Karlsruhe Institute of Technology

Software Design and Quality

Fine-tuning vs. Prompting: The Case of Requirements Classification

Bachelor's Thesis of

Maximilian Supp

At the KIT Department of Informatics
KASTEL – Institute of Information Security and Dependability

First examiner:	Prof. Dr.-Ing. Anne Koziolek
Second examiner:	Prof. Dr. Ralf Reussner
First advisor:	Dr.-Ing. Tobias Hey
Second advisor:	M.Sc. Dominik Fuchß

17. November 2025 – 17. March 2026

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Fine-tuning vs. Prompting: The Case of Requirements Classification (Bachelor's Thesis)

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Karlsruhe, 17. March 2026

. . . . . .

(Maximilian Supp)

Abstract

Requirements classification is a key task in software development, in which requirements are grouped into different categories, such as functional and non-functional requirements. It helps to prioritize requirements and improve quality of the software. To automate this process, many machine learning approaches have been studied in recent years. With the rise of LLMs, fine-tuning and prompting were evaluated, showing promising results. However, current research does not provide guidelines for when to use which technique.

This thesis aims to address this gap by systematically comparing state-of-the-art fine-tuning and prompting approaches on different application scenarios. For this purpose, three real-world scenarios are examined, covering the varying availability of sample data. Based on that, the goal is to investigate how much training data is required to achieve similar or better results with fine-tuning as with prompting techniques.

When evaluating only on unseen projects, zero-shot prompting often achieves a similar or even better performance than fine-tuning. If requirements from the same project are available for training, 20 to 40 requirements are enough to outperform prompting with fine-tuning. Depending on the task and fine-tuning approach, labeling 20 requirements manually improved the performance by up to 25 percentage points in the best case and often by at least 10 percentage points. The evaluated approaches showed that fine-tuning always achieves a better performance than prompting if enough training data was provided.

This thesis provides guidelines for requirements classification in practice and serves as a foundation for further research regarding this topic.

Zusammenfassung

Die Klassifizierung von Anforderungen ist eine zentrale Aufgabe in der Softwareentwicklung, bei der Anforderungen in verschiedene Kategorien wie funktionale und nichtfunktionale Anforderungen eingeteilt werden. Sie hilft dabei, Anforderungen zu priorisieren und die Qualität der Software zu verbessern. Um diesen Prozess zu automatisieren, wurden in den letzten Jahren zahlreiche Ansätze des maschinellen Lernens untersucht. Mit dem Aufkommen von LLMs wurden Fine-Tuning und Prompting evaluiert, was vielversprechende Ergebnisse zeigte. Die aktuelle Forschung liefert jedoch keine Richtlinien dafür, wann welche Technik eingesetzt werden sollte.

Diese Arbeit zielt darauf ab, diese Lücke zu schließen, indem sie modernste Fine-Tuning- und Prompting-Ansätze in verschiedenen Anwendungsszenarien systematisch vergleicht. Zu diesem Zweck werden drei reale Szenarien untersucht, die die unterschiedliche Verfügbarkeit von Beispieldaten abdecken. Auf dieser Grundlage soll untersucht werden, wie viel Trainingsdaten erforderlich sind, um mit Fine-Tuning ähnliche oder bessere Ergebnisse zu erzielen als mit Prompting-Techniken.

Bei der Bewertung ausschließlich anhand von unbekannten Projekten erzielt Zero-Shot-Prompting oft eine ähnliche oder sogar bessere Leistung als das Fine-Tuning. Wenn Anforderungen aus demselben Projekt für das Training zur Verfügung stehen, reichen 20 bis 40 Anforderungen aus, um das Prompting mit Fine-Tuning zu übertreffen. Je nach Aufgabe und Fine-Tuning-Ansatz verbesserte die manuelle Beschriftung von 20 Anforderungen die Leistung im besten Fall um bis zu 25 Prozentpunkte und oft um mindestens 10 Prozentpunkte. Die evaluierten Ansätze zeigten, dass Fine-Tuning immer eine bessere Leistung erzielt als Prompting, sofern genügend Trainingsdaten zur Verfügung standen.

Diese Arbeit liefert Leitlinien für die Anforderungsklassifizierung in der Praxis und dient als Grundlage für weitere Forschungen zu diesem Thema.

Contents

List of Figures

List of Tables

1. Introduction

Classifying requirements is an essential task in software development. It organizes requirements into categories, creating the foundation for effective requirements engineering. This enables better prioritization, as critical requirements can be distinguished from less urgent ones [1]. Additionally, quality assurance improves [5], since classified requirements, such as performance- or security-related ones, can be systematically checked against predefined standards [22]. Many machine learning techniques have been studied over the past almost 20 years to automate this process, with state-of-the-art methods often being based on LLMs (Large Language Models). These models have shown promise in automating requirements classification, often achieving better results than traditional machine learning concepts, especially in generalization settings [44]. In recent years various approaches have evaluated fine-tuning and prompting strategies for this task to improve the performance of LLMs. While fine-tuning makes it possible to optimize the model by learning domain-specific aspects, it is time-consuming and requires labeled training data [32]. Prompting, on the other hand, can be applied with minimal or even no labeled training data, offering greater flexibility. Now, the question arises, when to use which of the two.

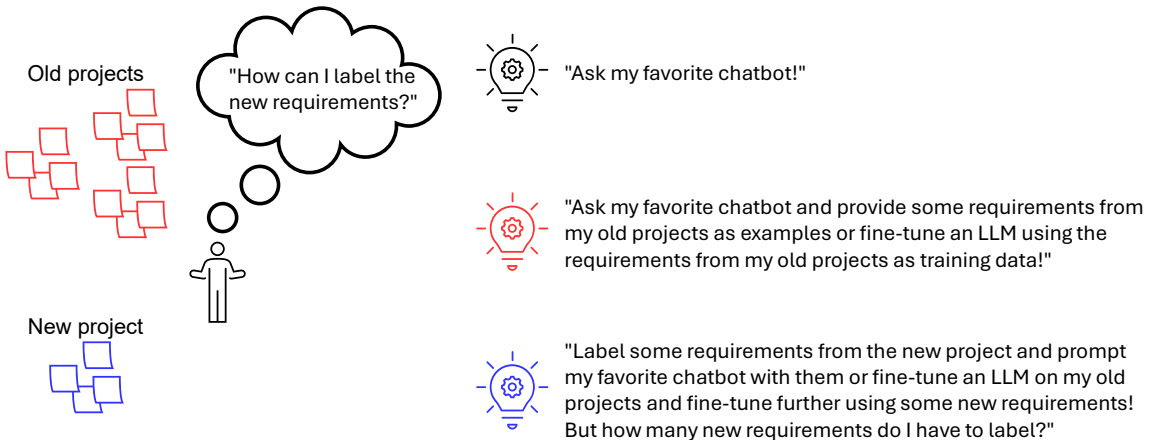

Figure 1.1.: Software engineer considering how to label requirements from a new project when using different amounts and types of additional data.

Figure 1.1 shows a typical challenge a software engineer might encounter when new requirements have to be labeled. There are requirements from a new project, and there may also be some from old projects. The software developer now wants to label the new requirements but is unsure whether to use fine-tuning or prompting. The simplest option would be to ask an LLM-based chatbot like ChatGPT to classify the requirements. In this case, no effort is required to label any data, and this would also be applicable if there were no

old projects at all. The second possibility would involve requirements from the old projects, if existing. They could either be used as examples to put into the prompt for the chatbot or as training data to fine-tune an LLM. For the last option, the software engineer would have to label some requirements from the new project and use them as examples for the prompt or to fine-tune the model created in the second option even further. The question now is how many requirements would have to be labeled in order to achieve the best results depending on the availability of examples/training data?

So far, research has exclusively compared fine-tuning and prompting in specific scenarios, mostly those that did not take into account that requirements originate from different projects or that, in reality, there may be a limited amount of examples/training data available. No research has yet provided guidelines for when to use which strategy. The thesis aims to bridge this gap by systematically comparing fine-tuning and prompting for requirement classification in different application scenarios, which cover the varying availability of examples/training data. The focus is to determine how many requirements have to be labeled in different scenarios. Depending on whether already labeled data from the target project is available, the performance might be different from having to rely exclusively on requirements from other projects or no requirements at all. To achieve this, controlled experiments are conducted to evaluate performance in scenarios that reflect real-world use cases using state-of-the-art approaches. The expected benefit of this work is to provide evidence-based guidelines to select the most suitable strategy for the corresponding scenario. Additionally, the results should offer a comprehensive overview of how LLMs can be effectively adapted for requirements classification using state-of-the-art methods.

Therefore, the thesis is structured as follows: Chapter 2 provides important foundations. Related work, including state-of-the-art approaches, is presented in Chapter 3. Chapter 4 contains detailed information about the mentioned application scenarios and the research questions as well as the experiment design. This also includes information about which approaches were chosen for the experiments and why. The experiment results can be found in Chapter 5. The implication of these results as well as threats to validity are provided in Chapter 6. Finally, Chapter 7 sums up the findings of the thesis.

2. Foundations

Before exploring the related work and the concept of the thesis, it is important to understand the foundations. This chapter provides a definition of requirements classification as well an overview of how LLMs work and how they can be adapted using different fine-tuning and prompting methods. This is followed by foundations about data splitting and performance metrics.

2.1. Requirements Classification

Software requirements specify attributes that a system must fulfill. Requirements classification is the task of categorizing these requirements based on a given classification scheme. The broadest and most commonly used task is classifying a requirement into either functional or non-functional, where the non-functional class is typically further divided into subclasses such as performance, security, and usability. Here, functional requirements specify what a system shall do, while non-functional requirements specify how a system shall do something [14]. It is also common practice to further classify non-functional subclasses, a very popular one being the classification into either security or non-security [28].

However, the distinction between functional and non-functional requirements is debatable [14, 18, 25], as the question arises whether a requirement can be considered purely non-functional as every requirement ultimately contains a function. An alternative classification scheme builds upon the ISO/IEC 25010 standard [1]. This model possesses two aspects: functional aspects, which specify a certain behavior or function, and quality aspects, which describe a basic perceivable or measurable characteristic. A requirement can consist of either one of the aspects, both aspects, or none, with the latter meaning that the requirement describes information. [10]

2.2. Large Language Models

To perform requirements classification with LLMs it is important to understand their basic functionality. LLMs are deep learning models designed to understand and generate natural language. They are trained with millions or even billions of parameters on diverse text (e.g., books and web pages). [12, 41]

[1] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

2.2.1. Transformer Architecture

The Transformer architecture builds the foundation for LLMs and was first introduced in 2017 by Vaswani et al. [40]. It originally consists of two main components: an encoder, which maps an input sequence to a vector representation, and a decoder, which maps such a representation to an output. While the original transformer uses both, modern LLMs typically employ only one component [41].

2.2.1.1. Encoder-Only

Encoder-only models take an input sequence and return a vector representation. The process of mapping the words of a given input sequence to the corresponding vector representation is called word embedding. Its purpose is to map words with similar meanings to similar-looking vectors. Static embeddings map each word of the vocabulary to one vector, which leads to a problem as words can have different meanings based on the context they appear in. An alternative are so-called contextual embeddings, which, in contrast to static embeddings, can create different embeddings for the same word based on all other words occurring in a sentence and therefore include the context the word appears in into the mapping. [41]

An example of an encoder-only model, which uses such contextual embeddings, is BERT (BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS) [12]. Traditionally, language models process the input sequentially, so they start on one side and go until they reach the end [7]. As the name suggests, BERT, however, is bidirectional, which means that the input sequence is processed from both left and right. This enables BERT to understand relations between words even if they are far apart spatially.

2.2.1.2. Decoder-Only (Generative)

Decoder-only LLMs are also called generative LLMs, as they have been designed to generate text in natural language. The input is called a prompt, which contains textual instructions for the LLM. Since the input does not need to be encoded to a vector, the natural language input can be passed directly to the model. This makes it much easier to use for humans than an encoder-only LLM, also because it produces text as output. [41]

One of the most popular examples is OpenAI's GPT (GENERATIVE PRE-TRAINED TRANSFORMER), which uses a left-to-right Transformer. This means that information about a word depends only on the previous but not the following words [12]. Among the first tasks were also classification tasks [30], which are highly relevant for this thesis. GPT also highly influenced the design of BERT, as the designers tried to make their model as close as possible to the GPT model for comparison reasons [12].

2.2.2. Fine-tuning

One option to adapt such LLMs is fine-tuning. This is the process of further training a pre-trained language model on specific data to improve its performance on particular tasks or domains. Fine-tuning changes the internal (model) parameters, including weights and biases, to better align with new data. Therefore, it requires a large amount of additional training data to be effective. Various strategies have been explored to ranging from comprehensive adaptation of all model parameters to resource-saving methods that only modify selected parts of the model. Three of the most important approaches are presented below:

Full Fine-tuning: All parameters of the pre-trained model are being optimized to adapt to the specific task or domain. Although this approach usually results in a great performance it is extremely expensive in terms of resources and time and often is not worth the effort. [35]

Task-Specific Head: This method adds an head layer to the end of the model. The rest remains frozen, while only the extra layer is trained. As only one layer is adapted, this approach can be very efficient. [42]

PEFT (Parameter-Efficient Fine-Tuning): To avoid fine-tuning the whole model, PEFT has been developed by freezing most of the model's parameters and only adjusting a small part of the parameters. Some of the most common methods are: 1. Adapters, which add small networks in between the layers of the pre-trained model and only the parameters of the adapter networks are trained. 2. Prompt-tuning, which concatenates additional prompts with the original input and optimizes those prompts. 3. Low-Rank Adaptation (LoRA), where low-rank matrices are learned instead of updating the full weight matrices. 4. BitFit, which doesn't introduce any additional parameters at all. It just trains the bias terms of the pre-trained layers. [13, 35]

Besides the model parameters that are automatically updated during the training process, there is another type of parameters, called hyperparameters, which define the model's structure [11]. Common hyperparameters include: 1. Maximum sequence length, referring to the number of input tokens the model can process at once. 2. Training epochs, referring to how often the model goes through the dataset. 3. Batch size, specifying the number of training samples used in one iteration. 4. Learning rate, describing how quickly the weights are adjusted during training. 5. Weight decay, referring to a penalty term inserted into the loss function to keep the weights small and avoid an exploding gradient when updating the weights. [32]

2.2.3. Prompting

Another adaption method for LLMs is prompting. Unlike fine-tuning, it does not adapt the model itself. Prompts are the natural language inputs and instructions given to an LLM to generate a response. A prompt can be subdivided into two components: a system prompt and a user prompt. A system prompt is an instruction that defines the general desired behavior of the model. This can include the assignment of a role or information about the

desired output format. A user prompt, on the other hand, contains the task-specific input query. It consists of a description of the task, optionally examples, and the input sequence, which contains the requirement to classify. In the case of requirements classification, this means that the user prompt is different for each requirement to classify, whereas the system prompt stays the same for all user prompts that relate to the same task. [3, 37]

The following techniques apply to a prompt as a whole and do not distinguish between system and user prompts. A basic distinction is made between zero-shot prompting, which doesn't need any additional data, and few-shot prompting, which contains examples of a task to help the model complete a similar task. In addition, it is possible to combine both of the previously mentioned techniques with expansions such as the following ones: Persona prompting, which adds a specific role to offer the model context about its identity and background to generate more in-character responses, and chain-of-thought prompts, where instructions are added for the model to provide intermediate reasoning steps before giving the final answer. When sufficient and high-quality data is available, fine-tuning usually achieves higher and more consistent performance than prompting alone. [5, 15, 34]

2.3. Machine Learning Evaluation

After taking a closer look at requirements classification and LLMs, this section focuses on how to evaluate LLMs for the task of requirements classification.

2.3.1. Classification Types

Depending on the type and number of classes, there are various ways to evaluate the classification. The following explanations assume that a label is 1 for a class if the requirement is part of that class and 0 if it is not. A distinction is made between three main types [44], illustrated with examples from Section 2.1:

Binary classification: A requirement is classified into either one of two classes. This means that the labels for both classes have to be different. In this case, usually one class is selected, and the label is only returned for this class as the other label has to contain the opposite value. A typical task is the distinction of functional and non-functional requirements.

Multi-class classification: A requirement is classified into either one of multiple (more than two) classes. Here, there is only one label allowed to be 1, the rest has to be 0. An example is the classification of all non-functional subclasses.

Multi-label classification: In contrast to the previous two, a requirement can be additionally classified into multiple or none of the available classes. For the case of functional and quality aspects, this means that, if the labels for the functional and the quality class are both 1, the requirement contains functional and quality aspects. If the requirement contains neither of these aspects, both labels would be 0.

Instead of assigning the labels for each class in one classification, the latter two classification types can be substituted by multiple binary classifications. To determine whether a requirement contains only one of the two aspects in regard to functionality and quality, a binary classification can be performed for each aspect. Based on the results, a label can be derived to indicate whether only one of the two aspects is included. If only one label is 1, the requirement contains only aspects of the corresponding class.

2.3.2. Data Splitting

To fine-tune an LLM for any classification task, a set of labeled data is required. The simplest method is to split the data into two disjoint sets: training and testing. The train set is used for training the model, and the test set is used to evaluate the performance. There are approaches using an additional validation set, which is utilized to monitor the performance during training and choose the model configuration that performs best on this set. As the performance depends on the data split and is therefore only a rough estimation of the model's actual capabilities, there is a more robust alternative, which is k-fold cross-validation. Here the data is split into k groups (folds) of similar size. The process of training and testing is repeated k times, where every time one group is used for testing and the remaining k - 1 groups are used for training. Therefore, every group is used exactly once for testing. The process ensuring the label distribution inside the sets is approximately the same as in the original dataset is called stratification. For a classification task, this means that the number of samples belonging to a class should be similar in all sets. [11, 39]

If the data originates from more than one project, it is not only possible to split the entire dataset randomly but also to partition the data using the different projects. This is therefore called p-fold validation. A special case is the LOPO (leave-one-project-out) method, where every fold consists of exactly one project. Depending on how different the sizes of the projects are, it is also possible to combine several projects into one fold. No matter how the projects are partitioned, the procedure for cross-validation remains the same: One partition is used for evaluation and the rest for training. The p-fold approach represents a more realistic scenario to examine how well the results generalize. [11]

2.3.3. Performance Metrics

To assess the performance of LLMs, standard metrics from requirements classification are used. Before defining these metrics, it is essential to introduce the four basic characteristics of a classifier using an example situation where the task is to classify a requirement as either functional or non-functional: In this case a positive classification means that the requirement is functional and a negative classification equals non-functional: 1. TP (True Positive) represents the number of requirements that have been properly classified to be functional. 2. TN (True Negative) represents the number of correctly classified requirements that are non-functional. 3. FP (False Positive) represents the number of requirements that were misclassified as functional, although they are non-functional. 4. FN (False Negative)

represents the number of incorrectly classified requirements as non-functional even though they are functional. [39]

With those basics the metrics are defined as follows [11]:

$$\text{Precision} = \frac{TP}{TP + FP}$$

The ratio of requirements correctly classified as functional (TP) to the total number of requirements classified as functional (TP+FP).

$$\text{Recall} = \frac{TP}{TP + FN}$$

The ratio of requirements correctly classified as functional (TP) to the total number of requirements that are actually functional (TP+TN).

$$\text{F}_\beta\text{-score} = \frac{(1 + \beta^2) \times Precision \times Recall}{(\beta^2 \times Precision) + Recall}$$

The weighted mean of precision and recall. The F_1-score represents the harmonic mean, balancing both metrics. F_2-score e.g. weighs recall two times higher than precision.

3. Related Work

As the thesis focuses on systematically comparing existing approaches, this chapter provides an overview of the current state of research and presents state-of-the-art methods. Starting with traditional machine learning approaches to have an understanding of what was used before the rise of LLMs. This will be followed by taking a look at the best fine-tuning and prompting techniques.

3.1. Traditional Machine Learning

As LLMs are still relatively new, it's important to have a look at approaches that were used before or even are still used. Cleland-Huang et al. [8] published the first study dealing with the automation of classifying different non-functional classes using machine learning in 2007. They trained a classifier using indicator terms, such as time or memory for performance requirements. The classifier was able to achieve up to 0.8 recall, but only 0.21 precision. They created their training dataset PROMISE NFR from 15 software projects, labeled by students. To this day, PROMISE NFR is one of the most frequently used datasets in research related to requirements classification (e.g. [19, 6, 4]).

Hussain, Kosseim, and Ormandjieva [21] introduced a method based on a decision tree classifier (C4.5) that achieves an F_1-score of up to 0.99 on a dataset that is related to PROMISE NFR. They made this possible by using certain categories of words that should indicate that a sentence contains non-functional requirements, such as adjectives, adverbs, cardinals, or numeric figures. Another approach based on the Support Vector Machine (SVM) classifier algorithm uses lexical, syntactical, and meta-data features—such as word n-grams, part-of-speech tags, and sentence structure—to automatically classify software requirements as either functional (FR) or non-functional (NFR). Kurtanović and Maalej [24] achieved F_1-scores up to 0.93 classifying requirements as functional or non-functional. Preprocessing the data has shown promise as well, as Abad et al. [2] investigated in their research. Their method was able to increase the F_1-score of the algorithm used by Hussain, Kosseim, and Ormandjieva [21] on the PROMISE NFR dataset from 0.91 to 0.95. Although those performances may sound excellent in the first place, they all rely on heavy pre-processing the given data before actually being able to classify them. To eliminate this need for manual pre-processing, approaches were being developed using LLMs.

3.2. Fine-tuning

In studies dealing with fine-tuning LLMs for requirements classification, one model clearly stood out: BERT. A large majority of state-of-the-art fine-tuning approaches use BERT-based models. Hey et al. [20] proposed an approach in 2020 that fine-tunes BERT, named NoRBERT, and has been used frequently as a baseline in papers when evaluating new approaches. They performed various classifications on PROMISE NFR, achieving an F_1-score of up to 0.915 on the FR vs. NFR task. When applying a multi-class classification to the most frequent NFR subclasses they reached up to 0.87 in F_1-score. They also applied their approach to the classification of functional and quality aspects on a relabeled version of PROMISE NFR [10] and were able to achieve an F_1-score of 0.9075. At the same conference (2020 IEEE 28th International Requirements Engineering Conference (RE)), Sainani et al. [33] presented their research that investigated different machine learning models to extract and classify requirements. The dataset used in the study contained software engineering contracts. The BERT model outperformed all other applied models (SVM, Random Forest, Naive Bayes and BiLSTM (Bidirectional Long-Short Term Memory)). They achieved an F_1-score of 0.8 or higher on 9 out of 14 classes.

A similar approach to NoRBERT was presented by Eltahier, Dawood, and Saeed [15], who were able to increase the F_1-score up to 0.98 on the FR vs. NFR task on PROMISE NFR, which is currently the best performance by an LLM on this specific task and dataset. Luo et al. [27] with their approach PRCBERT achieved best results on multiple tasks. On the PROMISE NFR dataset, they reached F_1-scores of up to 0.96 doing multi-class classification on all NFR subclasses. Their approach, Trans_PRCBERT, achieved F_1-scores of up to 0.9 in a transfer learning setting. The model was first fine-tuned on PROMISE and evaluated on non-functional requirements from other datasets, again classifying all NFR subclasses. Rejithkumar and Anish [31] focused on the classification of non-functional requirements. They introduced NICE, "a tool for NFR Identification, Classification, and Explanation." that uses GPT-4o to expand an existing dataset with explanations for the labels for each requirement. On this enriched dataset, four models were fine-tuned. When evaluated on a relabeled version of PROMISE NFR, they achieved an F_1-score of 0.95 on the multi-class classification of the NFR subclasses. The dataset, referred to as VenReq, contained requirements from a "major IT vendor organization". Due to a Non-Disclosure Agreement, the authors were not able to publish the dataset or the fine-tuned models. Instead of just classifying the requirements they also instructed the models to explain the classification. They applied DoRa (Decomposed Low-Rank Adaptation) [26], which is an enhanced version of LoRA. Tunstall et al. [38] introduced SetFit, a PEFT fine-tuning framework that only requires little amounts of data and outperforms standard task-specific head fine-tuning on 6 datasets. This framework was further explored by Shafikuzzaman et al. [34], who applied to models with SetFit on PROMISE NFR and SecReq, a security-related dataset. They achieved 0.95 when fine-tuning and evaluating on each of the three projects individually and calculating the macro average across the results.

3.3. Prompting

Since zero-shot prompting does not require labeled training data, it is very popular in research. Besides that fact, it also represents a frequently occurring scenario from the real world. When starting a new software project, the only option is usually just using a model, which is pre-trained on other data. Decoder-only LLMs like DeepSeek, Gemini, GPT-4, and Llama have proven to be very useful.

