipline label. As such, the best model will be considered on the F1 score, balancing both precision and recall.

We employ encoder with classification head classifiers and generative decoder styles trained to act as classifiers. The motivation for the latter is that new functionality can be introduced by simply training on more data without modifying the architecture of the model, such as generating decomposition of requirements, suggesting wording improvements, generating test routines, identifying conflicting requirements etc.

2.1 Dataset Creation

The vast majority of studies have revolved around the following publicly available datasets: PROMISE (Cleland-Huang, 2006) and CONCORDIA RE. (Feng-Lin Li, 2014) re-labelled the PROMISE Dataset to include quality tags, alongside the NFR category but did not make the dataset publicly available. Further labelling of the PROMISE dataset was performed by (Hanisch, 2020) using crowdsource.
Our dataset was created by concatenating information extracted from an IBM DOORs database using DXL, a scripting language with syntax similar to C++ on a subset of data which contains live and validated information and outputting it into a csv. DOORS has become the industry standard for requirements management in many industries. Whilst it has no API capabilities the scripting allows for automated information retrieval via a scheduler which can then be fed into a data repository, enabling the possibility of deploying a classifier in production and continuously feeding real-time requirements.
After cleansing the data, we present 14911 unique samples, each labelled with "Object Type" and "Discipline". The Object type has the classes "Information" and "Requirement", and are interchangeable with NFR and FR respectively. The 2nd label also contains 2 classes, pertaining to the discipline of the given requirement. We then discretise these values, the distribution is summarised in Figure 1.

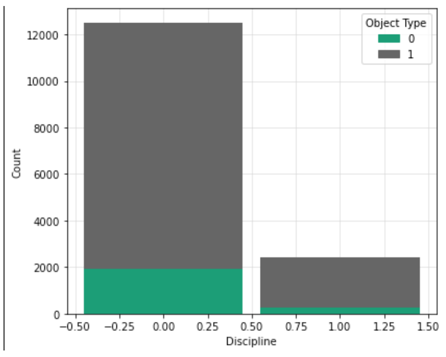

Figure 1: Dataset Distribution

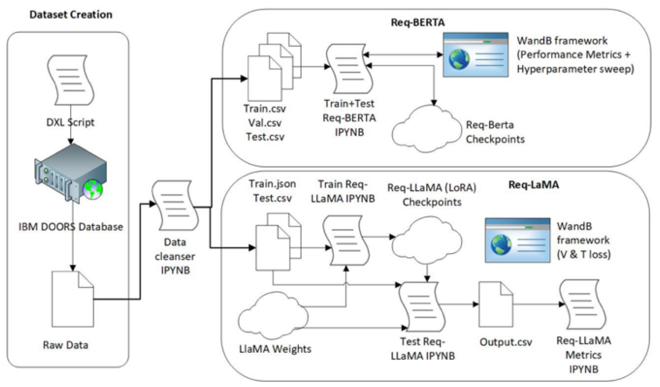

Figure 2: Key scripts

We use 30% stratified split for testing. The implementation of Req-BERTA expects 3 csv: training, validation and testing, so we take a further 20% of the 70% for validation. Req-LLaMA script splits the input json into validation taking 2000 samples, and as such only requires a pre-split for testing. We ensure that in all the subsets there is no data leakage. Figure 2 summarizes the dataflow of the dataset distribution in Table 2.

Table 2: Dataset sample split

Dataset	Req-BERTA	Req-LLaMA
Train	9542	10436
Validation	2386	-
Test	2983	4474

2.2 Req-BERTA: Architecture Design

Since the release of BERT (Jacob Devlin, 2018), several variations have been developed to improve on either predication metrics (Zhilin Yang, 2019) (Yinhan Liu, 2019) or computational speed (Victor Sanh, 2019). Whilst long inference times are undesirable, for our case it is preferable to maximize performance. XLNet (Zhilin Yang, 2019) improves the training procedure by introducing randomisation in the order of token predictions, which improves bidirectional relationship learning, increasing the model capabilities to learn word relationships. Additionally, XLNet was trained on 130 GB of textual data on a TPU processor, which is substantially more than the 16 GB of data that BERT was trained on. RoBERTA (Yinhan Liu, 2019) used even more data than XLNet, 160GB. The batch size was increased from 256 (used in XLNet and BERT) to 8000, which helped increase the

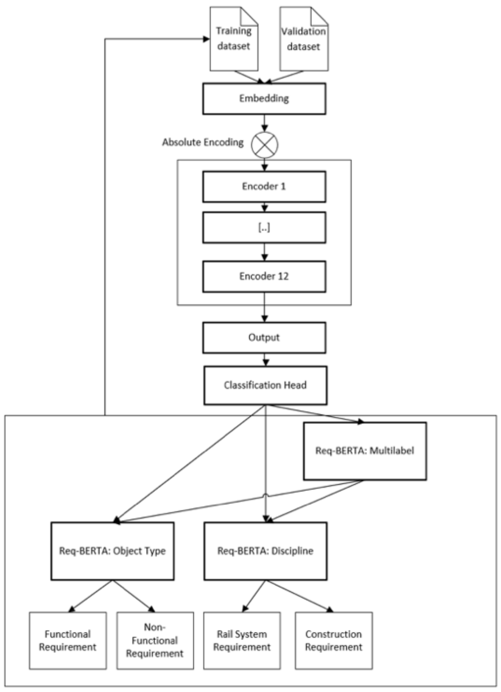

Figure 3: Architecture design of Req-BERTA

training routine from the increase in data and importantly aided in the model generalisation. Additionally, the masking of tokens in RoBERTA is dynamic, and changes during each training epoch. This offers a 2-20% task improvement performance when compared to the BERT baseline (Zhilin Yang, 2019) and exceeds the performance of XLNet. We largely base Req-BERTA on the architecture design of RoBERTA base, with 125 million parameters. We trained 3 models with the configuration described below on 3 different targets: Object Type Classifier, Discipline Type Classifier and a Multilabel Classifier (concatenating Object Type and Discipline).

Activation Function. As with BERT (Jacob Devlin, 2018) Req-BERTA uses the Gaussian Error Linear Unit (GELU) activation function, which has a very similar profile as ReLU but presents a smoother and more continuous profile. ReLU can encounter challenges with large neural networks where a notable proportion of neurons become zero and have minimal practical impact.

$$GELU(x) = x\phi(x) = x \cdot \frac{1}{2}\left[1 + \operatorname{erf}\left(x/_{\sqrt{2}}\right)\right] \quad (1)$$

Model dimensions & dropout rate. We employ a dropout rate of 0.1 for the 12 attention heads (each with a size of 64) and the fully connected layers of the embeddings, encoder and pooler. The dropout rate refers to the probability at which outputs of the layer are retained and is an effective technique to prevent overfitting. The hidden size of the encoder and pooler layers are 768, with the feed-forward encoder layer having a size of 3072.