Binkhonain et al. [5] explored zero-shot, few-shot, chain-of-thought, and persona-prompting on the PROMISE NFR and the SecReq dataset. They achieved F_1-scores of up to 0.88 with zero-shot doing multi-classification on all non-functional subclasses of PROMISE NFR and demonstrated that three examples can be enough to achieve an improvement in the F_1-score of up to +0.08. This indicates that even a few examples can lead to a significant increase in performance. But this improvement is not guaranteed. The experiments also showed that, depending on the task, it is possible that even ten examples do not lead to any improvement in performance. Martin et al. [28] evaluated not exactly the same prompts, but the same fundamental prompting techniques with different models. They also used SecReq, but not the original PROMISE NFR; instead, they combined a modified version with another dataset and were able to achieve F_1-scores of 0.83 with zero-shot prompting doing binary classification on SecReq. Their results have shown as well that three examples can increase the F_1-score by 0.4. Zadenoori et al. [43] adapted an existing approach to automatic prompt engineering and evaluated it on a relabeled version of the PROMISE NFR dataset. Like the previous two papers, they were able to show that having examples can increase the performance by 0.5. Shafikuzzaman et al. [34] applied zero-shot and few-shot prompting to decoder-only LLMs like the others and used PROMISE NFR and SeqRec. They showed that sometimes few-shot does not improve the F_1-scoreat all, which matches the observation from Binkhonain et al. [5], indicating that the performance heavily relies on the underlying model. This phenomenon was also examined by Tang et al. [37], which shows that too many examples can decrease the performance. They showed as well that the order and quality of few-shot examples have an impact on the performance.

Table 3.1 provides an overview of state-of-the-art approaches. The first column contains the dataset and the second column the task that was evaluated on this dataset. The F_1-score column presents the average F_1-score reported by the authors. The underlying model for each approach is found in the corresponding Model column. The fifth column describes the technique that was used, and the last column contains the reference to the corresponding study. Approaches that do have a specific name are mentioned together with the reference.

Dataset	Task	F_1-score	Model	Technique	Reference
PROMISE NFR	FR-NFR	0.8400	Gemini 2.0 Flash	0-Shot-P-CoT	[5]
PROMISE NFR	B-NFR (top-4)	0.7595	GPT-4o	0-Shot	[34]
PROMISE NFR	MC-NFR	0.8800	DeepSeek-V3	0-Shot	[5]

POMISE FQ	F-Q	0.7400	Llama-3-8B-Instruct	0-Shot	[43]
SecReq	Sec-NonSec	0.8300	DeepSeek-R1-Distill-Qwen-14B	0-Shot	[28]
SecReq	Sec-NonSec (proj)	0.8270	GPT-4o	0-Shot	[34]
PROMISE+, DOSSPRE, SecReq	Sec-NonSec	0.6700	DeepSeek-R1-Distill-Qwen-14B	0-Shot-P	[28]
PROMISE NFR	FR-NFR	0.9200	Gemini 2.0 Flash	3-Shot-P	[5]
PROMISE NFR	B-NFR (top-4)	0.7410	GPT-4o	10-Shot	[34]
PROMISE NFR	MC-NFR	0.9400	DeepSeek-V3	3-Shot-P	[5]
POMISE FQ	F-Q	0.7900	Llama-3-8B-Instruct	APE	[43]
SecReq	Sec-NonSec	0.8700	GPT-4 Turbo	3-Shot-P-CoT / 5-Shot / 5-Shot-P	[5]
SecReq	Sec-NonSec (proj)	0.8230	GPT-4o	10-Shot	[34]
PROMISE+, DOSSPRE, SecReq	Sec-NonSec	0.5800	Mistral-Small	2-Shot	[28]
PROMISE NFR	FR-NFR	0.9150	$BERT_{base/large}$	Task-Specifc Head	NoRBERT [20]
PROMISE NFR	B-NFR (top-4)	0.8300	$BERT_{base}$	Task-Specifc Head	NoRBERT [20]
PROMISE NFR	MC-NFR (top-4)	0.8700	$BERT_{large}$	Task-Specifc Head	NoRBERT [20]
PROMISE FQ	F-Q	0.9075	$BERT_{base}$	Task-Specifc Head	NoRBERT [20]
PROMISE NFR	FR-NFR	0.9800	$BERT_{base}$	Task-Specifc Head	[15]
FR_NFR	FR-NFR	0.9700	$BERT_{base}$	Task-Specifc Head	[15]
PROMISE NFR	FR-NFR	0.9423	$RoBERTa_{large}$	Prompt-Tuning	PRCBERT [27]
PROMISE NFR	MC-NFR	0.9616	$RoBERTa_{large}$	Prompt-Tuning	PRCBERT [27]
NFR-Review	MC-NFR	0.8999	$RoBERTa_{large}$	Prompt-Tuning	PRCBERT [27]
NFR-Review	MC-NFR	0.9014	$RoBERTa_{large}$	Prompt-Tuning	PRCBERT [27]
NFR-SO	MC-NFR	0.8622	$RoBERTa_{large}$	Prompt-Tuning	PRCBERT [27]

NFR-SO	MC-NFR	0.8665	$\text{RoBERTa}_{\text{large}}$	Prompt-Tuning	PRCBERT [27]
PROMISE NFR	MC-NFR	0.8880	All-MPNET-base-v2	10-Shot-SetFit	[34]
SecReq	Sec-NonSec	0.8560	All-MPNET-base-v2	10-Shot-SetFit	[34]
SecReq	Sec-NonSec (proj)	0.9580	All-DistilRoBERTa-v1	10-Shot-SetFit	[34]
$\text{PROMISE}_{\text{relabeled}}$	FR-NFR	0.8700	flan-t5-large	DoRa	[31]
$\text{PROMISE}_{\text{relabeled}}$	MC-NFR	0.9500	flan-t5-large	DoRa	[31]

Table 3.1.: Overview for the related work. Task abbreviations: FR-NFR = functional vs. non-functional, B-NFR (top-4) = binary classification of the top-4 most frequent NFR subclasses, MC-NFR = multi-class classification of all NFR subclasses, MC-NFR (top-4) = multi-class classification of the top-4 most frequent NFR subclasses, F-Q = classification into functional and quality aspects, Sec-NonSec = security vs. non-security, Sec-NonSec (proj) = classification on individual projects instead of the whole dataset; Technique abbreviations: P = Persona prompting, CoT = Chain-of-Thought prompting, APE = Automatic Prompt Engineering

Existing research has made great progress in automating requirement classification, achieving strong results across a wide range of approaches, including traditional machine learning, fine-tuning, and prompting. Studies have explored diverse technique, such as BERT-based fine-tuning, zero- and few-shot prompting, and hybrid methods, demonstrating high performance on different tasks. While some works have evaluated different scenarios (e.g., training on one project and evaluating on an unseen project or training and evaluating on the same project), these scenarios were often framed as isolated experiments rather than part of a broader, systematic comparison. The primary limitation of prior work is the absence of a comprehensive, structured evaluation that directly compares fine-tuning and prompting across realistic use cases. This thesis addresses this gap by systematically comparing these strategies in controlled experiments, providing guidelines for selecting the most effective approach for different scenarios.

4. Research Design

This chapter introduces the research design of the thesis to address the limitations of the related work, starting with the scenarios and corresponding research questions followed by the experiment design.

4.1. Research Questions

To formulate the research questions, it is necessary to first introduce the application scenarios that are examined in this thesis. The basic idea for all the scenarios is the same: A team wants to classify unlabeled requirements for one or multiple of their internal projects. Internal in this case means that the projects are somehow related to each other, e.g., because they are part of the same company or managed by the same team. External projects are therefore referred to as the ones that are not related to the internal projects. The purpose of distinguishing between external and internal projects is primarily to motivate the scenarios, as for most of the experiments only one dataset is used, and only assumptions can be made about the relation between projects.

Scenario 1 - Cross project: The first scenario represents a common use case in reality. The idea is that the internal project is either a completely new software project or a project where no requirements have been classified. In both cases' no labeled data related to this project is available yet. If the team wants to use few-shot prompting or fine-tune a model, they thus have to rely on already labeled data from external projects.

RQ1: *How do fine-tuning and prompting compare when classifying requirements on unseen projects?*

Scenario 2 - Intra project: This scenario represents a situation where requirements from an already established project are labeled, meaning that labeled data from the current project is existing. The team can therefore use labeled data from their own project to further classify requirements belonging to the same project.

Scenario 3 - Intra-cross project: The last scenario present a combination of the previous two. The idea is that a pre-trained LLM is available that was already fine-tuned on external data. This may be publicly available or fine-tuned by the team itself. They now want to fine-tune the model even further with their own labeled data to classify their own requirements. The interest of this scenario lies in the comparison with the other scenarios to investigate the importance of labeling your own data.

RQ2: *What impact has the adding of internal requirements on the performance of both fine-tuning and prompting?*

RQ3: *How many requirements from the same project have to be labeled so that it is worth fine-tuning an LLM instead of performing few-shot prompting?*

Baseline - Mixed project: In this case, the team has labeled data from external and internal projects, and the goal is to classify external as well as internal requirements. This scenario is the most commonly evaluated in the related work, but it does not represent any real-world situation. It is therefore only used as a reference for the above scenarios.

RQ4: *How does the performance on different scenarios compare to the baseline?*

4.2. Experiment Design

This section explains how the experiments that should help to answer the research questions, are designed in detail. This includes the tasks and datasets the experiments are executed on, but also which and why certain approaches were chosen to investigate. The experiment setups for the previous mentioned scenarios are being presented at the end.

4.2.1. Tasks

Based on the related work, five tasks were identified. Of those tasks, the following three were prioritized due to time constraints: Functional vs. Non-Functional, Security vs. Non-Security, Functional and Quality. The first two tasks are traditional binary classifications and the third is a multi-label classification as defined in **??**. As the functional and quality task includes four classifications (IsFunctional, IsQuality, OnlyFunctional, OnlyQuality) there are two possibilities to classify a requirement in practice: 1. A multi-class classification where the model is instructed to return a label for every classification. 2. Two independent binary classifications where the model has to classify a requirement as functional or non-functional and quality or non-quality, and based on these classifications, the labels for the remaining two are automatically derived. Because existing approaches from the reviewed literature implemented the second option, it is also chosen for this thesis. For all tasks, precision, recall, F_1-score and F_2-score are measured.

4.2.2. Datasets

In order to perform the tasks just mentioned, the appropriate labeled data is required. Table 4.1 provides details for all relevant datasets. The size represents the total number of requirements in a dataset. The labels represent the categories a requirement can be classified as. The first number in the distribution column refers to the first mentioned label in the labels' column. The total number of projects is shown in the projects' column, and a project does either originate from students or the industry. One of the most commonly

Dataset	Size	Labels	Distribution	Projects	Origin
PROMISE NFR	625	FR, NFR	255/370	15	students
PROMISE+	3677	FR, NFR	1831/1846	91	industry, students
SecReq	510	Sec, NonSec	187/323	3	industry
DOSSPRE	1317	Sec, NonSec	801/516	105	students
PROMISE FQ	625	F, Q	310/382	15	students

Table 4.1.: Datasets overview, Labels abbreviations: FR, NFR = functional, non-functional; Sec, NonSec = security, non-security; F, Q = functional, quality

used datasets for the functional vs. non-functional classification is PROMISE NFR [9]. Here, requirements are labeled as either functional or non-functional. As it only contains 625 requirements from student projects exclusively, it was expanded multiple times. PROMISE+ [16] presents the currently largest expansion and includes additional student and industry projects. Both datasets contain more labels than the two presented in the table because the non-functional class is further divided into 11 subclasses, such as performance, security, or usability. This offers the possibility to also perform the security vs. non-security task on these datasets. The two datasets that deal exclusively with security are SecReq [23] and DOSSPRE [29], with SecReq containing only industry and DOSSPRE containing only student projects. To perform the last task, PROMISE FQ by Dalpiaz et al. [10] can be used. It is a relabeled version of PROMISE NFR that contains the same requirements but different labels. For all datasets except PROMISE FQ, the two values in the distribution column add up exactly to the total size because a requirement is labeled as either label 1 or label 2. However, a requirement in PROMISE FQ can also be labeled as functional and quality or neither of both. The first experiments are executed on PROMISE NFR/FQ and SecReq as these are the smallest datasets in this thesis and therefore the fastest to evaluate. This approach ensures that there are definitely results to present at the end.

4.2.3. Approach

Having established the scenarios, tasks, and datasets, this section deals with the approaches from the related work that are being implemented to actually execute the experiments. While traditional machine learning methods may be briefly mentioned for contextual comparison, they are not evaluated in detail or adapted within this work because they are not as suitable for generalization settings as LLM-based methods [44]. The selection of approaches used for comparison was based on performance and feasibility of replication. Every experiment is conducted four times with different random seeds to ensure the reliability and robustness of the results. The seeds are further fixed to make sure that the results are reproducible. Cross-validation is used for all experiments, but for the ease of understanding, the splitting of data is simplified into a train and test set. Details about the folds are provided in Subsection 4.2.4.

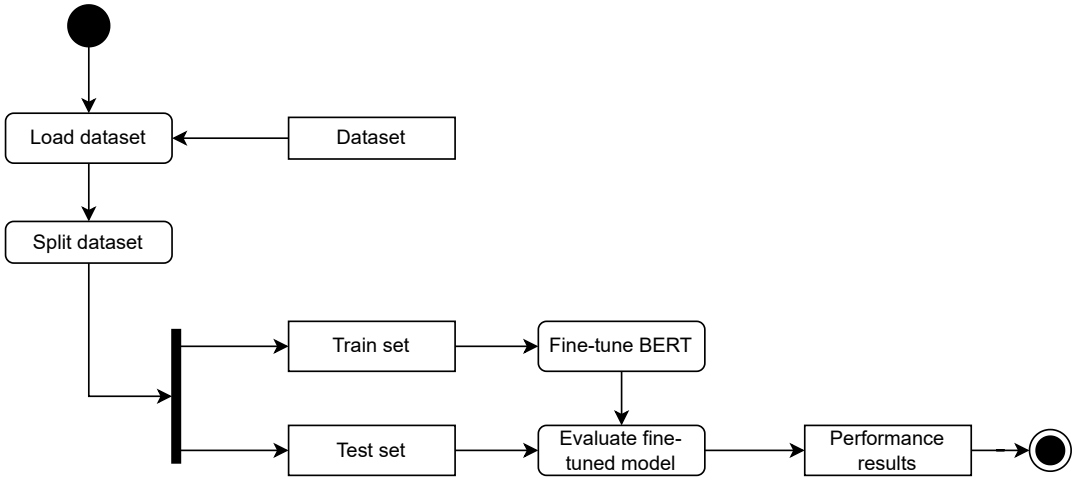

Figure 4.1.: Fine-tuning process

4.2.3.1. Fine-tuning

For fine-tuning, three approaches were chosen. The first two are NoRBERT with $\text{BERT}_{\text{base}}$ and $\text{BERT}_{\text{large}}$, as they achieved not the best but good results (Section 3.2). The third approach is Eltahier, where the general process is the same, but the hyperparameters are different, the data is preprocessed before splitting the dataset, and an extra validation set is used during training to select the model that performed best in validation. Although it is pretty similar to NoRBERT, it was chosen because Eltahier, Dawood, and Saeed [15] promised the best results on the PROMISE NFR dataset. Figure 4.1 illustrates the basic fine-tuning process: First, the dataset is loaded and split into a train and a test set. Then the pre-trained BERT model is trained on the train data. The resulting fine-tuned model is evaluated on the test data to obtain the performance results.

	$\text{NoRBERT}_{\text{base}}$	$\text{NoRBERT}_{\text{large}}$	Eltahier
Pre-trained Model	bert-base-uncased	bert-large-uncased	bert-base-uncased
Max sequence length	128	50	256
Num train epochs	10	10	8
Batch size	16	16	8
Learning rate	2e-5	2e-5	1e-5
Weight decay	0.01	0.01	0.01

Table 4.2.: Fine-tuning approaches details

Table 4.2 gives an overview of the different hyperparameters and the pre-trained models for the different approaches. The hyperparameter configurations for all approaches were chosen based on the performance on the binary classification of functional vs. non-functional on PROMISE NFR. Eltahier uses a maximum sequence length that is two times larger than the one for $\text{NoRBERT}_{\text{base}}$ and even five times larger than the one for $\text{NoRBERT}_{\text{large}}$. This means

that Eltahier can process the most input tokens simultaneously, but therefore allocates more memory. The number of training epochs is ten for both versions of NoRBERT and eight for Eltahier allowing the NoRBERT models to refine their parameters more effectively as they see the data two additional times. The batch size of Eltahier is only half the size of the NoRBERT models. The same ratio applies to the learning rate. Combining both values, Eltahier updates the model's weights more frequently, but the changes for each individual update are smaller. The weight decay is 0.01 for all approaches.

4.2.3.2. Prompting

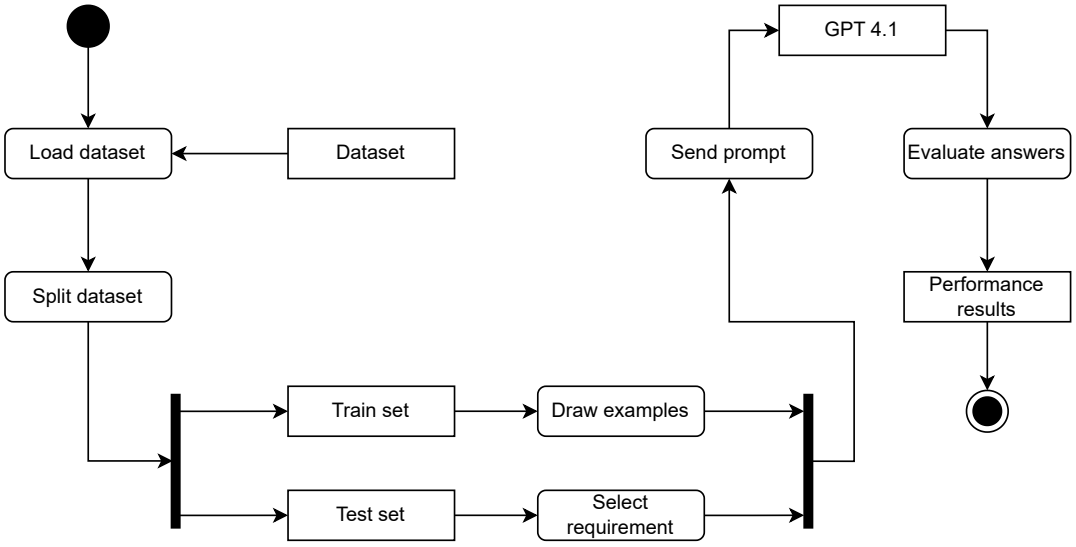

Figure 4.2.: Prompting process

The models for the prompting experiments are restricted to GPT-4.1. This decision was made for several reasons: 1. The access to OpenAI's API was already available, and the related work showed that GPT models generally achieve good results and even the best performance on some tasks. 2. As reasoning models would be unnecessarily complex for the task of classifying requirements, the idea was to use the latest non-reasoning model, which is GPT-4.1 and, according to OpenAI, the "Smartest non-reasoning model." [1]

Figure 4.2 illustrates the basic prompting process. Like for fine-tuning the dataset is split into a train and a test set. To build the prompt the examples are taken from the train set and the requirement to classify from the test set. This prompt is then given as input to GPT-4.1. The performance results are then calculated based on the answers.

The second important thing for prompting is the prompts themselves. The decision of which prompts to use was guided by both performance and replicability. Prompts were only chosen if they distinguished between system and user prompts, as this not only provides

[1] https://platform.openai.com/docs/models/gpt-4.1 (Accessed: 30.01.2026)

a better structure and consistent implementation but is also recommended by OpenAI [2]. Therefore, the decision was made to use the prompts provided by Zadenoori et al. [43] for the classification of requirements into functional and non-functional as well as functional and quality. They were the only ones evaluating on the functional and quality task in the reviewed literature, and they systematically created their prompts. Prompt 4.1 shows the system prompt at the top, which is the same for the zero-shot and few-shot prompts. It contains instructions, including the model's role and the required output format. The variables {A} and {B} contain the two categories a requirement can be classified as. The zero-shot user prompt consists just of the text "Requirement:" followed by a requirement, which is stored in the {requirement_text} variable. The few-shot prompt, which is presented last, adds examples and their corresponding label at the beginning before providing the requirement to be classified. For each example, the {example_text} variable contains the example requirement followed by an arrow and the corresponding label, stored in {label}.

Prompt 4.1: Zadenoori Basic

System Prompt

As an expert system for classifying software requirements, your task is to carefully analyze each given requirement and categorize it into one of the following two categories:
"0": {A}
"1": {B}
Output only the number (0 or 1) that corresponds to the appropriate category. Do not provide any additional explanations.
Please provide the categorized number for the given software requirement, with no additional text.

Zero-Shot User Prompt

Requirement: {requirement_text}

Few-Shot User-Prompt

Below are examples of different types of requirements and their classifications:
{example_text} → {label}
{example_text} → {label}
⋮
Requirement: {requirement_text}

Prompt 4.2 shows the chain-of-thought user prompt, which is the optimized version from the paper by Zadenoori et al. [43]. It is the result of their automated prompt engineering approach. The system prompt is the same as for the basic prompts. In contrast to all the other prompts, the model is provided with detailed instructions, which include five steps. The prompt not only contains examples but also explanations for each class as well as classification guidelines (Steps 1 and 2), and the model is instructed to understand how to classify requirements using the first two steps (Step 3). Applying this understanding to a given requirement [Step 4] is followed by the instruction to classify this requirement into one of the provided categories {A} and {B}. Here, only the the few-shot version is presented, as the zero-shot prompt only differs in leaving the presentation of examples (Step 1) out.

[2] https://help.openai.com/en/articles/9824968-prompt-management-in-playground (Accessed: 09.03.2026)

Prompt 4.2: Zadenoori Chain-of-Thought

Few-Shot User-Prompt

You are an expert system that needs to classify software requirements into two categories: **{A}** and **{B}**.

Let's analyze the classification of requirements step by step.

Step 1: Review the examples of different types of requirements and their classifications:
"'{example_text}'" → **{label}**
"'{example_text}'" → **{label}**
⋮
Class Explanations:
* **{A}**: These requirements specify …
* **{B}**: These requirements specify …
Classification Guideline:
1. If the requirement describes…, classify it as **{A}**.
2. If the requirement describes …, classify it as **{B}**.

Step 2: Read the explanations for these classifications:
- {A}: {A} requirements specify …
- {B}: {B} requirements specify …
Step 3: Understand how to classify requirements using the examples and explanations as guidance.

Step 4: Apply this understanding to the following requirement.

Requirement: "'{requirement_text}'"
Step 5: Determine the classification of the requirement and provide the final label of the class without any explanations.

The output categories should be exact the same as the categories mentioned here: **{A}** and **{B}**.

For the classification into security and non-security, the prompts by Shafikuzzaman et al. [34] were chosen as both Binkhonain and Alfayez [5] and Martin et al. [28] did not use any system messages. In contrast to the prompts by Zadenoori et al. [43], the required output format is only included in the user prompts. The variables represent the same values as before. Prompt 4.3 displays the system prompt, which only contains the role of the model. Again, this is used for both zero-shot and few-shot. Right below that the zero-shot user prompt is shown. The model is provided with the class names and is asked to classify a given requirement. The few-shot prompt, which is presented last, contains the zero-shot prompt followed by example requirements. Here also the number of examples as well as their format is given.

Prompt 4.3: Shafikuzzaman

System Prompt

You are an AI assistant for requirement classification task

Zero-Shot User Prompt

Given the following requirement text, can you classify the requirement? Select the class from '{A}, {B}'. No further explanation is needed. Return the name of the class only. Text: '{requirement_text}'

Few-Shot User Prompt

Given the following requirement text, can you classify the requirement? Select the class from '{A}, {B}'. No further explanation is needed. Return the name of the class only. Text: '{requirement_text}'. For better understanding, here I am giving 'n' examples where requirement text is given first, then class name seperated by comma: "{example_text}", {label} Requirement
"{example_text}", {label} Requirement
⋮
No need to return output for these examples.

Regarding the number of examples, the decision was made to examine one-, two-, four- and eight-shot for all tasks and scenarios. Zero-shot is only examined on the baseline as it uses no examples.

4.2.4. Scenarios

This section deals with the experiment setup for the different application scenarios. Here, every scenario consists of multiple experiments, one for each task.

4.2.4.1. Scenario 1 - Cross project: No project-specific data available

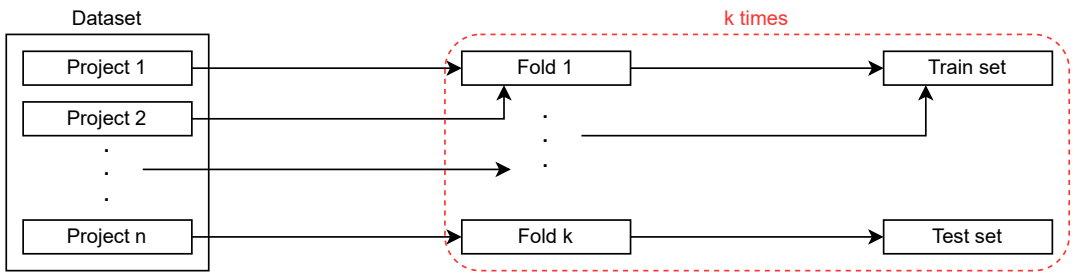

Figure 4.3.: Scenario 1 - Cross project split

In the first scenario, a team wants to classify requirements from their project just based on labeled data from external projects. For this scenario a p-fold validation is used (Subsection 2.3.2). As shown in Figure 4.3, the dataset is split into k folds. Then one fold is used for testing and the remaining k-1 folds for training. This process is repeated k times so

that every fold is used once for testing. In this case, there is no overlapping data from one project between training and test. The exact assignment of projects to folds depends on the dataset, as the datasets contain different amounts of projects.

For PROMISE NFR/FQ, two different splits were used: 1. Dalpiaz p-fold, which was introduced by Dalpiaz et al. [10] and also used by Hey et al. [20]. Since this dataset contains 15 projects, 12 projects are used for training and the remaining 3 for testing. They produced 10 partitions, where every partition consists of 3 projects and has at least 100 requirements. Additionally, they ensured that the same combination of two projects does not occur twice in two different partitions; e.g., if projects 1 and 2 are part of a partition, these projects are not allowed to occur together in any other partition. The reason for that is that the projects in PROMISE NFR differ very much in size and label distribution. The folds were therefore designed to be as balanced as possible. As every project appears in two folds, duplicate projects between train and test set are removed from the train set. 2. A custom p-fold was produced, which follows the same principles as the Dalpiaz p-fold, but the number of requirements across all partitions has a slightly smaller standard deviation. This was done to produce more reliable results and not being too dependent on only one way of partitioning the projects.

As SecReq only consists of three projects, the creation of folds for the splitting at project borders is pretty straightforward. One project is used for testing and the other two for training. This way every project is used exactly once as the test set.

4.2.4.2. Scenario 2 - Intra project: Only project-specific data available

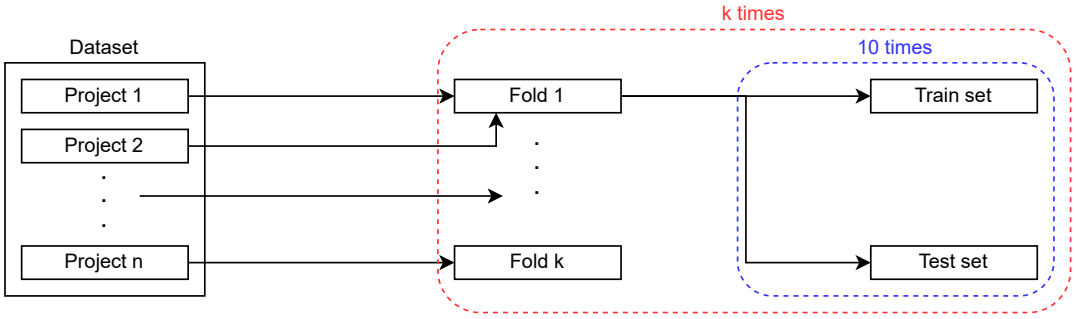

Figure 4.4.: Scenario 2 - Intra project split

The intra project scenario represents a situation where a team wants to label requirements from the same projects they already have labeled requirements on, and they only use requirements from these projects for training. Figure 4.4 shows how the data is split for this scenario. The same p-fold split is used as for the cross project scenario. Instead of a p-fold validation, only one fold is split into a train and a test set using a 10-fold cross-validation, meaning the requirements for training and testing originate from the same fold. Like in the previous scenario, this process is repeated k times.

4.2.4.3. Scenario 3 - Intra cross project: Project-specific data available

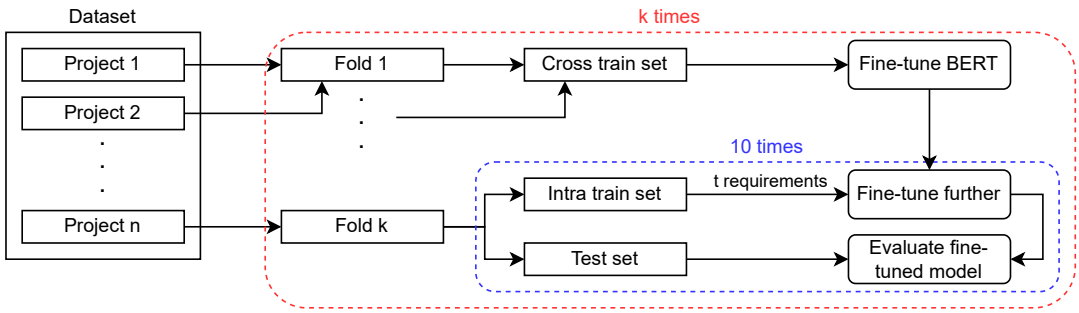

Figure 4.5.: Scenario 3 - Intra cross project split

In this scenario, labeled data from external projects is available as well as some labeled data from an internal project. The focus lies on the amount of internal requirements a team has to label. As shown in Figure 4.5, the data is first split like for the cross project scenario using a p-fold. The model is then fine-tuned using the cross train set, which is the same train set as for the cross project scenario. A 10-fold cross validation is applied to the remaining fold, resulting in an intra train and test set. The already fine-tuned model is further fine-tuned on the intra train set and then evaluated on the test set. The number t of requirements that are sampled from the intra train set to fine-tune the model further is varied to examine how many requirements have to be labeled from the internal project. This is repeated 10 times for every k folds of the p-fold split.

4.2.4.4. Baseline - Mixed project

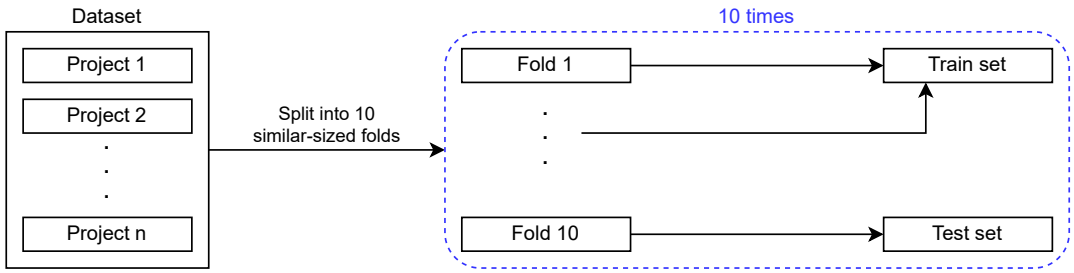

Figure 4.6.: Baseline - Mixed project split

For the baseline, data from both external and internal projects is available for training. Figure 4.6 visualizes the data splitting. For all datasets a 10-fold cross-validation is used (Subsection 2.3.2). This means that the data is split into ten independent folds. Then 9 folds are used for training and 1 fold is used for testing. This is repeated 10 times. As the data is split randomly and without caring about project borders, there could be an overlapping of projects between training and testing.

5. Results

In this chapter, the results of all conducted experiments are presented. The experiments are ordered by scenario, and within a scenario further distinction is made between the different tasks. For fine-tuning, $\text{NoRBERT}_{\text{base}}$, $\text{NoRBERT}_{\text{large}}$ and Eltahier were used with and without oversampling and for prompting, GPT-4.1 was used. All results are presented in this chapter are based on the F_1-score. Detailed results for all experiment including also precision, recall and F_2-score are provided in the Appendix.

5.1. Baseline - Mixed project

The first experiments were performed on the baseline. For the prompting experiments zero, one, two, four, and eight examples were used. To systematically compare the fine-tuning performance across different train sizes and tasks, always the whole dataset is split into training and testing using a 10-fold and then the size of the training set is reduced by subsampling a given number of requirements. Each experiment is first performed with no restrictions for the number of training requirements and after that the size is reduced by using powers of 2 with 64 being the lowest.

5.1.1. Functional vs. Non-Functional Requirements

The first task is a binary classification of requirements into either functional or non-functional (FR vs. NFR). All experiments regarding this task were performed on PROMISE NFR.

Figure 5.1 shows the prompting results where a distinction is made between basic and chain-of-thought prompts, with the number of examples on the X-axis and the average F_1-score on the Y-axis. A point represents the mean and standard derivation of the F_1-scores across four runs for the given number of examples. For both plots the performance gets better by adding more examples until a threshold of two for the basic and four for the chain-of-thought prompts. Adding more than two examples to the basic prompt does not increase the performance noticeably for the basic prompts. The difference there in the average F_1-score between four and two and eight and two examples is less than 0.001. Although the model receives more detailed instructions, the chain-of-thought prompts lead to a worse performance than the basic ones across all tested numbers of examples. As the performance of the chain-of-thought prompts is not only worse but the API usage is also

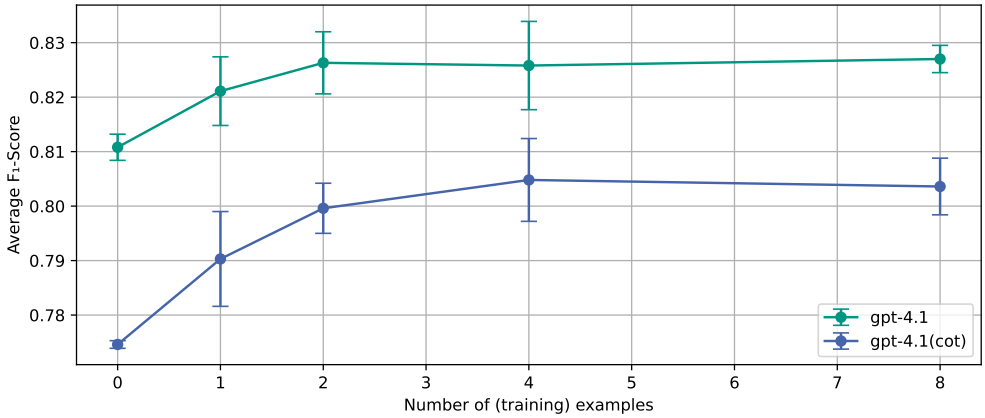

Figure 5.1.: Mixed project: Prompting on PROMISE NFR with different amounts of examples

a lot more expensive, the decision was made to only use basic zero- and few-shot for all following experiments.

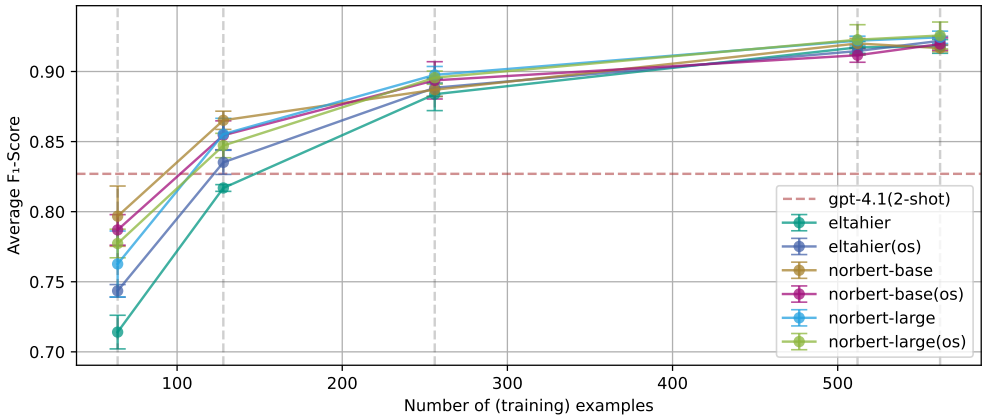

Figure 5.2.: Mixed project: Fine-tuning on PROMISE NFR with different amounts of examples (64, 128, 256, 512, 562)

Figure 5.2 displays the fine-tuning results for the six model configurations. The red dashed horizontal line highlights the best performing prompting approach on the FR vs. NFR classification task in this scenario. When given at least 256 requirements, all approaches achieve an F_1-score above 0.85. The biggest decline in performance is shown when given less than 128 requirements for training. When given 64 requirements the F_1-score drops below 0.8 for all approaches. Except for Eltahier without oversampling, which requires more than 128 requirements, all plots intersect with the prompting baseline between 128 and 64 requirements. As shown in the graph, oversampling can improve the performance, especially when having fewer requirements to train on, but it does not have to.

5.1.2. Security vs. Non-Security Requirements

The second binary classification distinguishes between security and non-security requirements (Sec vs. NonSec) and was performed on SecReq.

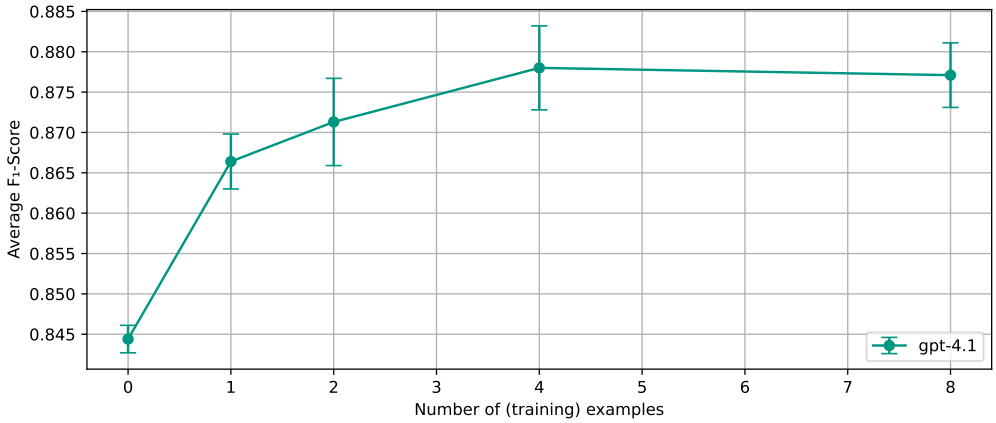

Figure 5.3.: Mixed project: Prompting on SecReq with different amounts of examples

Figure 5.3 shows the results of prompting with different numbers of examples. GPT-4.1 achieves the worst results with zero-shot. The F_1-score here is 0.8444. The best performance is achieved with four examples added to the prompt, resulting in an F_1-score of 0.8780. Similar to the first task, the performance increase is biggest when the model is given one example instead of none.

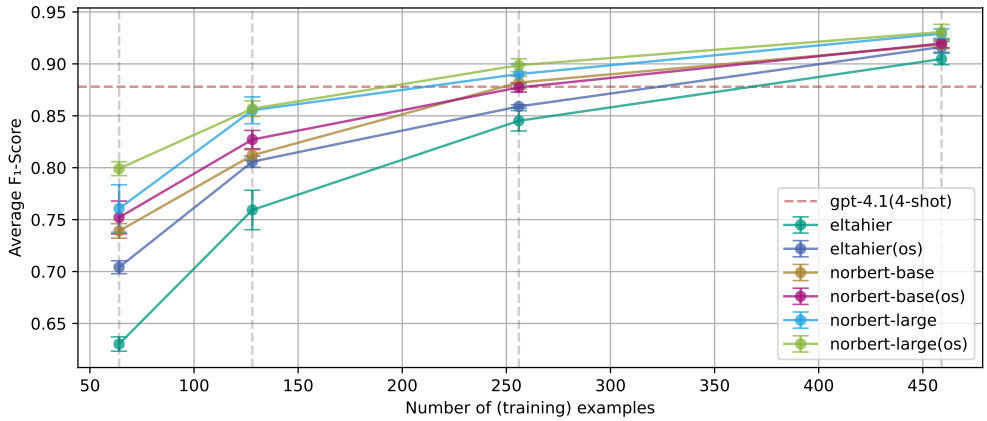

Figure 5.4.: Mixed project: Fine-tuning on SecReq with different amounts of examples (64, 128, 256, 459)

The fine-tuning results are presented in figure Figure 5.4. $\text{NoRBERT}_{\text{large}}$ performed best and Eltahier worst across all numbers of training examples. The performance of $\text{NoRBERT}_{\text{base}}$ is always in between those two. For this task the approaches with oversampling consistently performed better than the same approach without oversampling. The difference is particularly noticeable with 128 requirements or fewer. The prompting baseline intersects

with NoRBERT$_{base}$ at 256 training examples. Eltahier even needs over 300 examples to be as good as the four-shot prompt. NoRBERT$_{large}$ intersects with the prompting baseline at around 200 examples.

5.1.3. Functional and Quality Requirements

The last task is the only multi-label classification. The models were first instructed to classify the requirements as either functional or non-functional and then as either quality or non-quality. Whether a requirement was labeled as only functional or only qualitative was automatically derived based on the two binary classifications. Therefore, all diagrams regarding this task contain two graphs, one for each actual executed classification.

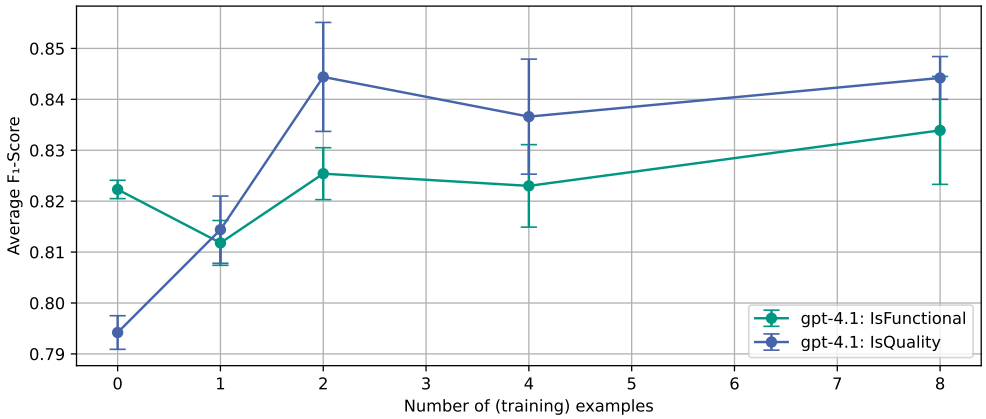

Figure 5.5.: Mixed project: Prompting on PROMISE FQ with different amounts of examples

Figure 5.5 shows the prompting results for the mixed project scenario on this task. Except for the zero-shot prompt, GPT-4.1 performs better on the classification of quality vs. non-quality than functional vs. non-functional. If read from left to right, so from none to many examples, the F_1-score drops from zero to one-shot. It then increases from one to two-shot, decreases again between two and four-shot and increases once again between four and eight examples. The graph for quality vs. non-quality shows a similar curve except that the performance increases when given one instead of zero examples. To be able to compare prompting with fine-tuning, one number of examples was chosen for both classifications. Across all classifications, eight examples show the best results. The prompt is only slightly worse for the quality vs. non-quality classification compared to the two-shot prompt, but the difference in the F_1-score is only 0.0002.

For the sake of space and readability, the fine-tuning results are only visualized for the best-performing approach, which in this case is NoRBERT$_{large}$ with oversampling. Figure 5.6 presents two prompting baselines, one for each classification. The color of the baseline corresponds to the color of the fine-tuning graph. This means green equals IsFunctional and blue is used for IsQuality. The green baseline intersects with the fine-tuning graph just below 256 training examples. The blue line intersects with the fine-tuning graph between 64 and 128 training examples.

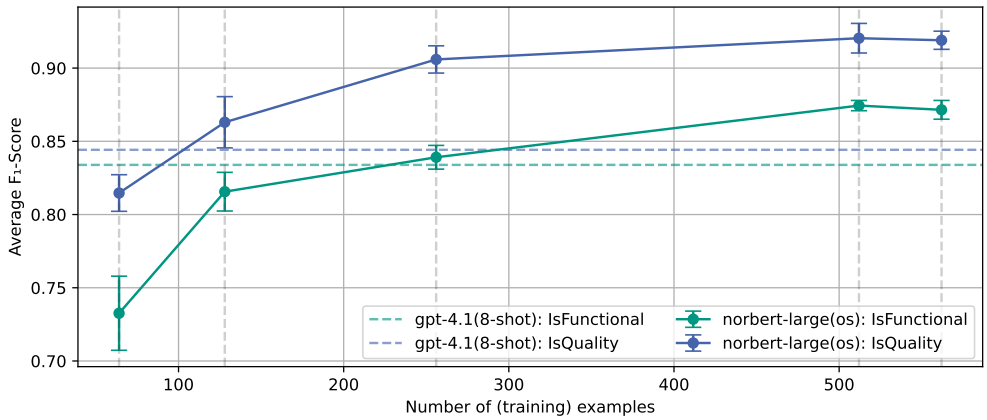

Figure 5.6.: Mixed project: Fine-tuning on PROMISE FQ with different amounts of examples (64, 128, 256, 512 and 562)

5.2. Scenario 1 - Cross project

After presenting the baseline results, this section contains the results for the first scenario. Here, care was taken to ensure that there was no overlap between training and test projects using a p-fold cross validation for all datasets. A project is therefore either part of the training or test set. The fine-tuning results are presented as tables instead of diagrams as there were always used all external requirements for training. The zero-shot prompts were not evaluated again for this scenario as it should not make any difference how the data is split as no examples are taken from the train set.

5.2.1. Functional vs. Non-Functional Requirements

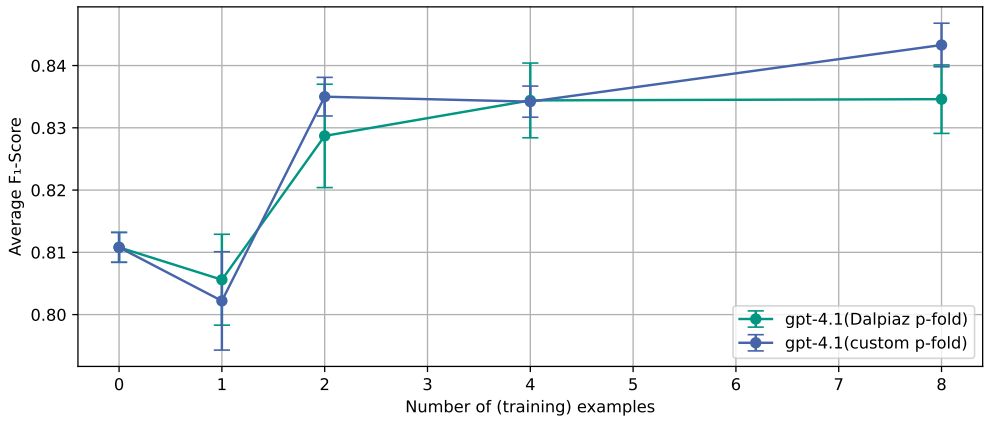

Figure 5.7.: Cross project: Prompting on PROMISE NFR with project folds and different amounts of examples

Figure 5.7 shows the prompting results for the first task where the data was split using the Dalpiaz project fold and the custom project fold. For both p-folds zero-shot is slightly better

than one-shot. For the Dalpiaz p-fold, the performance gets better the more examples are added to the prompts until a threshold of four. Adding eight examples to the prompt does increase the F_1-score by 0.0002 compared to four examples. The biggest incline is between one and two examples. Using the custom project fold does make a difference. The best performance on this project fold is achieved with eight examples, resulting in an F_1-score of 0.8433. On this project-fold, two examples even lead to a slightly better performance than four. Like for the Dalpiaz p-fold the performance increase is biggest between one and two examples.

Approach	F1 Avg	
	Dalpiaz p-fold	custom p-fold
norbert-base	0.7596 ± 0.0061	0.8096 ± 0.0028
norbert-base(os)	0.7679 ± 0.0025	0.7962 ± 0.0019
norbert-large	0.7574 ± 0.0034	0.7943 ± 0.0039
norbert-large(os)	0.7584 ± 0.0156	0.7541 ± 0.0031
eltahier	0.7563 ± 0.0041	0.7818 ± 0.0014
eltahier(os)	0.7359 ± 0.0054	0.7655 ± 0.0020
gpt-4.1(0-shot) / gpt-4.1(2-shot)	0.8108 ± 0.0024	0.8350 ± 0.0031

Table 5.1.: Cross project: Fine-tuning on PROMISE NFR with project folds

Table 5.1 provides an overview of the fine-tuning results using both project folds. The second column presents the results on the Dalpiaz p-fold. The first thing that immediately stands out is that all approaches performed significantly worse compared to prompting, even to zero-shot. $\text{NoRBERT}_{\text{base}}$ with oversampling achieved the highest F_1-score of 0.7679. In general, $\text{NoRBERT}_{\text{base}}$ performed best, $\text{NoRBERT}_{\text{large}}$ slightly worse and Eltahier the worst regardless of using oversampling or not. The fine-tuning results for the custom p-fold are presented in the third column. All approaches except $\text{NoRBERT}_{\text{large}}$ with oversampling performed better on this p-fold than on the other one. $\text{NoRBERT}_{\text{base}}$ achieves an F_1-score of 0.8096, being the highest. Compared with two-shot prompting, the fine-tuning results are all worse. The standard deviation was reduced for all approaches on this p-fold compared to the Dalpiaz one.

5.2.2. Security vs. Non-Security Requirements

The prompting results for the Sec vs. NonSec task are presented in Figure 5.8. Reading from left to right the plot goes up from zero to one and from one to two examples, down to four examples and up again to eight examples. On this task eight-shot performed best, having an F_1-score of 0.8786. The lowest F_1-score of 0.8444 was measured with zero-shot.

The results for fine-tuning are provided in Table 5.2. While $\text{NoRBERT}_{\text{large}}$ achieves a slightly worse performance on this task than on the functional vs. non-functional task, $\text{NoRBERT}_{\text{base}}$ and Eltahier performed significantly worse. They reached 0.6394 and 0.5694 in F_1-score.