Tokenizer and Masking. We use a byte-level BPE as a tokenizer (same as GPT-2) and employ dynamic masking over 4 epochs, which forces the model to learn a higher-level of abstraction when compared to the BERT training routine. Larger sequence length will allow longer requirements to be processed before truncating them, however they require higher RAM and increased training time. We managed to reduce the maximum sequence length to 128 and found no requirements were truncated in the dataset.

Table 3: Architecture configuration of Req-BERTA

Parameter	**Description**	**Value**
`hidden_act :`	Non-linear activation function in encoder and pooler.	"gelu"
`attention_probs_dropout_prob:`	Dropout rate for attention probabilities.	0.1
`hidden_dropout_prob:`	Dropout rate of the FC layers in the encoder, embeddings and pooler.	0.1
`Hidden_size`	Size of the encoder layers and the pooler layer.	768
`intermediate_size`	Size of the feed-forward Encoder Layer.	3072
`layer_norm_eps`	Layer Norm epsilon.	1e-05
`max_position_embeddings`	Maximum sequence length of tokens	128
`num_attention_heads`	Attention heads in each attention layer in the encoder.	12
`num_hidden_layers`	Hidden layer on the encoder.	12
`positional_embedding_type`	Type of the positional embedding.	"absolute"

Batch Size and Learning Rate. Contrary to RoBerta, we use smaller batch sizes to finetune Req-BERTA due to the limited size of our dataset, and employ a constant validation batch size of 8. Larger batch sizes perform better with high learning rates, which makes the case for favouring lower learning rates, of the order e-5.
It is highly desirable to have a learning rate warm-up stage. This slows down the hyperparameter tuning process but greatly dictates the final performance of the model (Ruibin Xiong, 2020) . The learning rate is linearly increased over 6% of the total training steps until it reaches the nominal value and employs a maximum gradient clipping of 1 to avoid exploding gradients. Afterwards the learning rate is decreased to a minimal value of 1e-7.

Optimiser. With the ever-expanding body of optimisation literature the selection of optimisers is often driven by anecdotal information. In fact, many optimisers that boast improved performance fail to deliver significant benefits beyond the specific tasks they were originally tested on. (Robin M. Schmidt, 2021) compared fifteen popular routines and found that Adam remains a strong contender. However, despite adaptive gradient optimisers -such as Adam- being the default choice of many studies, Stochastic Gradient Descent (SGD) with momentum are still being used for recent studies yielding state-of-the-art results (Ekin D Cubuk, 2018). However, (Hutter, 2019) found that Adam optimiser with L2 regularisation was able to compete with SGD with L2 regularisation by applying weight decay, all while training much faster. We investigate AdamW.

A summary of the routine used is presented in Equation 2, the full algorithm can be found in (Hutter, 2019).

$$\theta_{t+1,i} = \theta_{t,i} - \alpha \left(\frac{\beta_1 m_t + (1-\beta_1)\left(\nabla f_t + w\theta_{t,i}\right)}{\sqrt{\frac{v_t}{1-\beta_2}} + \in} \right), \forall t \tag{2}$$

Where θ_t are the weights at time t, $\alpha =$ learning rate, $\beta_1 = 0.9, \beta_2 = 0.9999, \in = 10^{-8}, w$ is the weight decay , f_t is the loss function and m_t is the momentum vector.
Another strong contender is Adafactor (Noam Shazeer, 2018) - also an Adam based stochastic optimiser. Adafactor replaces the exponentially smoothed squared gradients with a low-rank approximation, reducing the memory requirements from O(nm) to O(n+m) and removes momentum. To compensate for the instability caused by taking momentum away, the decay rate is increased with time and gradient clippings are used. The other significant difference Adafactor presents is that the learning rate is multiplied with a relative step size.

Loss Function. Loss functions fit the weights of the model to the given training data via an optimisation agent and compare the target and predicted output values. It is custom to create bespoke functions tailored towards specific tasks, depending on the nature of the problem, however for a binary classification task, the log-likelihood is a simple and robust algorithm that fits our purpose. The log-likelihood takes as input the probabilities assigned to the correct labels and applies a log, turning products into summations as illustrated in Equation 3:

$$log\mathbb{P}(\mathcal{D}|\theta) = \sum_{i=1}^{n} (y_i \log \hat{y}_{\theta,i} + (1-y_i)\log(1-\hat{y}_{\theta,i}) \tag{3}$$

Where the predicted probability of i being positive and i being negative are $\hat{y}$ and $(1-\hat{y}_{\theta,i})$ respectively. Since minimizing the log likelihood is equivalent to maximizing the likelihood, the Negative Log Likelihood loss may be written as:

$$l(\theta) = -\sum_{i=1}^{n} (y_i \log \hat{y}_{\theta,i} + (1-y_i)\log(1-\hat{y}_{\theta,i}) \tag{4}$$

Or in a simplified notation as:

$$H(P^*|P) = -\sum_{i} P^*(i)\log P(i) \tag{5}$$

Where i is the total number of classes, $P^*(i)$ is the true class distribution and $P(i)$ is the predicted class distribution. We use the torch implementation of Cross Entropy Loss, the implementation of which is slightly different from the Negative Log Likelihood Loss. `CrossEntropyLoss()` applies a softmax activation before the log transformation and as such expects raw prediction values while `NLLLoss()` take as input the log probabilities. Our case is the multilabel binary class problem, and a viable strategy is to treat the problem as a multiple binary classification problem. If the case were multiclass, the same would apply but having a binary classifier per class.

Hyperparameter Tuning For the training batch size, optimiser and learning rates we employ a grid-search optimisation routine, which although costly when compared to other algorithms -such as evolutionary genetic or Bayesian - yields the transparency of studying every permutation when a limited choice of hyperparameters and values are to be studied. In our case we make the following combinations possible via grid-search:

Table 4: Parameters and values used in the grid-search.

Parameter	**Value**
Batch size	[4,8,32]
Learning rate	[1e-5, 4e-5]
Optimiser	[AdamW, Adafactor]

Hardware. We train Req-BERTA on a single Tesla V100 GPU, with 12.7GB of RAM and 16GB of GPU RAM.

Training Time. We create 3 model variants trained for 3 problems: Object Type Classifier, Discipline Type Classifier and a Multilabel Classifier (concatenating Object Type and Discipline). The wall time for each in mins are 65, 60 and 70 mins respectively yielding a total training time of 3:15h, all trained on 4 epochs.

Validation and Model Selection. In our training routine, we perform the evaluation of the model on the validation set in a given step interval, 500, and take a snapshot of the model weights at this given time. Rather than using the model weights at the end of the training routine, the idea of the "Best Model" is to take the weight checkpoints that have the lowest validation loss in the training routine. This is an efficient use of computing resource since it allows to use a weight checkpoint which may have reached a minima but then fails to generalise in future timesteps and may have been discarded otherwise.

Early Stopping: The idea of Early Stopping (ES) is to periodically evaluate the model performance against the validation set and stop the training routine if the model performance does not improve. Although originally developed to prevent model overfitting and is redundant with Best Model selection technique, it is still beneficial to reduce training time and cost. In summary, our stopping routine is as follows:

```
initialize step_counter to 0
initialize best_eval_loss to infinity
initialize consecutive_improvements to 0

while training:
    perform one training step
    increment step_counter by 1

    if step_counter % 500 == 0:
        val_loss = calculate_val_loss(model_weights, validation_dataset)

        if val_loss < best_eval_loss:
            best_eval_loss = val_loss
            save_model_weights(model_weights)
            consecutive_improvements = 0
        else:
            consecutive_improvements += 1

        if consecutive_improvements >= 3 and (best_eval_loss - val_loss) <= 0.005:
            stop_training()
```

Frameworks and Source Code. We use the Weights and Bias framework for performance evaluation. We use an additional `early_terminate` criteria similar to the ES routine mentioned above in the sweep configuration for hyperparameter testing of the WandB framework and employ hyperband which is an order of magnitude faster than standard Bayesian optimisation routines (Lisha Li, 2016). The source code of Req-BERTA is based on the libraries developed originally by Hugging Face and further advanced by SimpleTransformers (Anon., n.d.).

2.3 Req-BERTA: Performance Evaluation

Table 5: Validation Performance of Req-BERTA.

			Object Type									**Discipline**									**Discipline**				
Hyperparameters			**Loss Values**				**Val Performance**					**Loss Values**				**Val Performance**					**Loss Values**				
Optimizer	**B**	**LR**	**T Loss**	**V Loss**	**V Loss min**	**ES**	**auprc**	**auroc**	**Precision**	**Recall**	**F1**	**T Loss**	**V Loss**	**V Loss min**	**ES**	**auprc**	**auroc**	**Precision**	**Recall**	**F1**	**T Loss**	**VLoss**	**Vloss Min**	**ES**	**LRAP**
Adafactor	4	1.E-05	0.001299	0.08888	0.08888	Y	0.99908	0.99476	0.98675	0.98772	0.98724	0.01428	0.09721	0.09721	Y	0.99853	0.99276	0.98735	0.97698	0.98214	0.01035	0.10574	0.09425	Y	0.99853
Adafactor	8	1.E-05	0.04445	0.09038	0.08795	Y	0.99919	0.99519	0.98581	0.98919	0.98750	0.00173	0.08349	0.06619	N	0.99921	0.99585	0.98946	0.98649	0.98797	0.01234	0.09193	0.09193	Y	0.99832
Adafactor	32	1.E-05	0.00174	0.07994	0.07994	Y	0.99812	0.99138	0.98865	0.98428	0.98646	0.00479	0.10225	0.10220	Y	0.9992	0.99574	0.99438	0.97397	0.98407	0.00693	0.09782	0.08007	Y	0.99790
AdamW	4	1.E-05	0.00035	0.06507	0.06200	N	0.99945	0.99667	0.99113	0.98821	0.98967	0.00202	0.10008	0.08464	N	0.99897	0.99468	0.98796	0.98549	0.98672	0.00135	0.07315	0.06708	N	0.99874
AdamW	8	1.E-05	0.16498	0.07989	0.07989	Y	0.99926	0.99549	0.98483	0.98870	0.98676	0.00286	0.09705	0.09705	Y	0.99892	0.99429	0.99085	0.97548	0.98310	0.00195	0.06805	0.06805	N	0.99832
AdamW	32	1.E-05	0.00359	0.06620	0.05682	N	0.99948	0.99694	0.99165	0.99165	0.99165	0.01308	0.12554	0.12550	Y	0.99091	0.97706	0.97578	0.98799	0.98185	0.0073	0.09747	0.09747	Y	0.99895
Adafactor	4	4.E-05	0.00102	0.07726	0.06496	N	0.99925	0.99566	0.98777	0.99165	0.98971	0.00281	0.0966	0.08421	N	0.99905	0.99519	0.99094	0.98549	0.98821	0.00276	0.07485	0.06574	N	0.99832
Adafactor	8	4.E-05	0.01924	0.08355	0.08355	Y	0.99906	0.994	0.98768	0.98428	0.98598	0.0048	0.09831	0.08454	Y	0.99904	0.99518	0.98699	0.98749	0.98724	0.00315	0.09048	0.09048	Y	0.99853
Adafactor	32	4.E-05	0.0014	0.06414	0.06414	N	0.99942	0.99654	0.99213	0.99067	0.99140	0.00412	0.14188	0.14190	Y	0.99845	0.99165	0.99380	0.96296	0.97814	0.00263	0.07635	0.07075	N	0.99832
AdamW	4	4.E-05	0.00115	0.07947	0.07765	N	0.99937	0.99626	0.99161	0.98723	0.98942	0.00036	0.08157	0.07789	N	0.99929	0.9963	0.99343	0.98448	0.98894	0.00213	0.07503	0.0717	N	0.99832
AdamW	8	4.E-05	0.01182	0.06900	0.06900	N	0.99938	0.99627	0.98919	0.98919	0.98919	0.00074	0.10015	0.08378	N	0.99918	0.99578	0.98849	0.98899	0.98874	0.00243	0.07244	0.07244	N	0.99874
AdamW	32	4.E-05	0.00211	0.07138	0.06278	N	0.99927	0.99569	0.99065	0.98919	0.98992	0.0048	0.11732	0.09121	N	0.99895	0.99473	0.98990	0.98098	0.98542	0.00184	0.07114	0.07114	N	0.99832

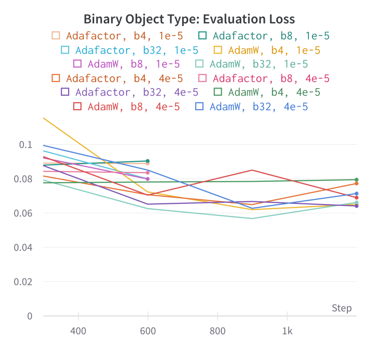

Figure 4: FR vs NFR validation loss.