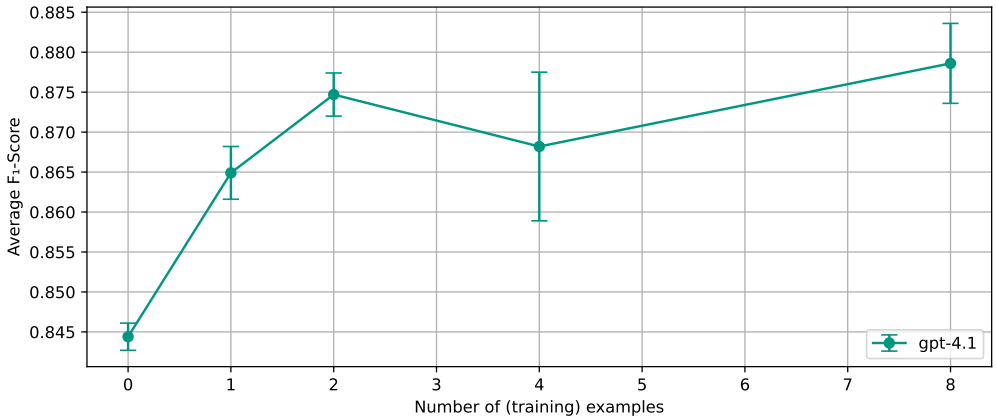

Figure 5.8.: Cross project: Prompting on SecReq with project fold and different amounts of examples

Approach	F1 Avg
norbert-base	0.6394 ± 0.0447
norbert-base(os)	0.6342 ± 0.0289
norbert-large	0.7545 ± 0.0379
norbert-large(os)	0.7166 ± 0.0348
eltahier	0.5694 ± 0.0165
eltahier(os)	0.5438 ± 0.0323
gpt-4.1(0-shot)	0.8444 ± 0.0017

Table 5.2.: Cross project: Fine-tuning on SecReq with project fold and different amounts of examples

What all fine-tuning approaches have in common, is a high standard deviation, ranging from 0.0165 up to 0.0447.

5.2.3. Functional and Quality Requirements

For the last task in this scenario, there are the same p-folds used as for the first task as the experiments were performed on PROMISE FQ which contains the same projects as PROMISE NFR. For the prompting experiments, four examples were the highest in order to avoid exceeding the API budget.

The results for prompting are presented in Figure 5.9. The curve of the graphs look very similar for both p-folds. Four-shot performs best for the IsFunctional classification and two-shot for the IsQuality classification. On the first classification the performance of one-shot is worse than zero-shot, on the second classification, one-shot shows slightly better results.

Table 5.3 contains the fine-tuning results for the classification into functional and quality aspects. Like for prompting, the performance is very similar between the different approaches and between the p-folds. But in contrast to prompting, no approach is clearly the best or

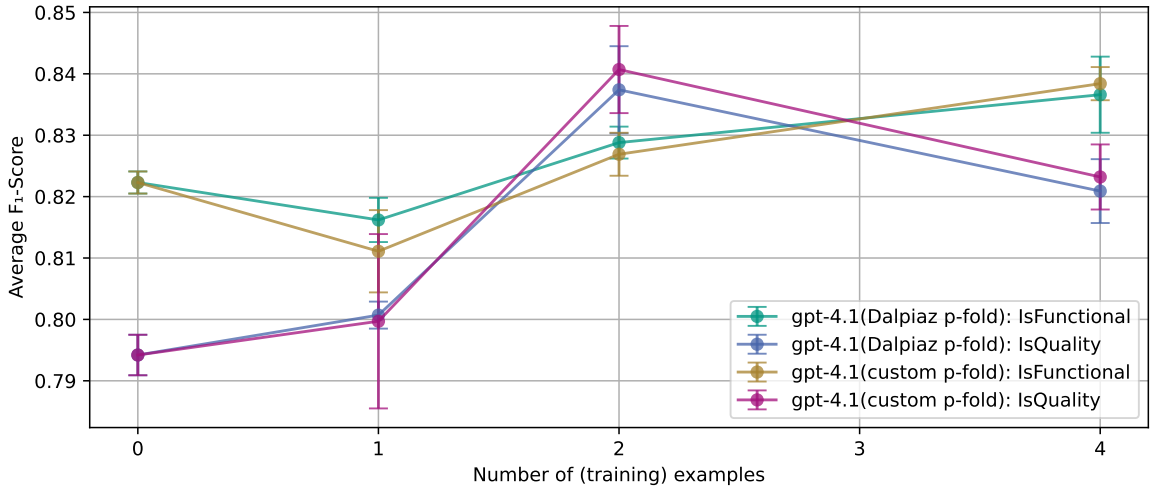

Figure 5.9.: Cross project: Prompting on PROMISE FQ with project folds and different amounts of examples

Approach	F1 Avg			
	Dalpiaz p-fold		custom p-fold	
	IsFunctional	IsQuality	IsFunctional	IsQuality
norbert-base	0.8123 ± 0.0019	0.8089 ± 0.0066	0.8143 ± 0.0017	0.8272 ± 0.0005
norbert-base(os)	0.8092 ± 0.0014	0.8286 ± 0.0109	0.8230 ± 0.0074	0.8365 ± 0.0038
norbert-large	0.8162 ± 0.0022	0.8463 ± 0.0000	0.8194 ± 0.0036	0.8543 ± 0.0031
norbert-large(os)	0.8193 ± 0.0029	0.8422 ± 0.0142	0.8183 ± 0.0033	0.8460 ± 0.0079
eltahier	0.8158 ± 0.0007	0.8096 ± 0.0040	0.8144 ± 0.0045	0.8334 ± 0.0021
eltahier(os)	0.8123 ± 0.0050	0.8005 ± 0.0084	0.8184 ± 0.0033	0.8262 ± 0.0069
gpt-4.1(2-shot)	0.8288 ± 0.0026	0.8374 ± 0.0071	0.8269 ± 0.0035	0.8407 ± 0.0071

Table 5.3.: Cross project: Fine-tuning on PROMISE FQ with project-folds

worse and also comparing the two p-folds, there is no classification that shows the best or worse results across all approaches. The best single performance was achieved with $\text{NoRBERT}_{\text{large}}$ on the custom p-fold, distinguishing requirements into quality or non-quality, reaching an F_1-score of 0.8543. The best prompting approach on this task was achieved with two examples and an F_1-score of 0.8407.

5.3. Scenario 2 - Intra project

In the second scenario all the data originates from the same project or group of projects. Due to lack of time, the scenario was evaluated using prompting only. However, the intra cross scenario was evaluated using only fine-tuning, so that these two scenarios can be compared. For prompting, the number of examples refers to the number of examples per

project. This means that for the PROMISE-related datastes in the first and third task, 3, 6, and 12 examples were used in total and 1, 2, and 4 were used for SecReq. Due to cost constraints, only one run was performed per task instead of four.

5.3.1. Functional vs. Non-Functional Requirements

Figure 5.10 shows the prompting results for the first task on the two project folds. Comparing the performance on both p-folds, it can be observed that the performance on the Dalpiaz p-fold is better across all tested numbers of examples. For both p-folds the performance increases when more examples are added. GPT-4.1 achieves an F_1-score of 0.8610 with four examples per project on the Dalpiaz p-fold and 0.8571 on the custom one.

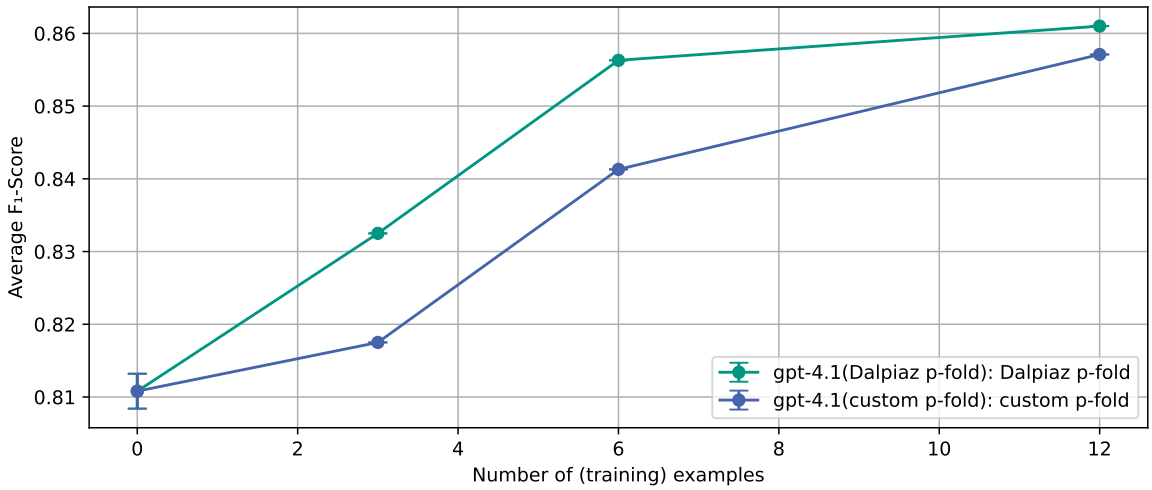

Figure 5.10.: Intra project: Prompting on PROMISE NFR with project folds and different amounts of examples

5.3.2. Security vs. Non-Security Requirements

The results for the Sec vs. NonSec task are visible in Figure 5.11. Like in the previous scenarios, GPT-4.1 performs better on this task than on the FR vs. NFR task. Zero-shot shows a slightly better performance then one-shot, but with at least one example that the performance increases by adding more examples to the prompt, resulting in a similar looking graph than the two for the first task. The highest F_1-score is 0.8688 (four-shot) and the lowest 0.8421 (one-shot).

5.3.3. Functional and Quality Requirements

Figure 5.12 visualizes the results for the last task using both p-folds. Both classifications on the Dalpiaz p-fold show an increasing performance when prompting with more examples.

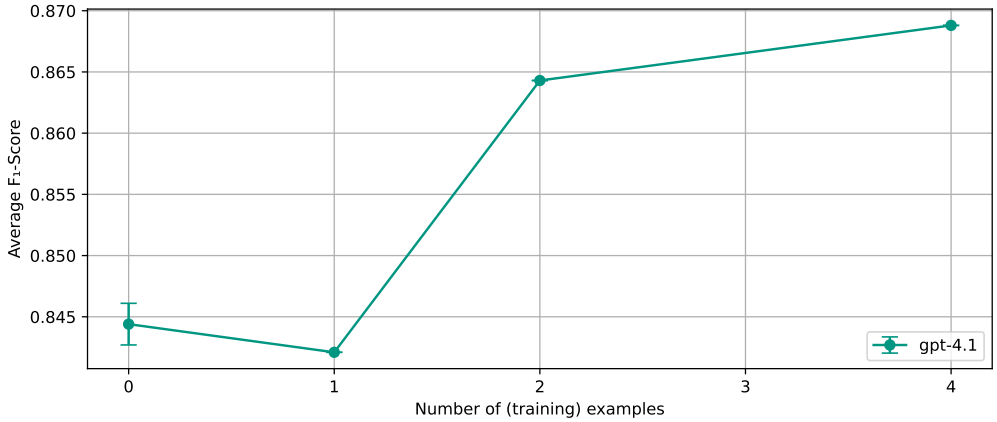

Figure 5.11.: Intra project: Prompting on SecReq with project fold and different amounts of examples

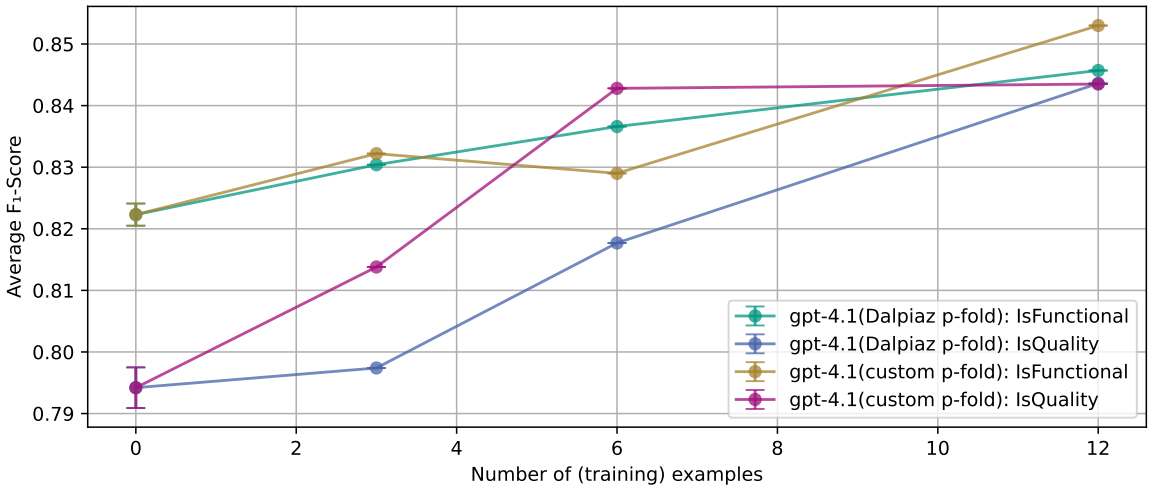

Figure 5.12.: Intra project: Prompting on PROMISE FQ with project folds and different amounts of examples

On the custom p-fold, the results look very different. For the IsQuality classification, the F_1-score increases by nearly 0.03 when performing 2-shot instead of 1-shot, but adding two additional examples, meaning four, for every project shows no visible improvement. The classification of the IsFunctional class however, gets worse from 1-shot to 2-shot and then increases when provided with 4 examples per project. Regardless of the classification or project fold, performance improves when examples are provided rather than none at all.

5.4. Scenario 3 - Intra cross project

In the last scenario external and some internal data is available. As the idea is that a model is fine-tuned on the external data and then further fine-tuned on the internal data, this scenario was only evaluated with fine-tuning. In the case of prompting it is not clear how the

distribution of external and internal requirements should be set to realistically implement this scenario. Therefore, there were no prompting experiments performed. The number of training examples in the diagrams refers to the additional internal requirements that were used to fine-tune the fine-tuned model even further. The second fine-tuning was started with 20 requirements and continuously increased in steps of 20. The highest number was chosen such that at least 10% of the data remains for evaluation. For the PROMISE NFR/FQ datasets, this means that the custom p-fold offered the possibility to add a maximum of 100 requirements and only 80 for the Dalpiaz p-fold. This was because the sizes of the folds in the custom p-fold do not vary as much as in the Dalpiaz p-fold.

5.4.1. Functional vs. Non-Functional Requirements

For the FR vs. NFR task on the last scenario, only the best fine-tuning approach is presented to keep the diagrams easy to read and understand.

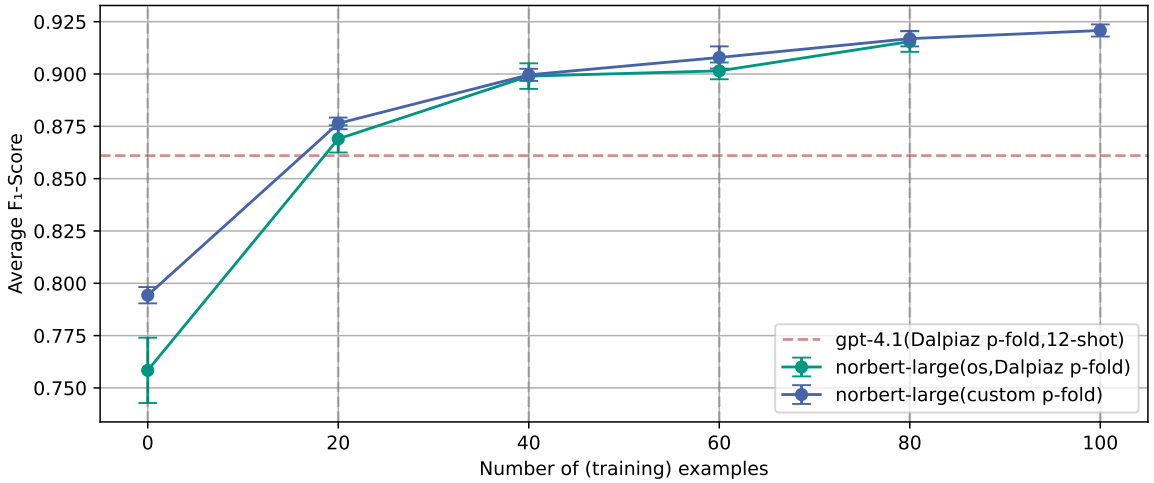

Figure 5.13.: Intra cross project: Fine-tuning on PROMISE NFR with project folds and different amounts of examples

The corresponding fine-tuning results with examples only from the same fold as the one that is used for evaluation are presented in Figure 5.13. Zero training examples in this case refers to the performance on the cross project scenario, so exclusively training on the external projects. While this shows a worse performance compared to the prompting baseline on both p-folds, labeling 20 requirements leads to a better performance than prompting. This is actually the biggest incline, although the performance still increases further when adding more requirements to the training. Using a training set size of 40 requirements reaches an F_1-score of around 0.9 for on both project folds.

5.4.2. Security vs. Non-Security Requirements

For the second task, there are all fine-tuning approaches presented as for SecReq only one project fold was used.

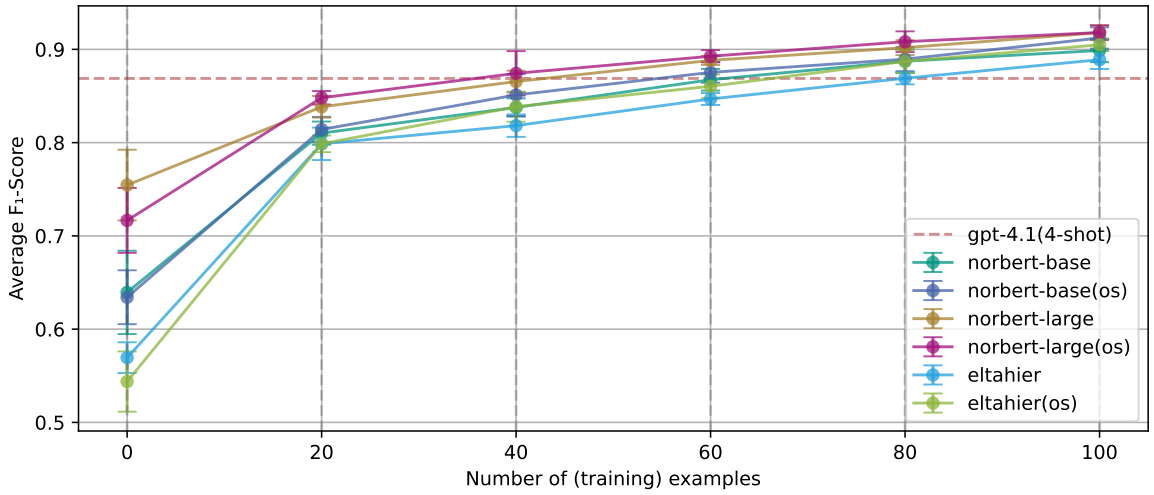

Figure 5.14.: Intra cross project: Fine-tuning on SecReq with project fold and different amounts of examples

The fine-tuning results for the Sec vs. NonSec task are shown in Figure 5.14. $\text{NoRBERT}_{\text{large}}$ performs best and Eltahier worst for all number of training examples regardless of oversampling. This results in $\text{NoRBERT}_{\text{large}}$ intersecting with the prompting baseline at 40 requirements and Eltahier without oversampling only after 80 requirements. Like for the first task, the performance increases when adding more examples to the training and the biggest incline is from having none to 20 internal requirements.

5.4.3. Functional and Quality Requirements

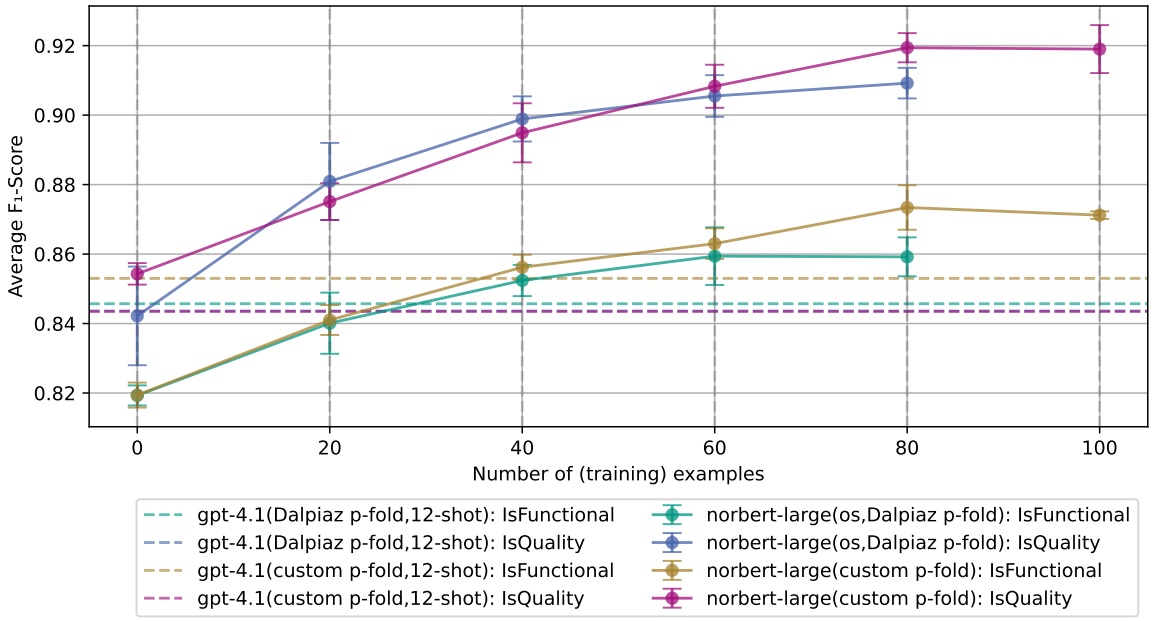

Figure 5.15.: Intra cross project: Fine-tuning on PROMISE FQ with Dalpiaz project fold and different amounts of examples

The results on the Dalpiaz and the custom p-fold for the functional and quality task are presented in Figure 5.15. As for the first task, only the best fine-tuning results are visualized. Until a number of 60 internal examples the performance increases regardless of classification or project fold. While the performance on the IsQuality classification increases further for 80 requirements, it slightly decreases for the IsFunctional classification, on the Dalpiaz p-fold. On the custom p-fold the curves of the classification graphs look similar as the F_1-score increases until a threshold of 80 training examples and decreases slightly when using a total of 100 training examples. The performance difference between the two classification is higher with fine-tuning than with prompting on both project folds. To outperform the prompting baseline on the Dalpiaz p-fold for both classifications, at least around 30 internal requirements have to be labeled. On the custom p-fold almost 40 training examples are needed.

6. Discussion

This chapter discusses the implications that can be drawn from the results presented in the previous chapter. For each scenario, fine-tuning and prompting are compared in order to answer the respective research questions. This is followed by a consideration of threats to validity, limitations, and future work.

6.1. Implications

The results are discussed chronologically in the order the research questions were formulated, so first the cross project scenario and then the two scenarios that include labeling internal requirements. Within a scenario, tasks are discussed independently of one another, and then potential cross-task connections that apply to the entire scenario are examined. At the end, the results from the scenarios are compared to the baseline. As a reminder, two project folds were used for both the first (FR vs. NFR) and last task (functional and quality aspects (F-Q)) for all scenarios: Dalpiaz p-fold and custom p-fold. Again, in all scenarios, internal projects are defined as those for which new requirements should be labeled.

6.1.1. Discussion of Scenario 1 - Cross project

For the first scenario, (training) examples were only taken from projects that were not used for evaluation. All external requirements were taken into account for fine-tuning, which amounted to approximately 500 for both PROMISE datasets on the first and third task and 460 for SecReq on the second task (Sec vs. NonSec).

The discussion starts with the first task of classifying requirements into either functional or non-functional on PROMISE NFR. On the Dalpiaz p-fold, zero-shot prompting is already better than all fine-tuning approaches. On the custom p-fold, zero-shot prompting performs similar to the best fine-tuning approach, but only two examples are enough for prompting to outperform fine-tuning. On the Sec vs. NonSec task the results are even clearer. Using no examples for prompting at all, leads to a 9% higher F_1-score. A reason why the fine-tuning performance on this task is so low, could be that SecReq only consists of three projects, meaning that an average of just around 66% of all the data is used for training instead of around 90% for the FR vs. NFR task. The only task where the performance is balanced between fine-tuning and prompting, is the last task. For the first classification distinguishing only between FR and NFR, 2-shot prompting is slightly better than fine-tuning, whereas

fine-tuning performs slightly better on the classification between quality and non-quality aspects.