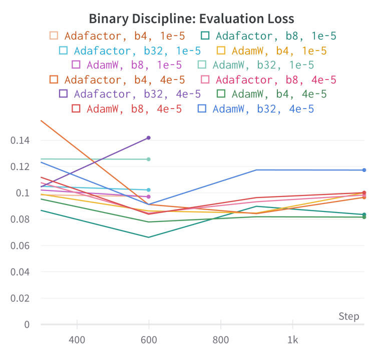

Figure 5: Discipline validation loss.

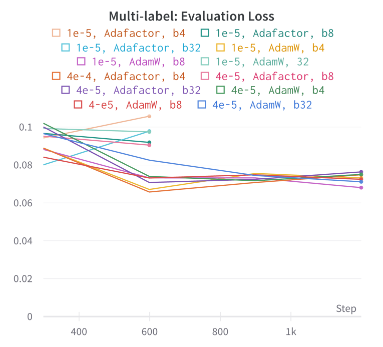

Figure 6: Multilabel validation loss.

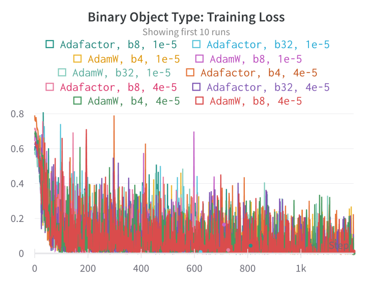

Figure 7: FR vs NFR training loss.

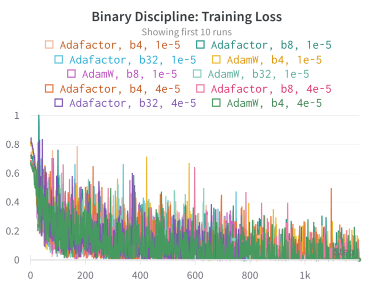

Figure 8: Discipline training loss.

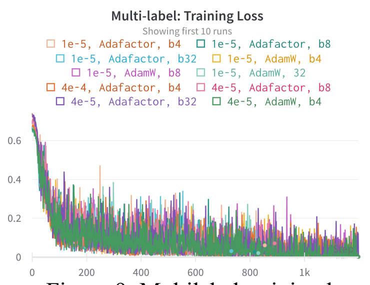

Figure 9: Multilabel training loss.

Table 5 captures the key validation metrics depicted in Figures 4-6 and are summarised in table 6, where we group the training runs into subsets according to the hyperparameters and average the final validation loss to dissect the influence of these parameters. Figures 7-9 illustrate the training loss for each hyperparameter permutation.

Table 6: Average hyperparameter validation loss.

		Average Val Loss		
	Parameters	**Object Type**	**Discipline**	**Multi-label**
Batch Size	**b4**	0.07767	0.09387	0.08219
	b8	0.08071	0.08289	0.08073
	b32	0.07042	0.12175	0.08570
LR	**1.E-05**	0.07839	0.10094	0.08903
	4.E-05	0.07413	0.10597	0.07672
Opt	**AdamW**	0.07184	0.10362	0.07621
	Adafactor	0.08069	0.10329	0.07674

Batch Size. A correlation with batch size and validation loss is observed for the discipline classification, indeed the lower batch subsets have an average lower loss. However, this is not the case for the Object Type model (FR vs NFR problem), where the subset of larger batch sizes is preferred. There are insufficient runs to generalise but one could infer by induction that smaller batch sizes are preferred for semantic analysis (Discipline) whereas larger batch sizes are preferred in syntax problems (Object Type).

Learning Rate. Higher learning rates are preferred for both the Object type and Multi-label models, with minimal difference for the discipline problem. If we consider the Object Type and Multilabel cases, it is preferable to increase the batch size than to decay the learning rate. This echoes the research by (Samuel L. Smith, 2018) who demonstrated that it is preferable to increase the batch size as opposed to decaying the learning rate to obtain the same learning curve when using SGD, Nesterov and Adam.

Optimiser. AdamW performs better with the Object Type and Multilabel cases, whereas the average validation loss of those runs using Adafactor yielded lower loss in the Discipline case.

Runtime. All Req-BERTA models were trained at 1200 steps. For the Object Type problems some runs present divergent behaviour after having found a minima, where the validation loss is not the lowest -and where the best model selection algorithm comes in to play. This indicates that the models would not benefit from training on further steps. However in the Multilabel case there is a strong correlation with `min_val_loss` and the `val_loss` observed at the end of the run, indicating that a the validation loss may have decreased further with a longer training time. This indeed makes sense since the multilabel case amalgamates both the Discipline and Object Type classification problem.

Threats to Validity. Even with the same hyper-parameter choices, due to the random order of the training batches and the initialisation of the classicisation weights the training routine could yield different results in another run.

Best Model Selection and Testing Performance: We select the "Best models" according to the hyperparameters and the checkpoint with the lowest validation loss. The winning configurations are captured in Table 7 with their respective testing performance.

Table 7: Performance on the Test dataset of the models with lowest validation loss:

	Hyperparameters				Confusion Matrix				Test Performance					
Problem	**Optimizer**	**B**	**LR**	**val loss**	**tn**	**fn**	**fp**	**tp**	**auprc**	**auroc**	**Precision**	**Recall**	**F1**	**LRAP**
Object Type	AdamW	32	1.E-05	0.10067	407	53	30	2493	0.99723	0.98777	0.98811	0.97918	0.98363	*
Discipline	Adafactor	8	1.E-05	0.12772	427	60	48	2438	0.99625	0.98264	0.98069	0.97598	0.97833	*
Multi-label	Adafactor	4	4.E-05	0.10885	*	*	*	*	*	*	*	*	*	0.99598

The F1 score indicates all models have generalised successfully with a very competitive precision and recall, adequate for the task in an industrial environment. The presented architectures beat the presented accuracy benchmark of the FR vs NFR problem which are the only ones known to the author to date. The F1 scores are exceeded only by (Ishrar Hussain, 2008) who used DT to maximize recall. In the case presented, we promote the use of the transformer architecture over Decision Trees as specified in the introduction.