The answer to **RQ1** depends on the evaluated task: For both, the FR vs. NFR and Sec vs. NonSec task, zero-shot prompting equals the performance of fine-tuning and even outperforms. And with just two examples, prompting is clearly superior. Although in this case it is assumed that the external requirements for training do not have to be labeled, the data has to be available and the model still has to be fine-tuned. So, if a team can only rely on requirements from external projects, fine-tuning is not worth the effort for the first two tasks. For the classification into functional and quality aspects, the choice depends on the base costs of fine-tuning and prompting. If enough, in this case about 500, examples are available for training, the difference in performance for fine-tuning and prompting is less than 1.5% regardless of classification or project fold. On this task and scenario the standard deviation of the F_1-score for two randomly selected prompting examples is less than 1%. This means that if prompting is cheaper, two labeled requirements are enough to achieve a performance similar to that of fine-tuning.

6.1.2. Discussion of Scenario 2 - Intra project / Scenario 3 - Intra cross project

Since the second scenario was evaluated using only prompting and the third using only fine-tuning, these two scenarios are used to compare the performance when a few internal requirements are manually labeled and used as (training) examples.

To determine the general impact of internal requirements on the performance of fine-tuning and prompting, both strategies are first discussed independently of each other, starting with prompting. On the FR vs. NFR task, the use of requirements from internal instead of external projects as examples, improves the performance for all tested numbers of examples. However, this does not apply to the Sec vs. NonSec task. Here, the performance drops when using internal requirements as examples for the prompts compared to using external ones. The performance on the last task shows a similar trend like the first one. Using examples from the same projects that are being evaluated on, the performance increases regardless of classification or project fold when compared to the cross project scenario. When it comes to fine-tuning, the interpretation of the results is even less complicated: The performance improves for all tasks when the fine-tuned model from the cross project scenario is further fine-tuned using internal examples. For the first and second task the performance increases when using more training examples. On the third task, there is a threshold of 60 requirements the Dalpiaz p-fold and 80 for the custom p-fold beyond which the performance does not improve further for both classifications.

RQ2 can therefore be answered as follows: Manually labeling requirements from the internal projects can have a positive impact on the performance of prompting, but it does depend on the evaluated task. For the FR vs. NFR and F-Q task the performance can be improved by a maximum of nearly 2-3%, whereas the performance on Sec vs. NonSec is not better for four examples and even worse when using less. For fine-tuning, the impact of adding internal requirements is clearly positive. The performance increases on all tasks and

approaches. The biggest improvement can be observed on the Sec vs. NonSec task, where Eltahier with oversampling could increase the F_1-score by 25% when fine-tuned with just 20 internal requirements and even 35% when comparing the cross project model with the one fine-tuned on 100 internal requirements. In general, fine-tuning benefits much more from using internal requirements than prompting does.

To compare fine-tuning and prompting more closely and determine how many internal requirements need to be labeled for fine-tuning to be considered superior, the individual tasks will first be examined separately. On the FR vs. NFR task, the best fine-tuning approach equals the performance of the best prompting approach just below 20 training examples. This applies to both project folds. As GPT-4.1 shows better results on the Sec vs. NonSec task, more internal requirements have to be labeled for fine-tuning to achieve a comparable performance to prompting. This means that for the best fine-tuning approach, nearly 40 requirements have to be labeled. On the F-Q task both classifications have to achieve a better performance with fine-tuning than with prompting in order to conclude that fine-tuning outperforms prompting. So, on the Dalpiaz p-fold around 30 internal requirements are needed, whereas on the custom p-fold, between 35 and 40 requirements are needed to equal the performance of the best prompting approach. GPT-4.1 achieved the best results with 4-shot on all tasks. Please keep in mind that for the first and last task twelve requirements were used in total as 4-shot refers to the number of examples per project and a fold on the PROMISE datasets contained three projects.

Based on that, **RQ3** can be answered. For the FR vs. NFR task, fine-tuning with 20 requirements from internal projects outperforms the best prompting approach, which in this case is 12-shot. With an average total of 125 requirements that have to labeled, this corresponds to $\frac{20}{125} = 16\%$ of the data that has to be labeled for fine-tuning and $\frac{12}{125} = 9.6\%$ for prompting. On the Sec vs. NonSec task, the best fine-tuning approach needs 40 requirements to achieve a better performance than the best prompting approach, that needs four examples. As the projects in SecReq contain a total average of 170 requirements, $\frac{40}{170} \approx 23.5\%$ for fine-tuning and $\frac{4}{170} \approx 2.35\%$ of the data has to be labeled. If considering both project folds on the last task, between $\frac{30}{125} = 24\%$ and $\frac{40}{125} = 32\%$ of the requirements are needed for the best fine-tuning approach to achieve a better performance than 12-shot prompting.

6.1.3. Discussion of Baseline - Mixed project

The last part of this discussion compares the results of the different scenarios with the baseline. Therefore, fine-tuning and prompting are first compared independently across all tasks and scenarios with the baseline. For prompting, the difference between the two scenarios (cross project, intra project) and the baseline heavily depends on the evaluated task. For the first task the performance increases in the following order: baseline, cross project, intra project. It seems that the performance is better if the examples are originating from the same group of projects instead of just sampling randomly. The results on the second task show the opposite pattern. For all evaluated numbers of examples, the performance is best when sampling the examples randomly instead of taking care that they originate from a common group of projects. On the F-Q task, no real pattern like for the other two tasks

can be observed when comparing the scenarios with the baseline. The best approach in the intra scenario shows a better performance than the ones on the cross project scenario and the baseline. The best approaches on the latter two show a very similar performance. Here, there is no clear dependence of the performance on whether the data is sampled randomly from any projects or only from specific projects.

Now, the performance on all tasks is compared for the fine-tuning approaches. For all tasks the curves of the graphs look very similar for the baseline and the intra cross project. For these two, the best performance is comparable for the first and last task. On the Sec vs. NonSec task, the baseline is even better than the intra cross project when using the maximum amount of requirements for training, which means that 10% of the data remains for evaluation. So, when using about 90% of the data on PROMISE NFR/FQ and 66% on SecReq for training, it does not make a big difference whether the model is fine-tuned completely on this data or the model is first fine-tuned on a group of projects and then further fine-tuned on the remaining gorup of projects. All approaches show a worse performance on unseen projects than using the same number of requirements for the baseline, where a requirement could originate from any project.

The answer for **RQ4** can be summed up as follows: For fine-tuning the performance does rely significantly on the sampling method of the training examples. So, whether requirements originate from any project or from a fixed group of projects. In contrast to that, prompting does not depend as much on the sampling method of examples. In comparison to fine-tuning the performance is quite similar regardless of how the examples are chosen.

6.2. Threats to Validity

Having addressed the research questions and discussed the key findings, it is essential to critically examine the potential threats to validity that may influence the interpretation and generalizability of these results.

6.2.1. Construct Validity

A possible threat to construct validity arises from the used evaluation metrics. To compare different approaches [11], four commonly used metrics were used and presented with their mean and standard deviation: Precision, Recall, F_1-score and F_2-score. Another potential threat relates to the datasets and the quality of their annotations. Although, all datasets in thesis were already used in prior studies, there could be limitations to the quality of the annotations, especially for PROMISE NFR as this was labeled by students and not professionals. Another aspect affecting construct validity are the chosen prompts as they heavily influence the model behavior. To mitigate this risk, only prompts were used that were presented in previous research.

6.2.2. Internal Validity

A potential threat to internal validity arises from the selected requirements for training and testing the models as performance differences could be influenced by this, especially when only few examples were used. To mitigate this thread, k-fold and p-fold cross validation were used, and an average was calculated across all folds. To mitigate possible implementation errors, libraries and packages from recommended widely accepted resources were used [17, 44]. Since the models used in this study were pre-trained and the datasets were sourced from open-source repositories, there is a potential risk that the models may have already seen the data, or parts of it, before the experiments were conducted.

6.2.3. External Validity

Like already mentioned, the datasets are open-source and only contain a little amount of requirements. This could be a threat to external validity as it could be that the results are not transferable to larger and closed-sourced datasets. To show the generalizability, all experiments except prompting on the second scenario, were performed with four different random seeds and the temperature for GPT-4.1 was set to zero. The seeds were further fixed to ensure reliability and reproducibility. Additionally, different realistic application scenarios were evaluated and compared to a baseline that was used widely in prior studies. The code and results for all experiments is available in a replication package at Zenodo [36].

6.3. Limitations and Future Work

There are various limitations to this work due to limited resources, such as time or money. However, those limitations also offer potential for future work.

Only GPT-4.1 was examined for the prompting experiments. However, Chapter 3 shows that the performance is highly influenced by the model and technique used to adapt this model. With the continuous improvement of LLMs and newer versions being published, future work should investigate other models like Gemini, Llama, DeepSeek or newer GPT-versions on the same experiments to improve the comparison between fine-tuning and prompting with more results to rely on. This also includes experiments with other fine-tuning approaches like SetFit [34, 38], DoRa [31] or Prompt-Tuning [27].

In this work, fine-tuning was not evaluated on the only intra project scenario, and prompting was not evaluated on the intra cross project scenario. Such experiments could examine the influence of internal examples in even greater detail.

Another limitation of this thesis is the datasets and the way that they were used. Meaning that security-related requirements were only taken from SecReq but could also be taken from PROMISE NFR or other datasets that contain security-related requirements as a subclass

of non-functional requirements. In addition to that, more datasets, and especially larger datasets, could be used to examine what influence the amount of data has on the results.

7. Conclusion

The thesis aimed to investigate the performance of fine-tuning and prompting for the task of requirements classification based on the number of available examples.

The central research questions were as follows:

1. *How do fine-tuning and prompting compare when classifying requirements on unseen projects?*
2. *What impact has the adding of internal requirements on the performance of both fine-tuning and prompting?*
3. *How many requirements from the same project have to be labeled so that it is worth fine-tuning an LLM instead of performing few-shot prompting?*
4. *How does the performance on different scenarios compare to the baseline?*

The experiments were evaluated on three different application scenarios and a baseline. For this, three different tasks were examined in each case with three datasets, one for each task. The scenarios were created in a way that they represent real-world situations. State-of-the-art methods were identified to compare fine-tuning and prompting on the different tasks and scenarios. Here, every method was evaluated on all tasks to ensure that the results are comparable.

The experiments showed that fine-tuning is much more sensitive to the distribution of data for training and testing, meaning that the performance declines significantly if the models were classifying requirements on unseen projects and inclines when projects are overlapping between training and testing. In contrast, no such trend is observed with prompting, as performance there tends to depend on the specific task: For example, prompting without any examples can achieve a better performance than fine-tuning with 340 to 500 requirements, when evaluated on unseen projects. Additionally, the fine-tuning performance increased on most experiments when using a larger train set, which was not the case with prompting. Depending on the task, 20 to 40 requirements from the same project have to be labeled for fine-tuning to outperform the best prompting approach (4 to 12 requirements). As requirements from any projects in a dataset are used for the baseline, the performance with fine-tuning can only be matched if adding internal requirements to the training, but not if only relying on external requirements.

However, the results of this thesis are only applicable to the approaches and datasets used. With more data or other models, different performance results might be obtained. The comparison should therefore be expanded in future studies.

This study addressed the lack of guidelines for when to use fine-tuning and prompting for requirements classification by systematically comparing these two strategies across different real-world scenarios. The thesis was able to show that labeling just 20 requirements manually can improve the performance of fine-tuning by over 20 percentage points.

Bibliography

[1] Muhammad Aasem, Muhammad Ramzan, and Arfan Jaffar. "Analysis and Optimization of Software Requirements Prioritization Techniques". In: *2010 International Conference on Information and Emerging Technologies*. 2010 International Conference on Information and Emerging Technologies (ICIET). Karachi, Pakistan: IEEE, June 2010, pp. 1–6. ISBN: 978-1-4244-8001-2. DOI: 10.1109/ICIET.2010.5625687. URL: http://ieeexplore.ieee.org/document/5625687/.

[2] Zahra Shakeri Hossein Abad et al. "What Works Better? A Study of Classifying Requirements". In: *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 2017 IEEE 25th International Requirements Engineering Conference (RE). Lisbon, Portugal: IEEE, Sept. 2017, pp. 496–501. ISBN: 978-1-5386-3191-1. DOI: 10.1109/RE.2017.36. URL: http://ieeexplore.ieee.org/document/8049172/.

[3] Aisvarya Adeseye, Jouni Isoaho, and Mohammad Tahir. "Systematic Prompt Framework for Qualitative Data Analysis: Designing System and User Prompts". In: *2025 IEEE 5th International Conference on Human-Machine Systems (ICHMS)*. 2025 IEEE 5th International Conference on Human-Machine Systems (ICHMS). Abu Dhabi, United Arab Emirates: IEEE, May 26, 2025, pp. 229–234. ISBN: 979-8-3315-2164-6. DOI: 10.1109/ICHMS65439.2025.11154183. URL: https://ieeexplore.ieee.org/document/11154183/.

[4] Waad Alhoshan, Alessio Ferrari, and Liping Zhao. "Zero-Shot Learning for Requirements Classification: An Exploratory Study". In: *Inf. Softw. Technol.* 159.C (July 1, 2023). ISSN: 0950-5849. DOI: 10.1016/j.infsof.2023.107202. URL: https://doi.org/10.1016/j.infsof.2023.107202.

[5] Manal Binkhonain and Reem Alfayez. "Are Prompts All You Need? Evaluating Prompt-Based Large Language Models (LLM)s for Software Requirements Classification". In: *Requirements Engineering* (Sept. 16, 2025). ISSN: 1432-010X. DOI: 10.1007/s00766-025-00451-8. URL: https://doi.org/10.1007/s00766-025-00451-8.

[6] Ranit Chatterjee et al. "A Pipeline for Automating Labeling to Prediction in Classification of NFRs". In: *2021 IEEE 29th International Requirements Engineering Conference (RE)*. 2021 IEEE 29th International Requirements Engineering Conference (RE). Notre Dame, IN, USA: IEEE, Sept. 2021, pp. 323–323. ISBN: 978-1-6654-2856-9. DOI: 10.1109/RE51729.2021.00036. URL: https://ieeexplore.ieee.org/document/9604524/.

[7] Krishna Teja Chitty-Venkata et al. "Neural Architecture Search for Transformers: A Survey". In: *IEEE Access* 10 (2022), pp. 108374–108412. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3212767. URL: https://ieeexplore.ieee.org/document/9913476/.

[8] Jane Cleland-Huang et al. “Automated Classification of Non-Functional Requirements”. In: *Requirements Engineering* 12.2 (Apr. 2007), pp. 103–120. ISSN: 0947-3602, 1432-010X. DOI: 10.1007/s00766-007-0045-1. URL: https://link.springer.com/10.1007/s00766-007-0045-1.

[9] Jane Cleland-Huang et al. *Nfr*. Zenodo, Mar. 17, 2007. DOI: 10.5281/ZENODO.268542. URL: https://zenodo.org/record/268542.

[10] Fabiano Dalpiaz et al. “Requirements Classification with Interpretable Machine Learning and Dependency Parsing”. In: *2019 IEEE 27th International Requirements Engineering Conference (RE)*. 2019 IEEE 27th International Requirements Engineering Conference (RE). Jeju Island, Korea (South): IEEE, Sept. 2019, pp. 142–152. ISBN: 978-1-7281-3912-8. DOI: 10.1109/RE.2019.00025. URL: https://ieeexplore.ieee.org/document/8920401/.

[11] Davide Dell’Anna, Fatma Başak Aydemir, and Fabiano Dalpiaz. “Evaluating Classifiers in SE Research: The ECSER Pipeline and Two Replication Studies”. In: *Empirical Software Engineering* 28.1 (Nov. 8, 2022). ISSN: 1573-7616. DOI: 10.1007/s10664-022-10243-1. URL: https://doi.org/10.1007/s10664-022-10243-1.

[12] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North*. Proceedings of the 2019 Conference of the North. Minneapolis, Minnesota: Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: http://aclweb.org/anthology/N19-1423.

[13] Ning Ding et al. “Parameter-Efficient Fine-Tuning of Large-Scale Pre-Trained Language Models”. In: *Nature Machine Intelligence* 5.3 (Mar. 2, 2023), pp. 220–235. ISSN: 2522-5839. DOI: 10.1038/s42256-023-00626-4. URL: https://www.nature.com/articles/s42256-023-00626-4.

[14] Jonas Eckhardt, Andreas Vogelsang, and Daniel Méndez Fernández. “Are "Non-Functional" Requirements Really Non-Functional?: An Investigation of Non-Functional Requirements in Practice”. In: *Proceedings of the 38th International Conference on Software Engineering*. ICSE ’16: 38th International Conference on Software Engineering. Austin Texas: ACM, May 14, 2016, pp. 832–842. ISBN: 978-1-4503-3900-1. DOI: 10.1145/2884781.2884788. URL: https://dl.acm.org/doi/10.1145/2884781.2884788.

[15] Safaa Eltahier, Omer Dawood, and Imtithal Saeed. “BERT Fine-Tuning for Software Requirement Classification: Impact of Model Components and Dataset Size”. In: *Information* 16.11 (Nov. 13, 2025), p. 981. ISSN: 2078-2489. DOI: 10.3390/info16110981. URL: https://www.mdpi.com/2078-2489/16/11/981.

[16] Federal University of Maranhão. *Promise+: A Dataset of Labeled Software Requirements*. Zenodo, July 24, 2024. DOI: 10.5281/ZENODO.12805484. URL: https://zenodo.org/doi/10.5281/zenodo.12805484.

[17] Julian Frattini et al. “NLP4RE Tools: Classification, Overview and Management”. In: *Handbook on Natural Language Processing for Requirements Engineering*. Ed. by Alessio Ferrari and Gouri Ginde. Cham: Springer Nature Switzerland, 2025, pp. 357–380. ISBN: 978-3-031-73142-6 978-3-031-73143-3. DOI: 10.1007/978-3-031-73143-3_13. URL: https://link.springer.com/10.1007/978-3-031-73143-3_13.

[18] Martin Glinz. “Rethinking the Notion of Non-Functional Requirements”. In: Proceedings of the Third World Congress for Software Quality. Vol. 2. Munich, Germany, Sept. 2005, pp. 55–64. URL: https://files.ifi.uzh.ch/rerg/amadeus/publications/papers/3WCSQ2005.pdf.

[19] Lu Han, QiXiang Zhou, and Tong Li. “Improving Requirements Classification Models Based on Explainable Requirements Concerns”. In: *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. 2023 IEEE 31st International Requirements Engineering Conference Workshops (REW). Hannover, Germany: IEEE, Sept. 2023, pp. 95–101. ISBN: 979-8-3503-2691-8. DOI: 10.1109/REW57809.2023.00023. URL: https://ieeexplore.ieee.org/document/10260874/.

[20] Tobias Hey et al. “NoRBERT: Transfer Learning for Requirements Classification”. In: *2020 IEEE 28th International Requirements Engineering Conference (RE)*. 2020 IEEE 28th International Requirements Engineering Conference (RE). Aug. 2020, pp. 169–179. DOI: 10.1109/RE48521.2020.00028.

[21] Ishrar Hussain, Leila Kosseim, and Olga Ormandjieva. “Using Linguistic Knowledge to Classify Non-functional Requirements in SRS Documents”. In: *Natural Language and Information Systems*. Ed. by Epaminondas Kapetanios, Vijayan Sugumaran, and Myra Spiliopoulou. Vol. 5039. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 287–298. ISBN: 978-3-540-69857-9 978-3-540-69858-6. DOI: 10.1007/978-3-540-69858-6_28. URL: http://link.springer.com/10.1007/978-3-540-69858-6_28.

[22] *ISO 25010*. URL: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010.

[23] Eric Knauss et al. *SecReq*. Zenodo, Feb. 10, 2021. DOI: 10.5281/zenodo.4530183. URL: https://zenodo.org/records/4530183.

[24] Zijad Kurtanović and Walid Maalej. “Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning”. In: *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 2017 IEEE 25th International Requirements Engineering Conference (RE). Sept. 2017, pp. 490–495. DOI: 10.1109/RE.2017.82. URL: https://ieeexplore.ieee.org/abstract/document/8049171.

[25] Feng-Lin Li et al. “Non-Functional Requirements as Qualities, with a Spice of Ontology”. In: *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. 2014 IEEE 22nd International Requirements Engineering Conference (RE). Karlskrona, Sweden: IEEE, Aug. 2014, pp. 293–302. ISBN: 978-1-4799-3033-3 978-1-4799-3031-9. DOI: 10.1109/RE.2014.6912271. URL: http://ieeexplore.ieee.org/document/6912271/.

[26] Shih-Yang Liu et al. “DoRA: Weight-Decomposed Low-Rank Adaptation”. In: *Proceedings of the 41st International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, July 8, 2024, pp. 32100–32121. URL: https://proceedings.mlr.press/v235/liu24bn.html.

[27] Xianchang Luo et al. “PRCBERT: Prompt Learning for Requirement Classification Using BERT-based Pretrained Language Models”. In: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. ASE ’22. New York, NY, USA: Association for Computing Machinery, Jan. 5, 2023, pp. 1–13. ISBN: 978-1-4503-9475-8. DOI: 10.1145/3551349.3560417. URL: https://dl.acm.org/doi/10.1145/3551349.3560417.

[28] Murilo Martin et al. “Evaluating the Potential of Large Language Models in Security-Related Software Requirements Classification”. In: *Anais Do XXXIX Simpósio Brasileiro de Engenharia de Software (SBES 2025)*. Simpósio Brasileiro de Engenharia de Software. Brasil: Sociedade Brasileira de Computação, Sept. 22, 2025, pp. 315–325. DOI: 10.5753/sbes.2025.9935. URL: https://sol.sbc.org.br/index.php/sbes/article/view/37009.

[29] Prudence Kadebu. *DOSSPRE*. Mendeley, July 7, 2022. DOI: 10.17632/23XTBVK6YP.1. URL: https://data.mendeley.com/datasets/23xtbvk6yp/1.

[30] Alec Radford et al. “Improving Language Understanding by Generative Pre-Training”. In: (2018). URL: https://www.mikecaptain.com/resources/pdf/GPT-1.pdf.

[31] Gokul Rejithkumar and Preethu Rose Anish. “NICE: Non-Functional Requirements Identification, Classification, and Explanation Using Small Language Models”. In: *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). Ottawa, ON, Canada: IEEE, Apr. 27, 2025, pp. 284–295. ISBN: 979-8-3315-3685-5. DOI: 10.1109/ICSE-SEIP66354.2025.00031. URL: https://ieeexplore.ieee.org/document/11121724/.

[32] Mehrdad Sabetzadeh and Chetan Arora. “Practical Guidelines for the Selection and Evaluation of Natural Language Processing Techniques in Requirements Engineering”. In: *Handbook on Natural Language Processing for Requirements Engineering*. Ed. by Alessio Ferrari and Gouri Ginde. Cham: Springer Nature Switzerland, 2025, pp. 407–433. ISBN: 978-3-031-73142-6 978-3-031-73143-3. DOI: 10.1007/978-3-031-73143-3_15. URL: https://link.springer.com/10.1007/978-3-031-73143-3_15.

[33] Abhishek Sainani et al. “Extracting and Classifying Requirements from Software Engineering Contracts”. In: *2020 IEEE 28th International Requirements Engineering Conference (RE)*. 2020 IEEE 28th International Requirements Engineering Conference (RE). Zurich, Switzerland: IEEE, Aug. 2020, pp. 147–157. ISBN: 978-1-7281-7438-9. DOI: 10.1109/RE48521.2020.00026. URL: https://ieeexplore.ieee.org/document/9218214/.

[34] Md Shafikuzzaman et al. “On the Effectiveness of Zero-Shot and Few-Shot Pretrained Language Models for Software Requirement Classification”. In: *IEEE Access* 13 (2025), pp. 159439–159453. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2025.3607813. URL: https://ieeexplore.ieee.org/document/11153850.

[35] Kunal Singh, Santosh Deshpande, and Swapnaja Patwardhan. “Refining the Giants: A Comprehensive Review of Fine-Tuning Strategies for Large Language Models”. In: *Computing and Machine Learning*. Ed. by Jagdish Chand Bansal et al. Vol. 1144. Singapore: Springer Nature Singapore, 2025, pp. 65–82. ISBN: 978-981-97-7838-6 978-981-97-7839-3. DOI: 10.1007/978-981-97-7839-3_5. URL: https://link.springer.com/10.1007/978-981-97-7839-3_5.

[36] Maximilian Supp. “Fine-Tuning vs. Prompting: The Case of Requirements Classifications”. Zenodo, Mar. 12, 2026. DOI: 10.5281/ZENODO.18979584. URL: https://doi.org/10.5281/zenodo.18979584.

[37] Yongjian Tang et al. *The Few-shot Dilemma: Over-prompting Large Language Models*. Sept. 16, 2025. DOI: 10.48550/arXiv.2509.13196. arXiv: 2509.13196 [cs]. URL: http://arxiv.org/abs/2509.13196. Pre-published.

[38] Lewis Tunstall et al. *Efficient Few-Shot Learning Without Prompts*. Version 1. 2022. DOI: 10.48550/ARXIV.2209.11055. URL: https://arxiv.org/abs/2209.11055. Pre-published.

[39] Gael Varoquaux and Olivier Colliot. “Evaluating Machine Learning Models and Their Diagnostic Value”. In: *Machine Learning for Brain Disorders*. Ed. by Olivier Colliot. Vol. 197. New York, NY: Springer US, 2023, pp. 601–630. ISBN: 978-1-0716-3194-2 978-1-0716-3195-9. DOI: 10.1007/978-1-0716-3195-9_20. URL: https://link.springer.com/10.1007/978-1-0716-3195-9_20.

[40] Ashish Vaswani et al. “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[41] Andreas Vogelsang and Jannik Fischbach. “Using Large Language Models for Natural Language Processing Tasks in Requirements Engineering: A Systematic Guideline”. In: *Handbook on Natural Language Processing for Requirements Engineering*. Ed. by Alessio Ferrari and Gouri Ginde. Cham: Springer Nature Switzerland, 2025, pp. 435–456. ISBN: 978-3-031-73142-6 978-3-031-73143-3. DOI: 10.1007/978-3-031-73143-3_16. URL: https://link.springer.com/10.1007/978-3-031-73143-3_16.

[42] Colin Wei, Sang Michael Xie, and Tengyu Ma. “Why Do Pretrained Language Models Help in Downstream Tasks? An Analysis of Head and Prompt Tuning”. In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 16158–16170. URL: https://proceedings.neurips.cc/paper_files/paper/2021/hash/86b3e165b8154656a71ffe8a327ded7d-Abstract.html.

[43] Mohammad Amin Zadenoori et al. “Automatic Prompt Engineering: The Case of Requirements Classification”. In: *Requirements Engineering: Foundation for Software Quality: 31st International Working Conference, REFSQ 2025, Barcelona, Spain, April 7–10, 2025, Proceedings*. Berlin, Heidelberg: Springer-Verlag, Apr. 7, 2025, pp. 217–225. ISBN: 978-3-031-88530-3. DOI: 10.1007/978-3-031-88531-0_15. URL: https://doi.org/10.1007/978-3-031-88531-0_15.

[44] Liping Zhao and Waad Alhoshan. “Machine Learning for Requirements Classification”. In: *Handbook on Natural Language Processing for Requirements Engineering*. Ed. by Alessio Ferrari and Gouri Ginde. Cham: Springer Nature Switzerland, 2025, pp. 19–59. ISBN: 978-3-031-73143-3. DOI: 10.1007/978-3-031-73143-3_2. URL: https://doi.org/10.1007/978-3-031-73143-3_2.

A. Appendix

A.1. Supplementary Material for Baseline - Mixed project

Approach	# Examples	P	R	F1	F2
gpt-4.1	0	.8359 ± .0021	.8378 ± .0022	.8108 ± .0024	.8374 ± .0022
	1	.8412 ± .0035	.8459 ± .0050	.8211 ± .0063	.8450 ± .0047
	2	.8439 ± .0029	.8500 ± .0044	.8263 ± .0057	.8488 ± .0040
	4	.8428 ± .0059	.8492 ± .0072	.8258 ± .0081	.8479 ± .0069
	8	.8443 ± .0025	.8507 ± .0025	.8270 ± .0025	.8494 ± .0025
gpt-4.1(cot)	0	.8169 ± .0004	.8079 ± .0006	.7746 ± .0007	.8097 ± .0006
	1	.8245 ± .0031	.8207 ± .0066	.7903 ± .0087	.8214 ± .0059
	2	.8304 ± .0022	.8288 ± .0036	.7996 ± .0046	.8291 ± .0033
	4	.8306 ± .0048	.8319 ± .0065	.8048 ± .0076	.8317 ± .0061
	8	.8316 ± .0039	.8317 ± .0047	.8036 ± .0052	.8317 ± .0045

Table A.1.: Mixed project: Prompting on PROMISE NFR with different amounts of (training) examples

Approach	# Examples	P	R	F1	F2
norbert-base	64	.8201 ± .0116	.7893 ± .0225	.7968 ± .0215	.7952 ± .0203
	128	.8702 ± .0053	.8618 ± .0074	.8652 ± .0065	.8635 ± .0068
	256	.8903 ± .0046	.8844 ± .0051	.8870 ± .0048	.8856 ± .0050
	512	.9229 ± .0029	.9176 ± .0042	.9200 ± .0036	.9186 ± .0040
	562	.9198 ± .0035	.9142 ± .0024	.9167 ± .0025	.9153 ± .0024
norbert-base(os)	64	.7958 ± .0112	.7823 ± .0107	.7870 ± .0109	.7850 ± .0108
	128	.8561 ± .0099	.8531 ± .0107	.8545 ± .0103	.8537 ± .0105
	256	.8950 ± .0132	.8925 ± .0134	.8937 ± .0133	.8930 ± .0133
	512	.9151 ± .0044	.9087 ± .0054	.9116 ± .0050	.9100 ± .0052
	562	.9225 ± .0048	.9173 ± .0044	.9196 ± .0044	.9183 ± .0044
norbert-large	64	.7865 ± .0199	.7556 ± .0230	.7627 ± .0235	.7616 ± .0224
	128	.8588 ± .0120	.8531 ± .0114	.8553 ± .0113	.8542 ± .0112
	256	.9023 ± .0066	.8941 ± .0056	.8976 ± .0060	.8957 ± .0058
	512	.9258 ± .0032	.9190 ± .0034	.9220 ± .0032	.9203 ± .0033
	562	.9263 ± .0050	.9226 ± .0040	.9244 ± .0044	.9234 ± .0042
norbert-large(os)	64	.7914 ± .0127	.7714 ± .0095	.7773 ± .0102	.7753 ± .0100
	128	.8535 ± .0069	.8432 ± .0098	.8472 ± .0087	.8452 ± .0092
	256	.8999 ± .0041	.8924 ± .0054	.8956 ± .0047	.8939 ± .0050
	512	.9269 ± .0102	.9195 ± .0112	.9227 ± .0107	.9210 ± .0110
	562	.9298 ± .0095	.9225 ± .0096	.9257 ± .0096	.9239 ± .0096
eltahier	64	.7615 ± .0057	.7087 ± .0111	.7141 ± .0120	.7186 ± .0101
	128	.8256 ± .0052	.8121 ± .0021	.8169 ± .0024	.8148 ± .0022
	256	.8863 ± .0113	.8820 ± .0121	.8838 ± .0117	.8828 ± .0118
	512	.9222 ± .0039	.9135 ± .0045	.9172 ± .0042	.9152 ± .0043
	562	.9217 ± .0046	.9149 ± .0051	.9179 ± .0049	.9162 ± .0050
eltahier(os)	64	.7709 ± .0085	.7369 ± .0048	.7435 ± .0045	.7434 ± .00405
	128	.8420 ± .0063	.8312 ± .0098	.8352 ± .0086	.8333 ± .0090
	256	.8928 ± .0050	.8851 ± .0069	.8884 ± .0062	.8866 ± .0065
	512	.9204 ± .0021	.9103 ± .0031	.9146 ± .0028	.9123 ± .0029
	562	.9268 ± .0036	.9181 ± .0029	.9219 ± .0032	.9198 ± .0030

Table A.2.: Mixed project: Fine-tuning on PROMISE NFR with different amounts of (training) examples

Approach	# Examples	P	R	F1	F2
gpt-4.1	0	.8389 ± .0016	.8586 ± .0017	.8444 ± .0017	.8546 ± .0017
	1	.8605 ± .0035	.8777 ± .0022	.8664 ± .0034	.8742 ± .0024
	2	.8653 ± .0052	.8819 ± .0059	.8713 ± .0054	.8785 ± .0057
	4	.8720 ± .0051	.8889 ± .0059	.8780 ± .0052	.8854 ± .0057
	8	.8714 ± .0037	.8865 ± .0051	.8771 ± .0040	.8835 ± .0048

Table A.3.: Mixed project: Prompting on SecReq with different amounts of (training) examples

Approach	# Examples	P	R	F1	F2
norbert-base	64	.7596 ± .0092	.7306 ± .0073	.7391 ± .0070	.7362 ± .0069
	128	.8190 ± .0055	.8071 ± .0063	.8120 ± .0052	.8095 ± .0056
	256	.8786 ± .0066	.8863 ± .0056	.8820 ± .0061	.8848 ± .0057
	459	.9144 ± .0039	.9239 ± .0040	.9186 ± .0039	.9220 ± .0040
norbert-base(os)	64	.7647 ± .0168	.7452 ± .0153	.7521 ± .0158	.7490 ± .0155
	128	.8291 ± .0090	.8254 ± .0088	.8271 ± .0088	.8261 ± .0087
	256	.8740 ± .0030	.8820 ± .0076	.8773 ± .0045	.8804 ± .0066
	459	.9178 ± .0047	.9222 ± .0045	.9197 ± .0040	.9213 ± .0041
norbert-large	64	.7777 ± .0134	.7537 ± .0255	.7607 ± .0229	.7583 ± .0226
	128	.8506 ± .0120	.8622 ± .0150	.8553 ± .0129	.8599 ± .0144
	256	.8858 ± .0102	.8961 ± .0094	.8902 ± .0098	.8940 ± .0095
	459	.9248 ± .0041	.9344 ± .0057	.9291 ± .0046	.9325 ± .0053
norbert-large(os)	64	.8017 ± .0074	.7972 ± .0073	.7991 ± .0067	.7981 ± .0070
	128	.8530 ± .0068	.8621 ± .0087	.8568 ± .0075	.8602 ± .0083
	256	.8950 ± .0046	.9038 ± .0090	.8987 ± .0062	.9020 ± .0081
	459	.9285 ± .0074	.9331 ± .0075	.9307 ± .0074	.9322 ± .0074
eltahier	64	.7216 ± .0111	.6314 ± .0054	.6302 ± .0069	.6475 ± .0057
	128	.7797 ± .0147	.7501 ± .0197	.7593 ± .0190	.7559 ± .0188
	256	.8436 ± .0103	.8473 ± .0095	.8452 ± .0098	.8465 ± .0096
	459	.9020 ± .0063	.9082 ± .0057	.9047 ± .0055	.9070 ± .0055
eltahier(os)	64	.7375 ± .0073	.6951 ± .0058	.7041 ± .0062	.7032 ± .0059
	128	.8132 ± .0027	.8006 ± .0070	.8058 ± .0054	.8031 ± .0061
	256	.8571 ± .0031	.8618 ± .0030	.8590 ± .0016	.8609 ± .0020
	459	.9154 ± .0039	.9173 ± .0071	.9163 ± .0051	.9170 ± .0063

Table A.4.: Mixed project: Fine-tuning on SecReq with different amounts of (training) examples

Approach	# Examples	IsFunctional			
		P	R	F1	F2
gpt-4.1	0	.8466 ± .0022	.8258 ± .0018	.8223 ± .0018	.8299 ± .0019
	1	.8315 ± .0027	.8149 ± .0041	.8118 ± .0044	.8182 ± .0038
	2	.8396 ± .0041	.8276 ± .0049	.8254 ± .0052	.8299 ± .0048
	4	.8369 ± .0057	.8252 ± .0076	.8230 ± .0081	.8275 ± .0073
	8	.8472 ± .0080	.8360 ± .0101	.8339 ± .0105	.8382 ± .0096

Table A.5.: Mixed project: Prompting on PROMISE FQ with different amounts of (training) examples (classification: IsFunctional)

Approach	# Examples	IsQuality			
		P	R	F1	F2
gpt-4.1	0	.8218 ± .0024	.8292 ± .0029	.7942 ± .0033	.8277 ± .0028
	1	.8341 ± .0032	.8459 ± .0049	.8144 ± .0066	.8435 ± .0045
	2	.8512 ± .0082	.8686 ± .0092	.8443 ± .0107	.8651 ± .0090
	4	.8467 ± .0072	.8628 ± .0088	.8366 ± .0113	.8595 ± .0085
	8	.8529 ± .0033	.8700 ± .0037	.8441 ± .0042	.8665 ± .0036

Table A.6.: Mixed project: Prompting on PROMISE FQ with different amounts of (training) examples (classification: IsQuality)

Approach	# Examples	IsOnlyFunctional			
		P	R	F1	F2
gpt-4.1	0	.8386 ± .0028	.8630 ± .0032	.8289 ± .0036	.8580 ± .0031
	1	.7727 ± .0164	.8694 ± .0146	.7933 ± .0180	.8482 ± .0149
	2	.7870 ± .0079	.8735 ± .0018	.8111 ± .0073	.8547 ± .0023
	4	.7830 ± .0113	.8717 ± .0058	.8064 ± .0113	.8523 ± .0069
	8	.7962 ± .0151	.8853 ± .0141	.8212 ± .0159	.8659 ± .0139

Table A.7.: Mixed project: Prompting on PROMISE FQ with different amounts of (training) examples (classification: IsOnlyFunctional)

Approach	# Examples	IsOnlyQuality			
		P	R	F1	F2
gpt-4.1	0	.8279 ± .0023	.7859 ± .0017	.7832 ± .0018	.7939 ± .0018
	1	.8446 ± .0138	.7356 ± .0125	.7581 ± .0141	.7551 ± .0123
	2	.8542 ± .0038	.7698 ± .0056	.7924 ± .0049	.7853 ± .0052
	4	.8538 ± .0163	.7659 ± .0166	.7885 ± .0177	.7819 ± .0162
	8	.8580 ± .0079	.7695 ± .0097	.7926 ± .0094	.7857 ± .0087

Table A.8.: Mixed project: Prompting on PROMISE FQ with different amounts of (training) examples (classification: IsOnlyQuality)

Approach	# Examples	IsFunctional			
		P	R	F1	F2
norbert-base	64	.7235 ± .0069	.7230 ± .0072	.7227 ± .0075	.7231 ± .0072
	128	.7994 ± .0068	.7992 ± .0069	.7992 ± .0069	.7992 ± .0069
	256	.8400 ± .0090	.8396 ± .0090	.8396 ± .0091	.8396 ± .0090
	512	.8601 ± .0034	.8600 ± .0033	.8600 ± .0033	.8600 ± .0033
	562	.8656 ± .0085	.8656 ± .0086	.8656 ± .0085	.8656 ± .0085
norbert-base(os)	64	.7235 ± .0069	.7230 ± .0072	.7227 ± .0075	.7231 ± .0072
	128	.7994 ± .0068	.7992 ± .0069	.7992 ± .0069	.7992 ± .0069
	256	.8411 ± .0080	.8408 ± .0081	.8407 ± .0082	.8409 ± .0081
	512	.8641 ± .0059	.8639 ± .0059	.8640 ± .0059	.8640 ± .0059
	562	.8640 ± .0049	.8640 ± .0049	.8640 ± .0049	.8640 ± .0049
norbert-large	64	.7337 ± .0250	.7330 ± .0252	.7326 ± .0253	.7331 ± .0252
	128	.8157 ± .0135	.8155 ± .0132	.8156 ± .0132	.8156 ± .0132
	256	.8355 ± .0081	.8350 ± .0085	.8348 ± .0086	.8351 ± .0084
	512	.8688 ± .0039	.8683 ± .0038	.8683 ± .0038	.8684 ± .0038
	562	.8686 ± .0112	.8683 ± .0111	.8684 ± .0111	.8684 ± .0111
norbert-large(os)	64	.7337 ± .0250	.7330 ± .0252	.7326 ± .0253	.7331 ± .0252
	128	.8157 ± .0135	.8155 ± .0132	.8156 ± .0132	.8156 ± .0132
	256	.8401 ± .0074	.8394 ± .0080	.8392 ± .0081	.8395 ± .0079
	512	.8746 ± .0037	.8743 ± .0034	.8744 ± .0035	.8744 ± .0035
	562	.8719 ± .0068	.8715 ± .0064	.8716 ± .0064	.8716 ± .0065
eltahier	64	.7008 ± .0084	.6988 ± .0089	.6977 ± .0092	.6992 ± .0087
	128	.7742 ± .0084	.7738 ± .0086	.7736 ± .0086	.7738 ± .0085
	256	.8291 ± .0119	.8286 ± .0124	.8284 ± .0125	.8287 ± .0123
	512	.8561 ± .0040	.8561 ± .0041	.8560 ± .0041	.8561 ± .0040
	562	.8609 ± .0033	.8609 ± .0034	.8608 ± .0034	.8609 ± .0034
eltahier(os)	64	.7008 ± .0084	.6988 ± .0089	.6977 ± .0092	.6992 ± .0087
	128	.7742 ± .0084	.7738 ± .0086	.7736 ± .0086	.7738 ± .0085
	256	.8282 ± .0088	.8270 ± .0096	.8267 ± .0098	.8273 ± .0094
	512	.8574 ± .0019	.8573 ± .0017	.8572 ± .0017	.8573 ± .0018
	562	.8578 ± .0020	.8577 ± .0020	.8576 ± .0020	.8577 ± .0020

Table A.9.: Mixed project: Fine-tuning on PROMISE FQ with different amounts of (training) examples (classification: IsFunctional)

Approach	# Examples	IsQuality			
		P	R	F1	F2
norbert-base	64	.8091 ± .0143	.8053 ± .0146	.8070 ± .0144	.8061 ± .0145
	128	.8525 ± .0058	.8602 ± .0053	.8557 ± .0057	.8587 ± .0054
	256	.9024 ± .0085	.9035 ± .0080	.9029 ± .0083	.9032 ± .0081
	512	.9218 ± .0055	.9169 ± .0050	.9192 ± .0051	.9179 ± .0050
	562	.9256 ± .0023	.9197 ± .0022	.9224 ± .0022	.9209 ± .0022
norbert-base(os)	64	.8196 ± .0175	.8074 ± .0187	.8123 ± .0184	.8098 ± .0184
	128	.8673 ± .0129	.8679 ± .0119	.8676 ± .0124	.8678 ± .0121
	256	.9057 ± .0037	.9027 ± .0026	.9042 ± .0031	.9033 ± .0028
	512	.9245 ± .0059	.9183 ± .0069	.9212 ± .0063	.9196 ± .0066
	562	.9234 ± .0064	.9168 ± .0057	.9199 ± .0060	.9181 ± .0058
norbert-large	64	.7936 ± .0060	.7520 ± .0097	.7615 ± .0084	.7599 ± .0070
	128	.8784 ± .0120	.8706 ± .0165	.8737 ± .0141	.8722 ± .0154
	256	.9092 ± .0053	.9069 ± .0046	.9080 ± .0049	.9074 ± .0048
	512	.9201 ± .0051	.9161 ± .0047	.9180 ± .0049	.9169 ± .0048
	562	.9220 ± .0054	.9204 ± .0053	.9212 ± .0052	.9207 ± .0053
norbert-large(os)	64	.8228 ± .0083	.8099 ± .0150	.8147 ± .0124	.8125 ± .0135
	128	.8689 ± .0145	.8588 ± .0191	.8630 ± .0175	.8608 ± .0182
	256	.9073 ± .0103	.9046 ± .0086	.9059 ± .0093	.9051 ± .0089
	512	.9230 ± .0098	.9183 ± .0105	.9204 ± .0101	.9192 ± .0103
	562	.9228 ± .0062	.9158 ± .0062	.9190 ± .0062	.9172 ± .0062
eltahier	64	.7895 ± .0193	.7412 ± .0173	.7513 ± .0184	.7504 ± .0176
	128	.8530 ± .0030	.8403 ± .0029	.8455 ± .0026	.8428 ± .0027
	256	.9088 ± .0086	.9038 ± .0049	.9060 ± .0064	.9048 ± .0056
	512	.9211 ± .0020	.9168 ± .0016	.9188 ± .0011	.9176 ± .0013
	562	.9221 ± .0056	.9174 ± .0044	.9196 ± .0049	.9184 ± .0046
eltahier(os)	64	.7953 ± .0190	.7630 ± .0200	.7720 ± .0203	.7692 ± .0198
	128	.8690 ± .0077	.8417 ± .0048	.8512 ± .0055	.8470 ± .0053
	256	.9036 ± .0079	.8934 ± .0044	.8978 ± .0057	.8954 ± .0050
	512	.9265 ± .0033	.9181 ± .0010	.9218 ± .0019	.9197 ± .0013
	562	.9244 ± .0044	.9175 ± .0045	.9207 ± .0044	.9189 ± .0044

Table A.10.: Mixed project: Fine-tuning on PROMISE FQ with different amounts of (training) examples (classification: IsQuality)

Approach	# Examples	IsOnlyFunctional			
		P	R	F1	F2
norbert-base	64	.7615 ± .0091	.7570 ± .0107	.7588 ± .0076	.7578 ± .0091
	128	.8023 ± .0143	.8096 ± .0174	.8055 ± .0144	.8081 ± .0159
	256	.8425 ± .0161	.8442 ± .0209	.8433 ± .0183	.8438 ± .0199
	512	.8747 ± .0218	.8603 ± .0268	.8671 ± .0244	.8632 ± .0258
	562	.8826 ± .0153	.8675 ± .0200	.8747 ± .0177	.8705 ± .0190
norbert-base(os)	64	.7682 ± .0114	.7506 ± .0125	.7583 ± .0102	.7540 ± .0111
	128	.8182 ± .0180	.8129 ± .0233	.8151 ± .0196	.8139 ± .0216
	256	.8503 ± .0208	.8432 ± .0269	.8465 ± .0238	.8446 ± .0256
	512	.8846 ± .0169	.8705 ± .0166	.8772 ± .0167	.8733 ± .0167
	562	.8885 ± .0181	.8702 ± .0234	.8788 ± .0209	.8738 ± .0223
norbert-large	64	.7457 ± .0296	.7012 ± .0340	.7178 ± .0334	.7096 ± .0330
	128	.8380 ± .0114	.8231 ± .0261	.8297 ± .0190	.8260 ± .0231
	256	.8505 ± .0057	.8516 ± .0145	.8509 ± .0095	.8514 ± .0125
	512	.8763 ± .0182	.8601 ± .0219	.8677 ± .0199	.8633 ± .0210
	562	.8786 ± .0145	.8678 ± .0143	.8729 ± .0139	.8699 ± .0140
norbert-large(os)	64	.7574 ± .0328	.7404 ± .0383	.7477 ± .0355	.7437 ± .0370
	128	.8256 ± .0129	.8079 ± .0212	.8158 ± .0171	.8114 ± .0192
	256	.8422 ± .0117	.8500 ± .0153	.8460 ± .0134	.8484 ± .0146
	512	.8793 ± .0162	.8639 ± .0261	.8708 ± .0207	.8668 ± .0235
	562	.8883 ± .0045	.8636 ± .0101	.8750 ± .0066	.8684 ± .0085
eltahier	64	.7531 ± .0171	.6965 ± .0162	.7166 ± .0173	.7071 ± .0162
	128	.7909 ± .0180	.7747 ± .0148	.7817 ± .0142	.7778 ± .0140
	256	.8389 ± .0097	.8389 ± .0143	.8389 ± .0119	.8389 ± .0133
	512	.8690 ± .0143	.8612 ± .0217	.8648 ± .0179	.8628 ± .0200
	562	.8688 ± .0171	.8630 ± .0201	.8658 ± .0186	.8642 ± .0195
eltahier(os)	64	.7449 ± .0103	.7117 ± .0174	.7250 ± .0153	.7181 ± .0159
	128	.8060 ± .0205	.7738 ± .0219	.7877 ± .0209	.7800 ± .0212
	256	.8376 ± .0126	.8340 ± .0186	.8357 ± .0155	.8347 ± .0173
	512	.8785 ± .0163	.8664 ± .0183	.8721 ± .0166	.8688 ± .0174
	562	.8740 ± .0189	.8656 ± .0279	.8695 ± .0235	.8672 ± .0261

Table A.11.: Mixed project: Fine-tuning on PROMISE FQ with different amounts of (training) examples (classification: IsOnlyFunctional)

Approach	# Examples	IsOnlyQuality			
		P	R	F1	F2
norbert-base	64	.7192 ± .0120	.7130 ± .0158	.7156 ± .0143	.7142 ± .0150
	128	.8120 ± .0139	.7999 ± .0151	.8052 ± .0146	.8023 ± .0149
	256	.8486 ± .0228	.8469 ± .0245	.8476 ± .0235	.8472 ± .0241
	512	.8638 ± .0180	.8647 ± .0165	.8642 ± .0173	.8645 ± .0168
	562	.8701 ± .0177	.8711 ± .0149	.8705 ± .0162	.8709 ± .0154
norbert-base(os)	64	.7170 ± .0196	.7142 ± .0214	.7155 ± .0206	.7148 ± .0210
	128	.8116 ± .0111	.8060 ± .0098	.8086 ± .0103	.8071 ± .0100
	256	.8486 ± .0198	.8467 ± .0222	.8476 ± .0211	.8471 ± .0217
	512	.8655 ± .0144	.8691 ± .0101	.8671 ± .0121	.8684 ± .0108
	562	.8625 ± .0117	.8646 ± .0089	.8635 ± .0104	.8642 ± .0095
norbert-large	64	.7240 ± .0223	.7324 ± .0236	.7273 ± .0228	.7307 ± .0233
	128	.8158 ± .0151	.8206 ± .0136	.8180 ± .0144	.8196 ± .0139
	256	.8513 ± .0151	.8419 ± .0138	.8461 ± .0141	.8438 ± .0138
	512	.8723 ± .0141	.8762 ± .0138	.8741 ± .0137	.8754 ± .0137
	562	.8739 ± .0143	.8750 ± .0144	.8744 ± .0143	.8748 ± .0144
norbert-large(os)	64	.7459 ± .0166	.7431 ± .0186	.7443 ± .0174	.7437 ± .0181
	128	.8103 ± .0047	.8167 ± .0061	.8132 ± .0052	.8154 ± .0058
	256	.8560 ± .0066	.8500 ± .0072	.8528 ± .0062	.8512 ± .0066
	512	.8802 ± .0057	.8842 ± .0068	.8821 ± .0058	.8834 ± .0064
	562	.8735 ± .0076	.8777 ± .0120	.8754 ± .0095	.8769 ± .0111
eltahier	64	.6960 ± .0255	.6972 ± .0270	.6965 ± .0262	.6970 ± .0267
	128	.7851 ± .0138	.7846 ± .0124	.7848 ± .0131	.7847 ± .0127
	256	.8470 ± .0232	.8410 ± .0259	.8438 ± .0246	.8422 ± .0253
	512	.8638 ± .0095	.8603 ± .0096	.8619 ± .0093	.8610 ± .0094
	562	.8677 ± .0171	.8648 ± .0155	.8661 ± .0161	.8653 ± .0157
eltahier(os)	64	.6989 ± .0219	.6974 ± .0231	.6981 ± .0226	.6977 ± .0229
	128	.7845 ± .0181	.7896 ± .0185	.7868 ± .0183	.7885 ± .0184
	256	.8412 ± .0233	.8359 ± .0280	.8383 ± .0258	.8369 ± .0271
	512	.8604 ± .0081	.8594 ± .0051	.8597 ± .0060	.8596 ± .0053
	562	.8645 ± .0111	.8630 ± .0089	.8637 ± .0099	.8633 ± .0093