2.4 Req-LLaMA: Architecture Design

Req-LLaMA is a generative decoder transformer trained to act as a classifier, which enables the possibility of generating additional text other than the labels, enabling new functionality with no architectural changes. We use the libraries developed by Lightning AI, lit-LLaMA released under Apache 2.0. Largely based on LLaMa (Meta AI, 2023), the following differences are presented from (Vaswani, 2017):

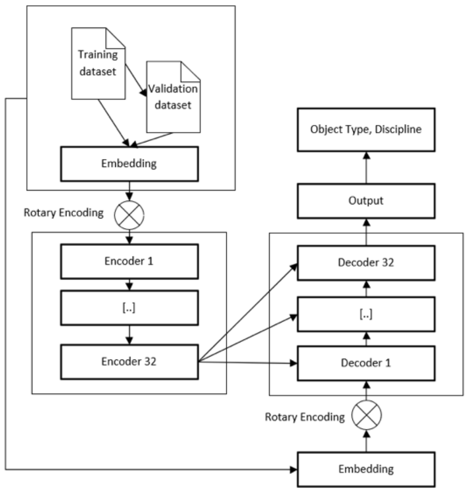

Figure 10: Architecture of Req-LLaMA

Normalization. The input of each transformer layer is normalized using Root Mean Square Normalisation RMSNorm function as introduced in (Biao Zhan, 2019) to improve training stability, instead of normalizing the output. GPT-3 (T.B. Brown, 2020) pioneered this in Large Language Models. This is implemented in (`RMSNorm class`), just as with the original LLaMA to help stabilizing the training process and improving the overall performance of the model. It implements the normalization operation by dividing the input tensor by the root mean square of its elements along a specified dimension.

Activation Function. The cutting-edge Swish-Gated Linear Unit (SiLU) activation function (Shazeer, 2020) is used instead of the traditional ReLU. PaLM (Google, 2020) proved the benefits in a LLM application and has gained popularity since. We apply the SiLU function in the forward pass element wise, and is defined in the `MLP` class implemented as `x = F.silu(self.c_fc1(x)) * self.c_fc2(x).`

Positional encodings. Rotary positional embeddings (RoPE) (Jianlin Su, 2021) are favoured from absolute positional embeddings in all layers of the model. RoPE allows flexible sequence length, decaying inter-token dependency with distance, and enhances linear self-attention with relative position encoding. These encodings were also used in GPTNeo (Sid Black, 2022). We implement RoPE in `build_rope_cache` and `apply_rope` function. `build_rope_cache` takes as inputs the number of elements per head and generates a cache tensor with cosine and sine values based on the position index and theta values. The cache is then used in the `apply_rope` function which transposes the input tensor, selects the relevant rope cache based on the input tensor's size, performs element-wise multiplication and addition operations between the input tensor and rope cache, and finally returns the modified tensor.

Loss Function: LLaMA (Meta AI, 2023) was originally trained using a cosine learning rate schedule to make the final learning rate 10% that of the maximal learning rate. Following the same reasoning behind the Req-BERTA design, we use the cross-entropy loss function for each data batch. This is calculated between the predicted logits and the target labels by first removing the last token from the logits and shifting the targets by one position to match the prediction. We use the torch module `torch.nn.functional.cross_entropy.`

Base Model Selection. From the base LlaMA models a number of fine-tuned models have been created, the most notable ones being Vicuna 13B (LMSYS, 2023) which was finetuned on 70K ShareGPT conversations and demonstrated superior performance to the base model LlaMA 13B as well as achieving comparable performance to GPT-4 (OpenAI, 2023) for Q&A tasks. Stanford Alpaca (Stanford University, 2023) developed a finetuned model of LlaMA using a 52K instruction-following demonstrations generated from OpenAI's text-davinci-003 with supervised finetuning, a method reminiscent of DistilBERT.

Table 8: Architecture summary of LLaMA based on number of Parameters.

LLaMA	**Model hyper parameters**					
n Parameters	**Dimension**	**n heads**	**n layers**	**Learning rate**	**Batch size**	**n tokens**
7B	4096	32	32	3.00E-04	4M	1T
13B	5120	40	40	3.00E-04	4M	1T
33B	6656	52	60	1.5.E-04	4M	1.4T
65B	8192	64	80	1.5.E-04	4M	1.4T

LLaMA 7B consists of 32 transformer blocks and 3 fully connected output layers. An observation from Table 8 is that the higher parameter models where trained on half the learning rate whilst keeping the same batch size, contrary to the advice in (Samuel L. Smith, 2018) where they recommend increasing the batch size as opposed to decaying the learning rate for improved training time. In this case it may become too expensive to increase the batch size further due to memory constraints, but arguably a lower learning rate could imply more power consumption due to increased training time. However, the impact this may have on generalisation might be more considerable when scaling these models.
Meta AI granted us access to the pre-trained weights of LLaMA. The reasoning for the selection of model weights is greatly dictated by our dataset. (J. Kaplan, 2020) provided an empirical study of scaling LLMs based on the Transformer architecture when using cross-entropy loss. His conclusion is that validation loss scales as a power-law with the number of parameters N, dataset samples D and the computing resources C used during training. Importantly, performance improves as long as N and D are scaled proportionally, but presents diminishing return in the case of N or D held fixed while other increases. Larger models also reach the same level of performance with less optimisation steps than smaller models (J. Kaplan, 2020). When working with a constrained compute budget, optimal performance is attained by training very large models and stopping short of convergence, recommending to avoid training to the lowest loss. However we found that larger modes require increased system RAM and GPU RAM requirements, which can become more costly even when training on shorter times.
For large models trained with a limited dataset with early stopping:

$$L(D) = \left({}^{D_C}/_{D}\right)^{\alpha_D} \tag{6}$$

Where $\alpha_D \sim 0.095$ $D_C \sim 5.4 \times 10^{13}$, with $D = 10436$. Our case is a common situation, where D is restricted, which makes the model with lowest number of parameters desirable, 7B. We follow a similar approach as with Alpaca (Stanford University, 2023) for the model selection, since higher parameter models require more computational resources. Worth recalling that that performance is not increased by only scaling model complexity but rather feeding more high-quality data (Deep Mind, 2022).

Finetuning routine. A number of mechanisms exists on fine-tuning large language models, the most prominent being LoRA (Edward Hu, 2021) and Adapters. Low-Rank Adaptation LoRA (Edward Hu, 2021) was proposed to overcome the computational expense of independent instances of LLM tuned for specific applications. Broadly speaking, LoRA injects new rank matrices to the existing weights. The pre-trained weights are frozen as to preserve the model capabilities, and the new lighter LoRA matrices are the only parameters updated in the training process. This prevents catastrophic forgetting, reduces training time since the new LoRA matrixes are much lighter than the original model weights (W), making the model more portable for production considerations. The weight update in a normal training routine would be calculated during backpropagation as the product of the learning rate and the negative gradient of the loss function:

$$\Delta W = \alpha(-\nabla H) \tag{7}$$

$$W' = W + \Delta W \tag{8}$$

Where W' : final Weights, W: Original Weights, ΔW: Weight Change, α: Learning Rate, H: loss function.
Suppose the model weights are kept separate and we calculate the output as:

$$h = Wx + \Delta Wx \tag{9}$$

where h: Output, x: Input. Full rank matrices are those that contain no linearly dependant rows or columns. In contrast, Low Rank Matrices contain redundant rows or columns. The weights of a pretrained model have full rank on the pretrained tasks, however the LoRA authors demonstrated that when pretrained large language models are adapted to a new task the matrices show a lower rank arguing these could be represented in a lower dimensional space without losing information. Suppose a weight update ΔW for $W \in \mathbb{R}^{A \times B}$. ΔWcan then be decomposed into:

$$\Delta W = W_A W_B \tag{10}$$

Where: $W_A \in \mathbb{R}^{A \times r}$,$W_B \in \mathbb{R}^{B \times r}$, r is the rank dimension of the LoRA matrices. Higher r allows more features in the matrix at the expense of computational cost and is a trade-off between model complexity, adaption capability and the risk of over training or under training the model.
This procedure was tested on GPT3 175B by the LoRA authors (Edward Hu, 2021) and was found to reduce the number of trainable parameters by 10000 times and GPU memory requirements by 3 times, whilst achieving the same performance as finetuning RoBERTA. Although not strictly limited, it is typical to only add Rank decomposition matrices to the attention layers of the transformer, which are sufficient to obtain a good downstream performance (Edward Hu, 2021). LoRA provides no additional inference latency compared to Adapters. As such we explored the LoRA routine for the training.
The stopping criteria for our implementation is dictated by a hard stop on the maximum number of training iterations. The number of epochs can be calculated as: `num_epochs = (max_iters * micro_batch_size) / len(train_data)`

Given that it is often only feasible to train the model a limited amount of times, estimating the right hyperparameters is critical. Below we illustrate the hyperparameter reasoning for configuring 4 different training runs on Req-LLaMA, with Table 9 capturing the values.

Learning Rate. Controls how quickly the model weights are updated to the problem. Lower learning rates increase training time since the model updates are smaller and more epochs (defined by the maximum iterations) are required. Higher learning rates cause the model to train faster but may cause divergent behaviour in the loss function. Increasing learning rate for the shorter training runs is necessary.
Batch Size. Larger batch sizes show a significant degradation in their ability to generalise (Nitish Shirish Keskar, 2017), since large batch sizes do fewer and coarser steps. Higher batches consume more memory but speed up the training process. We experiment with 3 different batch sizes, adjusted to the size of our dataset.
Micro Batch Size. The number of samples processed in parallel within a batch, to enable larger batch sizes without exceeding memory requirements.
Weight decay. Weight Decay is used in the optimiser as a L2. It adds a term to the loss functional to penalize large weights, preventing overfitting and helping the model generalise better. We experiment the effect this has by removing it entirely for the shorter runs.
LoRA Rank. The rank of the LoRA parameters. Smaller ranks leads to a simpler low-rank matrices, leading to faster training but reduces the capacity of the model to capture task specific intricacies.
LoRA Alpha. The scaling factor applied to the rank decomposition matries during the forward pass of the model. Controls the magnitude of the rank decomposition matrices contribution to the final output layer and the relative importance between the LoRA weights and the original weights. A higher alpha can help generalisation but a too high value may be underfit.
LoRA Dropout. Dropout can help generalisation but excessive dropout can lead to underfitting.
Evaluate Interval. Frequency at which the model's performance is evaluated on the validation dataset. It specifies the number of training iterations between consecutive evaluations. We do not employ cross-fold validation to speed up the training runs.
Evaluate Iteration. Number of evaluation iterations performed during each evaluation step. It determines how many batches of data are used for evaluation. Cross-validation fold is not used to avoid increasing the training time.
Hardware. We trained Req-LLAMA on a Tesla A100 GPU, with 83.5 GB of system RAM and 40GB of GPU RAM.
Training Time. Run 1: 1.24h , Run 2: 4.91h, Run 3: 2.53h, Run 4: 4.87h, with a total training time of 13.5h.

Table 9: Training Hyper-parameter configurations for Req-LLaMA

Hyper-parameters	**Run 1**	**Run 2**	**Run 3**	**Run 4**
`eval_interval`	100	100	100	100
`eval_iters`	100	100	100	200
`learning_rate`	1e-4	1e-4	1e-5	5e-5
`batch_size`	128	64	32	32
`micro_batch_size`	4	4	4	8
`gradient_accumulation_steps`	32	16	8	4
`max_iters`	37500	112500	56250	56250
`weight_decay`	0.0	1e-5	1e-5	5e-5
`max_seq_length`	256	256	256	256
`lora_r`	8	10	10	8
`lora_alpha`	16	14	14	16
`lora_dropout`	0.05	0.005	0.05	0.05
`Warmup steps`	100	100	100	100

2.5 Req-LLaMA: Performance Evaluation

The validation losses in all cases are considerably higher compared to the Req-BERTA classifier. Transformers with decoders -generative in nature- present higher loss values when compared to Transformers that have only classification heads instead of decoders, as we did with Req-BERTA. Indeed, the original LLaMA lowest validation loss presented was shy of 1.5 (Meta AI, 2023) . We are interested in the relative lowest validation loss, worth noting again that cross-fold validation was not employed to speed the training runs and may have been desirable if a higher computing budget was available. Another limiting factor with Req-LLaMA is that it is trained to act as a multilabel classifier. Although we present the performance metrics for each label, it is not possible to use one model over another for a particular label without having to initialise the model. We deem both labels equally as important and as such present the average F1 from both labels.

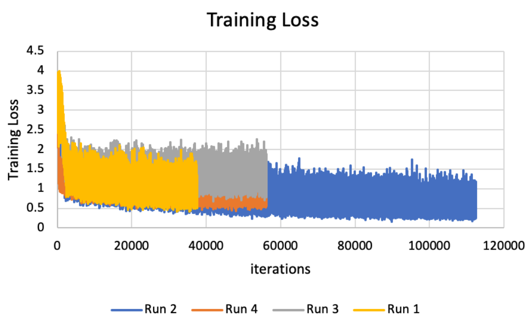

Figure 11: Req-LLaMA Training Loss

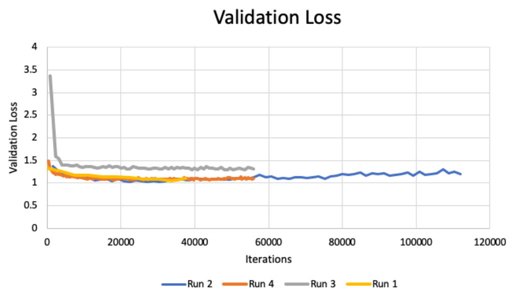

Figure 12: Req-LLaMA Validation Loss

Table 10: Req-LLaMA performance on the test dataset

						Test Dataset						
						Object Type			Discipline			Total
	T Loss	T Loss min	V Loss	V Loss min	Hallucination	Precision	Recall	F1	Precision	Recall	F1	Average F1
Run 1	0.84990	0.16423	1.09338	1.05585	0.10259	0.985244	0.985244	0.985244	0.961908	0.990716	0.976099	0.980672
Run 2	0.68444	0.43815	1.19669	1.02242	0.10796	0.983419	0.979778	0.981595	0.964004	0.991571	0.977593	0.979594
Run 3	1.83339	0.66965	1.31563	1.28779	0.13544	0.947016	0.992692	0.969316	0.908124	0.954066	0.930528	0.949922
Run 4	0.90858	0.41520	1.10386	1.06351	0.12116	0.986721	0.985298	0.986009	0.962097	0.992061	0.976849	0.981429