Table A.12.: Mixed project: Fine-tuning on PROMISE FQ with different amounts of (training) examples (classification: IsOnlyQuality)

A.2. Supplementary Material for Scenario 1 - Cross project

Approach	# Examples	P	R	F1	F2
gpt-4.1	1	.8326 ± .0052	.8333 ± .0065	.8056 ± .0073	.8332 ± .0062
	2	.8463 ± .0052	.8525 ± .0069	.8286 ± .0083	.8513 ± .0065
	4	.8504 ± .0042	.8576 ± .0052	.8344 ± .0060	.8561 ± .0049
	8	.8498 ± .0037	.8573 ± .0046	.8346 ± .0055	.8558 ± .0044

Table A.13.: Cross project: Prompting on PROMISE NFR with Dalpiaz p-fold and different amounts of (training) examples

Approach	# Examples	P	R	F1	F2
gpt-4.1	1	.8306 ± .0054	.8304 ± .0070	.8022 ± .0079	.8304 ± .0067
	2	.8500 ± .0023	.8577 ± .0027	.8350 ± .0031	.8562 ± .0026
	4	.8495 ± .0023	.8570 ± .0024	.8342 ± .0025	.8555 ± .0024
	8	.8552 ± .0017	.8645 ± .0025	.8433 ± .0035	.8626 ± .00234

Table A.14.: Cross project: Prompting on PROMISE NFR with custom p-fold and different amounts of (training) examples

Approach	# Examples	P	R	F1	F2
norbert-base	500	.8157 ± .0033	.7504 ± .0057	.7596 ± .0061	.7496 ± .0061
norbert-base(os)	500	.8235 ± .0014	.7581 ± .0023	.7679 ± .0025	.7576 ± .0025
norbert-large	500	.8157 ± .0023	.7483 ± .0031	.7574 ± .0034	.7473 ± .0033
norbert-large(os)	500	.8112 ± .0079	.7496 ± .0146	.7584 ± .0156	.7489 ± .0157
eltahier	500	.8071 ± .0024	.7475 ± .0039	.7563 ± .0041	.7469 ± .0042
eltahier(os)	500	.8005 ± .0022	.7286 ± .0049	.7359 ± .0054	.7262 ± .0054

Table A.15.: Cross project: Fine-tuning on PROMISE NFR with Dalpiaz p-fold

Approach	# Examples	P	R	F1	F2
norbert-base	500	.8157 ± .0033	.7504 ± .0057	.7596 ± .0061	.7496 ± .0061
norbert-base(os)	500	.8235 ± .0014	.7581 ± .0023	.7679 ± .0025	.7576 ± .0025
norbert-large	500	.8157 ± .0023	.7483 ± .0031	.7574 ± .0034	.7473 ± .0033
norbert-large(os)	500	.8112 ± .0079	.7496 ± .0146	.7584 ± .0156	.7489 ± .0157
eltahier	500	.8071 ± .0024	.7475 ± .0039	.7563 ± .0041	.7469 ± .0042
eltahier(os)	500	.8005 ± .0022	.7286 ± .0049	.7359 ± .0054	.7262 ± .0054

Table A.16.: Cross project: Fine-tuning on PROMISE NFR with custom p-fold

Approach	# Examples	P	R	F1	F2
gpt-4.1	1	.8593 ± .0033	.8747 ± .0034	.8650 ± .0033	.8716 ± .0033
	2	.8690 ± .0025	.8849 ± .0040	.8747 ± .0027	.8816 ± .0035
	4	.8621 ± .0092	.8806 ± .0091	.8682 ± .0093	.8768 ± .0091
	8	.8722 ± .0049	.8917 ± .0046	.8785 ± .0050	.8877 ± .0047

Table A.17.: Cross project: Prompting on SecReq with p-fold

Approach	# Examples	P	R	F1	F2
norbert-base	340	.6429 ± .0398	.6503 ± .0386	.6394 ± .0447	.6441 ± .0424
norbert-base(os)	340	.6445 ± .0227	.6549 ± .0250	.6342 ± .0289	.6426 ± .0279
norbert-large	340	.7534 ± .0363	.7624 ± .0315	.7545 ± .0379	.7583 ± .0351
norbert-large(os)	340	.7162 ± .0326	.7269 ± .0292	.7166 ± .0348	.7215 ± .0324
eltahier	340	.5740 ± .0138	.5792 ± .0146	.5694 ± .0165	.5735 ± .0160
eltahier(os)	340	.5554 ± .0263	.5595 ± .0284	.5438 ± .0323	.5497 ± .0313

Table A.18.: Cross project: Fine-tuning on SecReq with p-fold

Approach	# Examples	IsFunctional			
		P	R	F1	F2
gpt-4.1	1	.8348 ± .0023	.8191 ± .0033	.8162 ± .0036	.8222 ± .0031
	2	.8430 ± .0010	.8310 ± .0024	.8288 ± .0026	.8333 ± .0021
	4	.8499 ± .0042	.8386 ± .0059	.8366 ± .0062	.8408 ± .0056

Table A.19.: Cross project: Prompting on PROMISE FQ with Dalpiaz p-fold and different amounts of (training) examples (classification: IsFunctional)

Approach	# Examples	IsQuality			
		P	R	F1	F2
gpt-4.1	1	.8248 ± .0013	.8341 ± .0016	.8007 ± .0023	.8323 ± .0015
	2	.8473 ± .0051	.8637 ± .0060	.8374 ± .0072	.8603 ± .0058
	4	.8363 ± .0031	.8501 ± .0041	.8210 ± .0051	.8473 ± .0039

Table A.20.: Cross project: Prompting on PROMISE FQ with Dalpiaz p-fold and different amounts of (training) examples (classification: IsQuality)

Approach	# Examples	IsOnlyFunctional			
		P	R	F1	F2
gpt-4.1	1	.8396 ± .0012	.8644 ± .0014	.8315 ± .0015	.8593 ± .0013
	2	.8556 ± .0033	.8821 ± .0034	.8546 ± .0042	.8766 ± .0034
	4	.8516 ± .0036	.8779 ± .0039	.8491 ± .0050	.8725 ± .0038

Table A.21.: Cross project: Prompting on PROMISE FQ with Dalpiaz p-fold and different amounts of (training) examples (classification: IsOnlyFunctional)

Approach	# Examples	IsOnlyQuality			
		P	R	F1	F2
gpt-4.1	1	.8195 ± .0025	.7861 ± .0049	.7842 ± .0053	.7925 ± .0044
	2	.8325 ± .0035	.8091 ± .0042	.8089 ± .0044	.8136 ± .0040
	4	.8330 ± .0036	.8068 ± .0038	.8064 ± .0039	.8119 ± .0037

Table A.22.: Cross project: Prompting on PROMISE FQ with Dalpiaz p-fold and different amounts of (training) examples (classification: IsOnlyQuality)

Approach	# Examples	IsFunctional			
		P	R	F1	F2
gpt-4.1	1	.8333 ± .0023	.8146 ± .0060	.8111 ± .0067	.8182 ± .0052
	2	.8421 ± .0032	.8292 ± .0034	.8269 ± .0035	.8318 ± .0033
	4	.8513 ± .0028	.8403 ± .0026	.8385 ± .0028	.8425 ± .0026

Table A.23.: Cross project: Prompting on PROMISE FQ with custom p-fold and different amounts of (training) examples (classification: IsFunctional)

Approach	# Examples	IsQuality			
		P	R	F1	F2
gpt-4.1	1	.8244 ± .0067	.8331 ± .0105	.7996 ± .0142	.8313 ± .0098
	2	.8495 ± .0046	.8664 ± .0057	.8407 ± .0072	.8630 ± .0054
	4	.8368 ± .0041	.8514 ± .0048	.8233 ± .0053	.8485 ± .0046

Table A.24.: Cross project: Prompting on PROMISE FQ with custom p-fold and different amounts of (training) examples (classification: IsQuality)

Approach	# Examples	IsOnlyFunctional			
		P	R	F1	F2
gpt-4.1	1	.8377 ± .0068	.8619 ± .0080	.8286 ± .0110	.8569 ± .0077
	2	.8555 ± .0061	.8819 ± .0065	.8544 ± .0082	.8765 ± .0064
	4	.8516 ± .0056	.8778 ± .0060	.8498 ± .0069	.8724 ± .0059

Table A.25.: Cross project: Prompting on PROMISE FQ with custom p-fold and different amounts of (training) examples (classification: IsOnlyFunctional)

Approach	# Examples	IsOnlyQuality			
		P	R	F1	F2
gpt-4.1	1	.8169 ± .0035	.7804 ± .0109	.7779 ± .0121	.7874 ± .0095
	2	.8374 ± .0018	.8137 ± .0035	.8136 ± .0037	.8183 ± .0032
	4	.8364 ± .0031	.8111 ± .0052	.8109 ± .0055	.8160 ± .0048

Table A.26.: Cross project: Prompting on PROMISE FQ with custom p-fold and different amounts of (training) examples (classification: IsOnlyQuality)

Approach	# Examples	IsFunctional			
		P	R	F1	F2
norbert-base	500	.8141 ± .0016	.8123 ± .0018	.8123 ± .0019	.8121 ± .0019
norbert-base(os)	500	.8113 ± .0018	.8093 ± .0015	.8092 ± .0014	.8090 ± .0014
norbert-large	500	.8186 ± .0020	.8163 ± .0022	.8162 ± .0022	.8159 ± .0022
norbert-large(os)	500	.8208 ± .0029	.8194 ± .0029	.8193 ± .0029	.8192 ± .0029
eltahier	500	.8158 ± .0007	.8158 ± .0007	.8158 ± .0007	.8158 ± .0007
eltahier(os)	500	.8125 ± .0049	.8123 ± .0050	.8123 ± .0050	.8123 ± .0050

Table A.27.: Cross project: Fine-tuning on PROMISE FQ with Dalpiaz p-fold (classification: IsFunctional)

Approach	# Examples	IsQuality			
		P	R	F1	F2
norbert-base	500	.8271 ± .0047	.8000 ± .0070	.8089 ± .0066	.8025 ± .0070
norbert-base(os)	500	.8492 ± .0077	.8187 ± .0117	.8286 ± .0109	.8215 ± .0116
norbert-large	500	.8596 ± .0005	.8385 ± .0010	.8463 ± .0008	.8410 ± .0010
norbert-large(os)	500	.8641 ± .0082	.8318 ± .0157	.8422 ± .0142	.8348 ± .0156
eltahier	500	.8330 ± .0024	.7994 ± .0043	.8096 ± .0040	.8021 ± .0043
eltahier(os)	500	.8293 ± .0057	.7894 ± .0087	.8005 ± .0084	.7919 ± .0088

Table A.28.: Cross project: Fine-tuning on PROMISE FQ with Dalpiaz p-fold (classification: IsQuality)

Approach	# Examples	IsOnlyFunctional			
		P	R	F1	F2
norbert-base	500	.8278 ± .0028	.7939 ± .0053	.8052 ± .0048	.7973 ± .0052
norbert-base(os)	500	.8417 ± .0071	.8073 ± .0116	.8189 ± .0107	.8108 ± .0115
norbert-large	500	.8500 ± .0002	.8231 ± .0013	.8331 ± .0010	.8264 ± .0012
norbert-large(os)	500	.8496 ± .0082	.8146 ± .0175	.8262 ± .0156	.8180 ± .0173
eltahier	500	.8376 ± .0006	.7959 ± .0019	.8090 ± .0016	.7995 ± .0019
eltahier(os)	500	.8242 ± .0056	.7773 ± .0084	.7907 ± .0082	.7806 ± .0086

Table A.29.: Cross project: Fine-tuning on PROMISE FQ with Dalpiaz p-fold (classification: IsOnlyFunctional)

Approach	# Examples	IsOnlyQuality			
		P	R	F1	F2
norbert-base	500	.8159 ± .0030	.8160 ± .0031	.8152 ± .0032	.8155 ± .0032
norbert-base(os)	500	.8202 ± .0017	.8200 ± .0016	.8190 ± .0015	.8193 ± .0015
norbert-large	500	.8320 ± .0014	.8320 ± .0015	.8310 ± .0015	.8313 ± .0015
norbert-large(os)	500	.8377 ± .0029	.8378 ± .0027	.8370 ± .0026	.8373 ± .0026
eltahier	500	.8170 ± .0024	.8169 ± .0023	.8170 ± .0023	.8169 ± .0023
eltahier(os)	500	.8220 ± .0066	.8222 ± .0065	.8221 ± .0066	.8221 ± .0065

Table A.30.: Cross project: Fine-tuning on PROMISE FQ with Dalpiaz p-fold (classification: IsOnlyQuality)

Approach	# Examples	IsFunctional			
		P	R	F1	F2
norbert-base	500	.8171 ± .0015	.8145 ± .0017	.8143 ± .0017	.8141 ± .0017
norbert-base(os)	500	.8256 ± .0067	.8231 ± .0073	.8230 ± .0074	.8227 ± .0075
norbert-large	500	.8220 ± .0032	.8195 ± .0035	.8194 ± .0036	.8191 ± .0036
norbert-large(os)	500	.8228 ± .0015	.8186 ± .0031	.8183 ± .0033	.8179 ± .0035
eltahier	500	.8145 ± .0044	.8143 ± .0045	.8144 ± .0045	.8143 ± .0046
eltahier(os)	500	.8194 ± .0033	.8184 ± .0033	.8184 ± .0033	.8183 ± .0033

Table A.31.: Cross project: Fine-tuning on PROMISE FQ with custom p-fold (classification: IsFunctional)

Approach	# Examples	IsQuality			
		P	R	F1	F2
norbert-base	500	.8520 ± .0002	.8161 ± .0007	.8272 ± .0005	.8190 ± .0007
norbert-base(os)	500	.8586 ± .0045	.8259 ± .0036	.8365 ± .0038	.8289 ± .0037
norbert-large	500	.8709 ± .0021	.8450 ± .0034	.8543 ± .0031	.8479 ± .0033
norbert-large(os)	500	.8693 ± .0046	.8349 ± .0087	.8460 ± .0079	.8380 ± .0087
eltahier	500	.8527 ± .0029	.8237 ± .0019	.8334 ± .0021	.8265 ± .0019
eltahier(os)	500	.8525 ± .0058	.8148 ± .0074	.8262 ± .0069	.8178 ± .0075

Table A.32.: Cross project: Fine-tuning on PROMISE FQ with custom p-fold (classification: IsQuality)

Approach	# Examples	IsOnlyFunctional			
		P	R	F1	F2
norbert-base	500	.8437 ± .0011	.8065 ± .0010	.8189 ± .0010	.8101 ± .0010
norbert-base(os)	500	.8538 ± .0023	.8168 ± .0022	.8293 ± .0013	.8205 ± .0020
norbert-large	500	.8570 ± .0018	.8279 ± .0023	.8386 ± .0022	.8314 ± .0023
norbert-large(os)	500	.8559 ± .0069	.8144 ± .0113	.8279 ± .0106	.8183 ± .0113
eltahier	500	.8459 ± .0016	.8145 ± .0011	.8256 ± .0013	.8179 ± .0012
eltahier(os)	500	.8493 ± .0043	.8067 ± .0083	.8202 ± .0075	.8105 ± .0083

Table A.33.: Cross project: Fine-tuning on PROMISE NFR with custom p-fold (classification: IsOnlyFunctional)

Approach	# Examples	IsOnlyQuality			
		P	R	F1	F2
norbert-base	500	.8237 ± .0014	.8228 ± .0014	.8214 ± .0014	.8218 ± .0014
norbert-base(os)	500	.8332 ± .0034	.8328 ± .0038	.8316 ± .0040	.8320 ± .0039
norbert-large	500	.8370 ± .0032	.8366 ± .0033	.8354 ± .0034	.8358 ± .0034
norbert-large(os)	500	.8404 ± .0013	.8394 ± .0022	.8379 ± .0026	.8383 ± .0026
eltahier	500	.8217 ± .0037	.8219 ± .0036	.8217 ± .0038	.8218 ± .0037
eltahier(os)	500	.8247 ± .0018	.8250 ± .0018	.8244 ± .0017	.8246 ± .0017

Table A.34.: Cross project: Fine-tuning on PROMISE FQ with custom p-fold (classification: IsOnlyQuality)

A.3. Supplementary Material for Scenario 2 - Intra project

Approach	# Examples	P	R	F1	F2
gpt-4.1	1	0.8453	0.8539	0.8325	0.8522
	2	0.8651	0.876	0.8563	0.8738
	4	0.8677	0.8795	0.861	0.8771

Table A.35.: Intra project: Prompting on PROMISE NFR with Dalpiaz p-fold and different amounts of (training) examples

Approach	# Examples	P	R	F1	F2
gpt-4.1	1	0.8388	0.8429	0.8175	0.8421
	2	0.854	0.8629	0.8413	0.8611
	4	0.8652	0.8764	0.8571	0.87411

Table A.36.: Intra project: Prompting on PROMISE NFR with custom p-fold and different amounts of (training) examples

Approach	# Examples	P	R	F1	F2
gpt-4.1	1	0.8365	0.8538	0.8421	0.8503
	2	0.8583	0.8753	0.8643	0.8718
	4	0.8627	0.8818	0.8688	0.8779

Table A.37.: Intra project: Prompting on SecReq with p-fold and different amounts of (training) examples

Approach	# Examples	IsFunctional				IsQuality			
		P	R	F1	F2	P	R	F1	F2
gpt-4.1	1	0.8471	0.8328	0.8304	0.8356	0.8237	0.8318	0.7974	0.8302
	2	0.8479	0.8383	0.8366	0.8402	0.8327	0.8466	0.8177	0.8438
	4	0.8538	0.847	0.8457	0.8484	0.8507	0.8682	0.8436	0.8646

Table A.38.: Intra project: Prompting on PROMISE FQ with Dalpiaz p-fold and different amounts of (training) examples (classifications: IsFunctional, IsQuality)

Approach	# Examples	IsOnlyFunctional				IsOnlyQuality			
		P	R	F1	F2	P	R	F1	F2
gpt-4.1	1	0.7922	0.8707	0.8061	0.8538	0.847	0.7569	0.7743	0.7733
	2	0.8844	0.9433	0.9065	0.9309	0.9204	0.8618	0.8803	0.8729
	4	0.8228	0.8882	0.8429	0.8743	0.8547	0.7923	0.8089	0.804

Table A.39.: Intra project: Prompting on PROMISE FQ with Dalpiaz p-fold and different amounts of (training) examples (classifications: IsOnlyFunctional, IsOnlyQuality)

Approach	# Examples	IsFunctional				IsQuality			
		P	R	F1	F2	P	R	F1	F2
gpt-4.1	1	0.8469	0.8344	0.8322	0.8369	0.8301	0.8433	0.8138	0.8406
	2	0.8432	0.8312	0.829	0.8336	0.8501	0.8675	0.8428	0.864
	4	0.8607	0.8542	0.853	0.8555	0.8503	0.8678	0.8435	0.8642

Table A.40.: Intra project: Prompting on PROMISE FQ with custom p-fold and different amounts of (training) examples (classifications: IsFunctional, IsQuality)

Approach	# Examples	IsOnlyFunctional				IsOnlyQuality			
		P	R	F1	F2	P	R	F1	F2
gpt-4.1	1	0.8126	0.8823	0.829	0.8674	0.8514	0.7723	0.7885	0.7869
	2	0.8189	0.8858	0.8366	0.8715	0.855	0.7902	0.8059	0.8023
	4	0.829	0.8924	0.8478	0.879	0.8587	0.8045	0.8193	0.8147

Table A.41.: Intra project: Prompting on PROMISE FQ with custom p-fold and different amounts of (training) examples (classifications: IsOnlyFunctional, IsOnlyQuality)

A.4. Supplementary Material for Scenario 3 - Intra cross project

Approach	# Examples	P	R	F1	F2
norbert-base	20	.8703 ± .0090	.8647 ± .0089	.8672 ± .0089	.8656 ± .0089
	40	.8971 ± .0071	.8919 ± .0045	.8942 ± .0055	.8928 ± .0048
	60	.9066 ± .0028	.9033 ± .0034	.9048 ± .0029	.9039 ± .0032
	80	.9157 ± .0023	.9112 ± .0046	.9132 ± .0035	.9120 ± .0042
norbert-base(os)	20	.8736 ± .0107	.8641 ± .0112	.8680 ± .0110	.8655 ± .0112
	40	.8927 ± .0083	.8875 ± .0078	.8898 ± .0079	.8883 ± .0078
	60	.9066 ± .0048	.9023 ± .0049	.9043 ± .0048	.9030 ± .0049
	80	.9159 ± .0045	.9092 ± .0032	.9122 ± .0037	.9103 ± .0034
norbert-large	20	.8673 ± .0064	.8572 ± .0075	.8612 ± .0070	.8585 ± .0074
	40	.8994 ± .0077	.8978 ± .0084	.8986 ± .0081	.8981 ± .0083
	60	.9087 ± .0014	.9083 ± .0022	.9085 ± .0015	.9083 ± .0019
	80	.9163 ± .0014	.9137 ± .0037	.9149 ± .0025	.9141 ± .0032
norbert-large(os)	20	.8741 ± .0054	.8657 ± .0079	.8690 ± .0065	.8668 ± .0074
	40	.8998 ± .0060	.8982 ± .0062	.8990 ± .0061	.8985 ± .0062
	60	.9033 ± .0040	.8999 ± .0041	.9015 ± .0040	.9005 ± .0040
	80	.9179 ± .0046	.9135 ± .0052	.9155 ± .0049	.9143 ± .0050
eltahier	20	.8669 ± .0090	.8620 ± .0051	.8640 ± .0067	.8627 ± .0056
	40	.8955 ± .0064	.8915 ± .0052	.8933 ± .0057	.8922 ± .0053
	60	.9007 ± .0033	.8972 ± .0049	.8988 ± .0041	.8978 ± .0046
	80	.9112 ± .0039	.9074 ± .0037	.9091 ± .0038	.9080 ± .0038
eltahier(os)	20	.8635 ± .0103	.8520 ± .0104	.8565 ± .0104	.8535 ± .0104
	40	.8920 ± .0058	.8849 ± .0036	.8879 ± .0043	.8860 ± .0038
	60	.9028 ± .0015	.8965 ± .0026	.8993 ± .0019	.8975 ± .0024
	80	.9121 ± .0041	.9051 ± .0039	.9082 ± .0038	.9062 ± .0038

Table A.42.: Intra cross project: Further fine-tuning on PROMISE NFR with Dalpiaz p-fold and different amounts of internal (training) examples