Results Discussion. The model outputs a sequence of words corresponding to the predicted 2 labels separated by a comma. Since Req-LLaMA have decoders and contain all the underlying information and capabilities of the original LLaMA, there is nothing preventing the generated text to be different than the expected 2 labels for Object Type and Discipline other than the LoRA parameters. However, during inference, an important hyperparameter is the temperature, which adjusts the probability of the logits before the activation function. It dictates the randomness and creativity of the generated text, between 0.1 and 2, 1 corresponding to the standard activation function (SiLU). Since we are training a generative transformer to act like a classifier, it is desirable to decrease the temperature to prevent the transformer hallucinating any text other than the labels. Despite the efforts to do so, over 10% of outputs had to be discarded. The metrics were calculated in a separate notebook to clean the outputs (such as discarding the hallucinated outputs and separating the predicted labels to compare them against the actual labels). This percentage is included in Table 10. Note that the hallucinated percentage was not considered when calculating Precision, Recall and F1 scores and are presented separately.
Run 1 is one of the shortest runs and as such uses the highest learning rate and batch size. It was the most computationally efficient optimally trained run, achieving an overall F1 score of 0.981.
Run 2 was trained with the highest amount of iterations. Whilst the training loss keeps decreasing, divergence is observed in the validation loss (Figure 12), indicating an overfitted model. Indeed, optimal performance is attained in very large models by stopping short of convergence (J. Kaplan, 2020). Run 2 presents the highest F1 score for the Discipline category and achieves the minimum validation loss in the saved checkpoint. This would be the recommended model if Discipline classification were the only task in question, but computational resources could have been saved with a shorter training run.
Run 3 employed one of the highest LoRA Rank and Alpha parameters, 10 and 14 respectively. This should have increased the capacity of the model to learn task specific intricacies, but the test results indicate an undertrained model, with over 13% of predictions having to be discarded and presenting the lowest F1 score, greatly attributed to the low learning rate. This is visually discernible in Figure 11 and 12.
Run 4 uses one of the lowest batch sizes of 32 (just as with Run 3). Although not the lowest validation loss of the runs, it achieves the highest F1 score for the Object Type classification, as well as the highest average F1 score, making it the recommended model for production.
Although we used the lowest validation loss in Req-BERTA as an indicator of best model this correlation does not hold with Req-LLaMA since the performance metrics are calculated having removed the hallucination events.

3 Conclusion

This study has demonstrated that fine-tuning approaches for the transformer architecture are highly accurate and require no feature engineering. We have beaten the benchmark for the FR vs NFR problem with Req-BERTA and demonstrate that larger models are not always desirable given a dataset. Future work will involve investigating other LoRA parameters to reduce the hallucination of Req-LLAMA and extending the functionality, capitalising on the textual generation capability. We release our source code and model weights for the community to further develop and build on Req-LLaMA and Req-BERTA in `https://github.com/alanpaddy/Req-LLaMa`

4 References

Abderahman Rashwan, O. O. R. W., 2013. Ontology-Based Classification of Non-Functional Requirements in Software Specifications: A new Corpus and SVM-Based Classifier. *2013 IEEE 37th Annual Computer Software and Applications Conference.*
Accenture, 2017. *Detecting Vague Words & Phrases in Requirements Documents in a Multilingual Environment.* s.l.:s.n.
Alex Dekhtyar, V. F., 2017. RE Data Challenge: Requirements Identification with Word2Vec and TensorFlow. *2017 IEEE 25th International Requirements Engineering Conference.*
Alhindawi, N. T., 2018. Information Retrieval - Based Solution for Software Requirements Classification and Mapping. *2018 5th International Conference on Mathematics and Computers in Sciences and Industry (MCSI).*
Anon., n.d. *Simple Transformers.* [Online]
Available at: https://simpletransformers.ai/
B. Rosadini, A. F. G. G. A. F. S. G. I. T. a. S. B., 2017. Using NLP to Detect Requirements Defects: An Industrial Experience in the Railway Domain. *Requirements Engineering: Foundation for Software Quality, Refsq,* p. 344.
Benedikt Gleich, O. C. K., 2010. Ambiguity Detection: Towards a Tool Explaining Ambiguity Sources. p. 228.
Berry, D. K. E. K. M., 2003. *From contract drafting to software specification: Linguistic sources of ambiguity.* s.l.: http://se.uwaterloo.ca/̃dberry/handbook/ambiguityHandbook.pdf.
Biao Zhan, R. S., 2019. *Root mean square layer normalization.* s.l.:Advances in Neural Information Processing Systems.
Cleland-Huang, J., 2006. *The Detection and Classification of Non-Functional Requirements.* s.l.:s.n.
Cody Baker, L. D. S. C. J. D., 2019. Automatic Multi-Class Non-Functional Software Requirements Classification using Neural Networks. *IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC).*
Deep Mind, 2022. *Scaling Language Models: Methods, Analysis & Insights from Training Gopher.* s.l.:s.n.
Deep Mind, 2022. *Training Compute-Optimal Large Language Models.* s.l.:s.n.
Edward Hu, Y. S. P. W. Z. A.-Z. Y. L. S. W. L. W. W. C., 2021. *LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS.* s.l.:s.n.
Ekin D Cubuk, B. Z. D. M. V. V. a. Q. V. L., 2018. *Learning augmentation policies from data..* s.l.:s.n.
F Fabbrini, M. F. S. G. G. L., 2001. The Linguistic Approach to the Natural Languare Requirements Quality: Benefits of teh use of an Automatic Tool.
Fabiano Dalpiaz, D. D. F. B. A. S. Ç., 2019. Requirements Classification with Interpretable Machine Learning and Dependency Parsing. *2019 IEEE 27th International Requirements Engineering Conference (RE).*
Feng-Lin Li, J. H. J. M. R. S. S. G. G. G. A. B. L. L., 2014. Non-functional Requirements as Qualities, with a Spice of Ontology.
Google, 2022. *LaMDA: Language Models for Dialog Applications.* s.l.:s.n.
Hanisch, L., 2020. Detecting Vague Requirements with Machine Learning. *Technical University of Munich, Department of Informatics.*
Henning Femmera, ,. D. M. F. S. W. S. E., 2017. Rapid Quality Assurance with Requirements Smells.
Hutter, I. L. &. F., 2019. *DECOUPLED WEIGHT DECAY REGULARIZATION.* s.l.:s.n.
Ishrar Hussain, L. K. a. O. O., 2008. Using Linguistic Knowledge to Classify Non-functional Requirements in SRS documents. *13th International Conference on Applications of Natural Language to Information Systems,* p. 296.
J. Kaplan, S. M. T. H. T. B. B. B. C. R. C. S. G. A. R. J. W. a. D. A., 2020. *Scaling laws for neural language models.* s.l.:s.n.
J. M. Lanza-Guiterrez, A. A. F. J. P., 2017. The importance of requirements quality in the performance of manufacturing firms. *Journal of Manufacturing Systems,* pp. 1-10.
Jacob Devlin, M.-W. C. K. L. K. T., 2018. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* s.l.:s.n.
Jane Cleland-Huang, R. S. X. Z., 2007. Automated classification of non-functional requirements. *Springer-Verlag London Limited.*
Jianlin Su, Y. L. S. P. A. M. B. W. Y. L., 2021. *Enhanced transformer with rotary position embedding..* s.l.:s.n.
Jonas Winkler, A. V., 2016. Automatic Classification of Requirements Based Convolutional Neural Networks. *2016 IEEE 24th International Requirements Engineering Conference Workshops.*
Klaus Pohl, C. R., 2009. Requirements Engineering: A roadmap.
Lisha Li, K. J. G. D. A. R. A. T., 2016. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimisation.* s.l.:s.n.
LMSYS, 2023. [Online]
Available at: https://lmsys.org/blog/2023-03-30-vicuna/

Manal Binkhonain, L. Z., 2019. A review of machine learning algorithms for identification and classification of non-functional requirements. *Elsevier.*
Md. Abdur Rahman, M. A. H. M. N. A. T. M. S. S., 2019. Classifying Non-functional Requirements using RNN Variants for Quality Software Development.
Meta AI, 2023. *LLaMA: Open and Efficient Foundation Language Models.* s.l.:s.n.
Muhammad Younas, D. N. A. J. K. W. M. A. S., 2019. An Automated Approach for Identification of Non-Functional Requirements using Word2Vec model.
Nitish Shirish Keskar, D. M. J. N. M. S. P. T. P. T., 2017. *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.* s.l.:s.n.
Noam Shazeer, M. S., 2018. *Adafactor: Adaptive Learning Rates with Sublinear Memory Cost.* s.l.:s.n.
OpenAI, 2023. *GPT-4 Technical Report.* s.l.:s.n.
Osamah AlDhafer, I. A. S. M., 2022. An end-to-end deep learning system for requirements classification using recurrent neural networks. *Elsevier.*
Prateek Singh, D. S. A. S., 2016. Rule-Based System for Automated Classification of Non-Functional Requirements from Requirement Specifications. *2016 Intl. Conference on Advances in Computing, Communications and Informatics (ICACCI), Sept. 21-24, 2016, Jaipur, India.*
Raul Navarro-Almanza, R. J.-R. G. L., 2017. Towards supporting Software Engineering using Deep Learning: A case of Software Requirements Classification. *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT).*
Research, G., 2020. *PaLM: Scaling Language Modeling with Pathways.* s.l.:s.n.
Robin M. Schmidt, F. S. P. H., 2021. *Descending through a Crowded Valley — Benchmarking Deep Learning Optimizers.* s.l.:s.n.
Ruibin Xiong, Y. Y. D. H. K. Z. S. Z. C. X. H. Z., 2020. *On Layer Normalization in the Transformer Architecture.* s.l.:s.n.
Samuel L. Smith, P.-J. K. C. Y. Q. V. L., 2018. *Don't Decay the Learning Rate, Increase the Batch Size.* s.l., s.n.
Shazeer, N., 2018. *Adafactor: Adaptive Learning Rates with Sublinear Memory Cost.* s.l.:s.n.
Shazeer, N., 2020. *Glu variants improve transformer.* s.l.:s.n.
Sid Black, S. B. E. H. Q. A. L. G. L. G. H. H. C. L. K. M. J. P., 2022. *Gpt-neox-20b: An open-source autoregressive language model..* s.l.:s.n.
Sousuke Amasaki, P. L., 2018. The Effects of Vectorization Methods on Non-Functional Requirements Classification. *2018 44th Euromicro Conference on Software Engineering and Advanced Applications.*
Sri Fatimah Tjong, D. M. B., 2013. The Design of SREE - Ambiguity Finder for Requirements Specifications and Lessons Learned. *19th International Working Conference, REFSQ 2013,* p. 96.
Stanford University, 2023. *Alpaca: A Strong, Replicable Instruction-Following Model.* s.l.:s.n.
T.B. Brown, B. M. N. R. M. S., 2020. *Language Models are Few-Shot Learners.* s.l.:John Hopkins University, Open AI.
Tobias Hey, J. K. A. K. W. F. T., 2020. NoRBERT: Transfer Learning for Requirements Classification. *Karlsruhe Institute of Technology (KIT).*
Vaswani, A., 2017. *Attention Is All You Need.* s.l.:Google.
Victor Sanh, L. D. J. C. T. W., 2019. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.* s.l.:s.n.
Yinhan Liu, M. O. N. G. J. D. M. J. D. C. O. L. M. L. L. Z. V. S., 2019. *RoBERTa: A Robustly Optimised BERT Pretraining Approach.* s.l.:s.n.
Zahra Shakeri Hossein Abad, O. K. P. G. M. G. G. R. K. S., 2017. What Works Better? A Study of Classifying Requirements. *2017 IEEE 25th International Requirements Engineering Conference.*
Zhilin Yang, Z. D. Y. Y. J. C. R. S. Q. V. L., 2019. *XLNet: Generalized Autoregressive Pretraining for Language Understanding.* s.l.:s.n.
Zijad Kurtanovic, W. M., 2017. Automatically Classifying Functional and Non-Functional Requirements Using Supervised Machine Learning. *2017 IEEE 25th International Requirements Engineering Conference.*