Approach	# Examples	P	R	F1	F2
norbert-base	20	.8825 ± .0053	.8701 ± .0058	.8750 ± .0055	.8717 ± .0057
	40	.8987 ± .0059	.8919 ± .0068	.8948 ± .0064	.8929 ± .0067
	60	.9126 ± .0039	.9067 ± .0044	.9093 ± .0041	.9077 ± .0043
	80	.9189 ± .0030	.9139 ± .0040	.9161 ± .0034	.9147 ± .0038
	100	.9250 ± .0024	.9189 ± .0031	.9216 ± .0028	.9199 ± .0030
norbert-base(os)	20	.8847 ± .0046	.8695 ± .0090	.8752 ± .0076	.8713 ± .0087
	40	.9020 ± .0037	.8927 ± .0057	.8966 ± .0050	.8941 ± .0055
	60	.9114 ± .0048	.9029 ± .0054	.9066 ± .0051	.9042 ± .0053
	80	.9205 ± .0027	.9142 ± .0032	.9170 ± .0030	.9152 ± .0031
	100	.9223 ± .0033	.9175 ± .0030	.9197 ± .0029	.9183 ± .0029
norbert-large	20	.8840 ± .0025	.8716 ± .0035	.8764 ± .0028	.8732 ± .0033
	40	.9013 ± .0032	.8982 ± .0026	.8996 ± .0029	.8988 ± .0027
	60	.9099 ± .0056	.9062 ± .0052	.9079 ± .0053	.9068 ± .0052
	80	.9183 ± .0035	.9157 ± .0041	.9169 ± .0037	.9162 ± .0040
	100	.9227 ± .0030	.9192 ± .0030	.9208 ± .0029	.9198 ± .0030
norbert-large(os)	20	.8776 ± .0063	.8660 ± .0051	.8707 ± .0055	.8676 ± .0052
	40	.9042 ± .0077	.9002 ± .0082	.9020 ± .0079	.9009 ± .0081
	60	.9138 ± .0078	.9085 ± .0066	.9109 ± .0070	.9094 ± .0067
	80	.9188 ± .0037	.9156 ± .0039	.9171 ± .0037	.9162 ± .0038
	100	.9246 ± .0066	.9215 ± .0060	.9229 ± .0062	.9220 ± .0061
eltahier	20	.8734 ± .0075	.8632 ± .0042	.8673 ± .0053	.8646 ± .0045
	40	.8972 ± .0027	.8892 ± .0043	.8926 ± .0035	.8904 ± .0040
	60	.9089 ± .0023	.9024 ± .0026	.9053 ± .0023	.9035 ± .0024
	80	.9140 ± .0014	.9100 ± .0013	.9118 ± .0010	.9107 ± .0011
	100	.9217 ± .0022	.9164 ± .0017	.9188 ± .0019	.9173 ± .0017
eltahier(os)	20	.8774 ± .0023	.8666 ± .0031	.8710 ± .0027	.8681 ± .0030
	40	.8942 ± .0029	.8871 ± .0032	.8902 ± .0030	.8882 ± .0032
	60	.9098 ± .0037	.9045 ± .0048	.9069 ± .0043	.9054 ± .0047
	80	.9151 ± .0049	.9105 ± .0050	.9126 ± .0050	.9113 ± .0050
	100	.9194 ± .0032	.9145 ± .0038	.9167 ± .0035	.9154 ± .0037

Table A.43.: Intra cross project: Further fine-tuning on PROMISE NFR with custom p-fold and different amounts of internal (training) examples

Approach	# Examples	P	R	F1	F2
norbert-base	20	.8091 ± .0111	.8139 ± .0147	.8102 ± .0124	.8121 ± .0135
	40	.8353 ± .0099	.8406 ± .0097	.8377 ± .0098	.8394 ± .0097
	60	.8633 ± .0117	.8732 ± .0115	.8674 ± .0116	.8706 ± .0115
	80	.8826 ± .0129	.8941 ± .0123	.8873 ± .0126	.8911 ± .0124
	100	.8936 ± .0124	.9060 ± .0126	.8988 ± .0125	.9028 ± .0126
norbert-base(os)	20	.8117 ± .0133	.8193 ± .0171	.8141 ± .0135	.8169 ± .0151
	40	.8500 ± .0248	.8529 ± .0197	.8511 ± .0225	.8521 ± .0209
	60	.8763 ± .0118	.8749 ± .0110	.8753 ± .0111	.8750 ± .0109
	80	.8868 ± .0119	.8927 ± .0142	.8895 ± .0129	.8914 ± .0136
	100	.9108 ± .0110	.9137 ± .0119	.9122 ± .0114	.9131 ± .0117
norbert-large	20	.8346 ± .0115	.8445 ± .0124	.8384 ± .0116	.8418 ± .0119
	40	.8602 ± .0112	.8738 ± .0125	.8655 ± .0116	.8701 ± .0121
	60	.8838 ± .0058	.8943 ± .0032	.8882 ± .0049	.8916 ± .0039
	80	.8960 ± .0079	.9110 ± .0090	.9018 ± .0080	.9069 ± .0084
	100	.9129 ± .0082	.9245 ± .0074	.9179 ± .0080	.9216 ± .0076
norbert-large(os)	20	.8460 ± .0066	.8512 ± .0092	.8481 ± .0072	.8498 ± .0082
	40	.8728 ± .0260	.8767 ± .0214	.8743 ± .0238	.8756 ± .0224
	60	.8924 ± .0089	.8935 ± .0068	.8926 ± .0066	.8930 ± .0064
	80	.9041 ± .0105	.9136 ± .0121	.9082 ± .0111	.9113 ± .0116
	100	.9140 ± .0076	.9232 ± .0074	.9181 ± .0075	.9210 ± .0075
eltahier	20	.7964 ± .0167	.8018 ± .0186	.7987 ± .0174	.8004 ± .0180
	40	.8163 ± .0123	.8207 ± .0117	.8182 ± .0120	.8196 ± .0118
	60	.8434 ± .0074	.8513 ± .0050	.8468 ± .0064	.8493 ± .0055
	80	.8645 ± .0063	.8752 ± .0069	.8690 ± .0065	.8725 ± .0067
	100	.8839 ± .0096	.8961 ± .0107	.8889 ± .0099	.8929 ± .0103
eltahier(os)	20	.7948 ± .0085	.8076 ± .0090	.7987 ± .0089	.8034 ± .0088
	40	.8354 ± .0169	.8432 ± .0150	.8385 ± .0161	.8411 ± .0154
	60	.8565 ± .0140	.8666 ± .0090	.8604 ± .0127	.8638 ± .0108
	80	.8830 ± .0123	.8929 ± .0125	.8871 ± .0124	.8904 ± .0124
	100	.9000 ± .0063	.9117 ± .0072	.9049 ± .0065	.9088 ± .0069

Table A.44.: Intra cross project: Further fine-tuning on SecReq with p-fold and different amounts of internal (training) examples

Approach	# Examples	IsFunctional			
		P	R	F1	F2
norbert-base	20	.8350 ± .0044	.8346 ± .0042	.8345 ± .0042	.8345 ± .0042
	40	.8445 ± .0052	.8442 ± .0052	.8442 ± .0051	.8442 ± .0051
	60	.8533 ± .0040	.8532 ± .0041	.8532 ± .0042	.8532 ± .0042
	80	.8589 ± .0004	.8588 ± .0004	.8588 ± .0004	.8588 ± .0004
norbert-base(os)	20	.8341 ± .0016	.8337 ± .0020	.8337 ± .0020	.8337 ± .0020
	40	.8446 ± .0051	.8444 ± .0052	.8444 ± .0052	.8444 ± .0052
	60	.8497 ± .0061	.8496 ± .0060	.8496 ± .0061	.8496 ± .0061
	80	.8539 ± .0034	.8537 ± .0035	.8538 ± .0035	.8537 ± .0035
norbert-large	20	.8385 ± .0072	.8380 ± .0078	.8379 ± .0077	.8379 ± .0078
	40	.8515 ± .0045	.8511 ± .0048	.8510 ± .0048	.8509 ± .0048
	60	.8608 ± .0013	.8606 ± .0015	.8606 ± .0015	.8605 ± .0015
	80	.8625 ± .0033	.8623 ± .0032	.8624 ± .0032	.8623 ± .0032
norbert-large(os)	20	.8404 ± .0088	.8401 ± .0088	.8401 ± .0088	.8401 ± .0088
	40	.8527 ± .0047	.8524 ± .0046	.8524 ± .0045	.8524 ± .0045
	60	.8596 ± .0084	.8594 ± .0083	.8594 ± .0083	.8593 ± .0083
	80	.8595 ± .0054	.8592 ± .0055	.8592 ± .0056	.8591 ± .0056
eltahier	20	.8326 ± .0031	.8325 ± .0031	.8324 ± .0031	.8324 ± .0031
	40	.8425 ± .0049	.8417 ± .0042	.8415 ± .0041	.8415 ± .0041
	60	.8450 ± .0033	.8447 ± .0031	.8446 ± .0030	.8446 ± .0030
	80	.8506 ± .0037	.8503 ± .0037	.8502 ± .0037	.8502 ± .0037
eltahier(os)	20	.8351 ± .0033	.8350 ± .0032	.8350 ± .0032	.8350 ± .0032
	40	.8402 ± .0031	.8399 ± .0029	.8398 ± .0028	.8398 ± .0028
	60	.8480 ± .0027	.8479 ± .0027	.8478 ± .0027	.8478 ± .0027
	80	.8565 ± .0040	.8564 ± .0040	.8564 ± .0040	.8564 ± .0040

Table A.45.: Intra cross project: Further fine-tuning on PROMISE FQ with Dalpiaz p-fold and different amounts of internal (training) examples (classification: IsFunctional)

Approach	# Examples	IsQuality			
		P	R	F1	F2
norbert-base	20	.8819 ± .0054	.8804 ± .0054	.8811 ± .0053	.8807 ± .0053
	40	.8991 ± .0042	.8973 ± .0039	.8982 ± .0040	.8976 ± .0039
	60	.9058 ± .0041	.9040 ± .0054	.9049 ± .0047	.9044 ± .0051
	80	.9138 ± .0009	.9097 ± .0012	.9116 ± .0010	.9105 ± .0011
norbert-base(os)	20	.8814 ± .0031	.8764 ± .0020	.8787 ± .0022	.8773 ± .0020
	40	.8982 ± .0060	.8960 ± .0054	.8970 ± .0056	.8964 ± .0054
	60	.9035 ± .0075	.9028 ± .0075	.9031 ± .0074	.9029 ± .0074
	80	.9183 ± .0049	.9179 ± .0047	.9181 ± .0048	.9180 ± .0047
norbert-large	20	.8798 ± .0028	.8722 ± .0054	.8755 ± .0040	.8734 ± .0049
	40	.8952 ± .0100	.8896 ± .0115	.8920 ± .0105	.8904 ± .0111
	60	.9038 ± .0050	.9015 ± .0048	.9025 ± .0046	.9019 ± .0047
	80	.9120 ± .0036	.9111 ± .0030	.9115 ± .0032	.9112 ± .0031
norbert-large(os)	20	.8855 ± .0095	.8773 ± .0122	.8809 ± .0111	.8786 ± .0119
	40	.9012 ± .0078	.8970 ± .0056	.8989 ± .0065	.8977 ± .0059
	60	.9066 ± .0059	.9044 ± .0061	.9055 ± .0060	.9048 ± .0061
	80	.9109 ± .0051	.9077 ± .0040	.9092 ± .0044	.9082 ± .0042
eltahier	20	.8873 ± .0094	.8866 ± .0088	.8869 ± .0091	.8867 ± .0089
	40	.9004 ± .0079	.8991 ± .0058	.8997 ± .0068	.8993 ± .0062
	60	.9070 ± .0058	.9055 ± .0066	.9062 ± .0062	.9058 ± .0065
	80	.9113 ± .0094	.9112 ± .0077	.9112 ± .0086	.9112 ± .0081
eltahier(os)	20	.8706 ± .0101	.8660 ± .0105	.8681 ± .0103	.8668 ± .0104
	40	.8946 ± .0047	.8940 ± .0051	.8943 ± .0048	.8941 ± .0050
	60	.9043 ± .0048	.9031 ± .0061	.9037 ± .0055	.9034 ± .0058
	80	.9139 ± .0029	.9120 ± .0032	.9129 ± .0028	.9124 ± .0030

Table A.46.: Intra cross project: Further fine-tuning on PROMISE FQ with Dalpiaz p-fold and different amounts of internal (training) examples (classification: IsQuality)

Approach	# Examples	IsOnlyFunctional			
		P	R	F1	F2
norbert-base	20	.8624 ± .0154	.8517 ± .0089	.8567 ± .0117	.8537 ± .0099
	40	.8746 ± .0125	.8717 ± .0054	.8728 ± .0073	.8720 ± .0054
	60	.8872 ± .0032	.8808 ± .0056	.8839 ± .0037	.8820 ± .0047
	80	.8941 ± .0082	.8833 ± .0067	.8884 ± .0071	.8853 ± .0068
norbert-base(os)	20	.8645 ± .0081	.8452 ± .0025	.8541 ± .0046	.8486 ± .0032
	40	.8733 ± .0080	.8656 ± .0067	.8691 ± .0054	.8669 ± .0058
	60	.8859 ± .0075	.8755 ± .0067	.8804 ± .0059	.8774 ± .0062
	80	.8944 ± .0056	.8873 ± .0028	.8908 ± .0038	.8887 ± .0030
norbert-large	20	.8735 ± .0104	.8536 ± .0092	.8625 ± .0076	.8569 ± .0084
	40	.8833 ± .0189	.8700 ± .0107	.8756 ± .0111	.8720 ± .0099
	60	.8944 ± .0102	.8816 ± .0082	.8874 ± .0065	.8838 ± .0071
	80	.8938 ± .0114	.8859 ± .0037	.8896 ± .0071	.8873 ± .0049
norbert-large(os)	20	.8767 ± .0054	.8500 ± .0107	.8618 ± .0077	.8544 ± .0096
	40	.8855 ± .0136	.8780 ± .0031	.8814 ± .0079	.8793 ± .0049
	60	.8916 ± .0112	.8808 ± .0099	.8859 ± .0097	.8828 ± .0096
	80	.8952 ± .0092	.8880 ± .0025	.8913 ± .0041	.8893 ± .0022
eltahier	20	.8636 ± .0119	.8549 ± .0052	.8590 ± .0083	.8565 ± .0064
	40	.8767 ± .0097	.8721 ± .0075	.8741 ± .0060	.8728 ± .0063
	60	.8811 ± .0074	.8789 ± .0037	.8798 ± .0044	.8792 ± .0035
	80	.8857 ± .0053	.8826 ± .0019	.8841 ± .0033	.8832 ± .0023
eltahier(os)	20	.8575 ± .0133	.8409 ± .0081	.8485 ± .0101	.8438 ± .0088
	40	.8708 ± .0048	.8642 ± .0070	.8672 ± .0037	.8654 ± .0054
	60	.8807 ± .0030	.8768 ± .0057	.8786 ± .0017	.8775 ± .0041
	80	.8921 ± .0075	.8847 ± .0016	.8882 ± .0031	.8860 ± .0012

Table A.47.: Intra cross project: Further fine-tuning on PROMISE FQ with Dalpiaz p-fold and different amounts of internal (training) examples (classification: IsOnlyFunctional)

Approach	# Examples	IsOnlyQuality			
		P	R	F1	F2
norbert-base	20	.8390 ± .0069	.8378 ± .0054	.8384 ± .0060	.8380 ± .0056
	40	.8538 ± .0024	.8523 ± .0017	.8530 ± .0019	.8526 ± .0018
	60	.8613 ± .0033	.8588 ± .0042	.8600 ± .0036	.8593 ± .0039
	80	.8677 ± .0028	.8660 ± .0048	.8668 ± .0036	.8663 ± .0043
norbert-base(os)	20	.8378 ± .0067	.8373 ± .0055	.8375 ± .0059	.8374 ± .0056
	40	.8515 ± .0052	.8500 ± .0040	.8507 ± .0046	.8502 ± .0042
	60	.8561 ± .0068	.8531 ± .0096	.8544 ± .0081	.8536 ± .0090
	80	.8652 ± .0072	.8653 ± .0080	.8652 ± .0075	.8653 ± .0077
norbert-large	20	.8370 ± .0078	.8351 ± .0049	.8359 ± .0062	.8354 ± .0054
	40	.8556 ± .0074	.8517 ± .0072	.8534 ± .0068	.8523 ± .0069
	60	.8608 ± .0047	.8595 ± .0056	.8601 ± .0049	.8597 ± .0053
	80	.8674 ± .0070	.8673 ± .0084	.8673 ± .0076	.8673 ± .0081
norbert-large(os)	20	.8388 ± .0143	.8388 ± .0143	.8388 ± .0143	.8388 ± .0143
	40	.8554 ± .0066	.8546 ± .0044	.8549 ± .0053	.8547 ± .0047
	60	.8643 ± .0031	.8632 ± .0047	.8637 ± .0039	.8634 ± .0044
	80	.8627 ± .0050	.8630 ± .0069	.8627 ± .0056	.8629 ± .0063
eltahier	20	.8443 ± .0021	.8369 ± .0020	.8402 ± .0020	.8381 ± .0020
	40	.8541 ± .0020	.8459 ± .0051	.8495 ± .0032	.8472 ± .0044
	60	.8593 ± .0045	.8538 ± .0042	.8563 ± .0042	.8547 ± .0042
	80	.8668 ± .0025	.8585 ± .0040	.8622 ± .0033	.8599 ± .0038
eltahier(os)	20	.8312 ± .0092	.8293 ± .0073	.8302 ± .0082	.8296 ± .0076
	40	.8517 ± .0073	.8453 ± .0069	.8482 ± .0070	.8464 ± .0069
	60	.8552 ± .0031	.8514 ± .0039	.8532 ± .0033	.8520 ± .0036
	80	.8662 ± .0037	.8628 ± .0056	.8643 ± .0045	.8634 ± .0052

Table A.48.: Intra cross project: Further fine-tuning on PROMISE FQ with Dalpiaz p-fold and different amounts of internal (training) examples (classification: IsOnlyQuality)

Approach	# Examples	IsFunctional			
		P	R	F1	F2
norbert-base	20	.8312 ± .0068	.8307 ± .0073	.8307 ± .0073	.8307 ± .0074
	40	.8424 ± .0053	.8424 ± .0053	.8424 ± .0053	.8424 ± .0053
	60	.8482 ± .0057	.8480 ± .0058	.8480 ± .0058	.8480 ± .0058
	80	.8540 ± .0068	.8540 ± .0068	.8540 ± .0068	.8540 ± .0068
	100	.8574 ± .0046	.8574 ± .0047	.8574 ± .0046	.8574 ± .0047
norbert-base(os)	20	.8368 ± .0057	.8365 ± .0058	.8365 ± .0057	.8365 ± .0057
	40	.8494 ± .0037	.8491 ± .0038	.8491 ± .0038	.8491 ± .0038
	60	.8543 ± .0078	.8539 ± .0080	.8539 ± .0080	.8539 ± .0080
	80	.8639 ± .0071	.8637 ± .0072	.8638 ± .0072	.8638 ± .0072
	100	.8614 ± .0053	.8611 ± .0051	.8612 ± .0051	.8611 ± .0051
norbert-large	20	.8412 ± .0043	.8410 ± .0044	.8410 ± .0043	.8409 ± .0043
	40	.8562 ± .0036	.8562 ± .0036	.8562 ± .0036	.8562 ± .0036
	60	.8631 ± .0041	.8630 ± .0044	.8630 ± .0044	.8629 ± .0044
	80	.8735 ± .0064	.8733 ± .0064	.8734 ± .0064	.8733 ± .0064
	100	.8713 ± .0011	.8711 ± .0011	.8712 ± .0011	.8711 ± .0011
norbert-large(os)	20	.8376 ± .0071	.8371 ± .0075	.8371 ± .0074	.8370 ± .0075
	40	.8499 ± .0041	.8495 ± .0041	.8495 ± .0041	.8495 ± .0041
	60	.8601 ± .0084	.8599 ± .0085	.8600 ± .0085	.8599 ± .0085
	80	.8705 ± .0079	.8703 ± .0081	.8704 ± .0081	.8703 ± .0081
	100	.8701 ± .0036	.8700 ± .0035	.8700 ± .0035	.8700 ± .0035
eltahier	20	.8341 ± .0031	.8340 ± .0031	.8340 ± .0031	.8340 ± .0031
	40	.8444 ± .0050	.8444 ± .0050	.8444 ± .0050	.8444 ± .0050
	60	.8511 ± .0016	.8510 ± .0016	.8510 ± .0015	.8510 ± .0015
	80	.8583 ± .0052	.8582 ± .0052	.8582 ± .0052	.8582 ± .0052
	100	.8607 ± .0031	.8607 ± .0030	.8606 ± .0030	.8606 ± .0030
eltahier(os)	20	.8306 ± .0061	.8303 ± .0062	.8303 ± .0062	.8303 ± .0062
	40	.8458 ± .0070	.8458 ± .0071	.8458 ± .0071	.8458 ± .0071
	60	.8472 ± .0104	.8472 ± .0104	.8472 ± .0104	.8472 ± .0104
	80	.8576 ± .0050	.8576 ± .0050	.8576 ± .0050	.8576 ± .0050
	100	.8560 ± .0062	.8560 ± .0062	.8560 ± .0062	.8560 ± .0062

Table A.49.: Intra cross project: Further fine-tuning on PROMISE FQ with custom p-fold and different amounts of internal (training) examples (classification: IsFunctional)

Approach	# Examples	IsQuality			
		P	R	F1	F2
norbert-base	20	.9028 ± .0078	.8937 ± .0073	.8977 ± .0073	.8952 ± .0073
	40	.9036 ± .0052	.9001 ± .0060	.9017 ± .0054	.9008 ± .0057
	60	.9105 ± .0031	.9047 ± .0033	.9074 ± .0032	.9058 ± .0033
	80	.9163 ± .0041	.9100 ± .0054	.9129 ± .0048	.9111 ± .0052
	100	.9214 ± .0020	.9146 ± .0021	.9177 ± .0019	.9158 ± .0020
norbert-base(os)	20	.8803 ± .0058	.8739 ± .0064	.8767 ± .0060	.8749 ± .0062
	40	.8898 ± .0078	.8876 ± .0080	.8886 ± .0078	.8880 ± .0079
	60	.8997 ± .0034	.8984 ± .0047	.8990 ± .0040	.8987 ± .0044
	80	.9110 ± .0037	.9085 ± .0036	.9097 ± .0034	.9090 ± .0035
	100	.9124 ± .0017	.9120 ± .0008	.9122 ± .0011	.9121 ± .0009
norbert-large	20	.8810 ± .0028	.8708 ± .0070	.8751 ± .0053	.8723 ± .0064
	40	.8952 ± .0074	.8947 ± .0095	.8949 ± .0085	.8948 ± .0091
	60	.9093 ± .0070	.9075 ± .0058	.9083 ± .0062	.9078 ± .0059
	80	.9208 ± .0043	.9181 ± .0044	.9194 ± .0042	.9186 ± .0043
	100	.9212 ± .0078	.9171 ± .0061	.9190 ± .0069	.9179 ± .0064
norbert-large(os)	20	.8849 ± .0042	.8783 ± .0052	.8810 ± .0035	.8792 ± .0045
	40	.8938 ± .0075	.8944 ± .0070	.8941 ± .0072	.8943 ± .0070
	60	.9066 ± .0050	.9051 ± .0063	.9058 ± .0056	.9053 ± .0060
	80	.9146 ± .0062	.9111 ± .0072	.9127 ± .0067	.9117 ± .0070
	100	.9175 ± .0036	.9136 ± .0034	.9154 ± .0035	.9143 ± .0034
eltahier	20	.8925 ± .0057	.8864 ± .0040	.8891 ± .0047	.8874 ± .0042
	40	.9032 ± .0074	.8981 ± .0078	.9005 ± .0076	.8990 ± .0077
	60	.9100 ± .0064	.9055 ± .0046	.9076 ± .0054	.9063 ± .0049
	80	.9181 ± .0026	.9138 ± .0032	.9158 ± .0028	.9146 ± .0030
	100	.9202 ± .0025	.9147 ± .0030	.9172 ± .0024	.9156 ± .0027
eltahier(os)	20	.8802 ± .0084	.8776 ± .0087	.8786 ± .0080	.8779 ± .0083
	40	.8965 ± .0098	.8966 ± .0099	.8965 ± .0097	.8965 ± .0098
	60	.9018 ± .0068	.9016 ± .0081	.9017 ± .0075	.9016 ± .0079
	80	.9103 ± .0055	.9094 ± .0060	.9098 ± .0058	.9096 ± .0059
	100	.9140 ± .0036	.9123 ± .0030	.9131 ± .0033	.9126 ± .0031

Table A.50.: Intra cross project: Further fine-tuning on PROMISE FQ with custom p-fold and different amounts of internal (training) examples (classification: IsQuality)

Approach	# Examples	IsOnlyFunctional			
		P	R	F1	F2
norbert-base	20	.8714 ± .0067	.8498 ± .0020	.8596 ± .0032	.8535 ± .0021
	40	.8813 ± .0044	.8689 ± .0027	.8747 ± .0018	.8712 ± .0020
	60	.8832 ± .0056	.8712 ± .0089	.8769 ± .0074	.8734 ± .0083
	80	.8898 ± .0087	.8768 ± .0081	.8830 ± .0083	.8792 ± .0082
	100	.9008 ± .0059	.8816 ± .0034	.8905 ± .0038	.8850 ± .0033
norbert-base(os)	20	.8609 ± .0039	.8398 ± .0067	.8493 ± .0044	.8433 ± .0058
	40	.8807 ± .0087	.8609 ± .0109	.8699 ± .0087	.8643 ± .0100
	60	.8901 ± .0048	.8715 ± .0041	.8800 ± .0029	.8748 ± .0034
	80	.8952 ± .0077	.8813 ± .0056	.8878 ± .0054	.8838 ± .0052
	100	.8964 ± .0077	.8823 ± .0079	.8890 ± .0073	.8849 ± .0076
norbert-large	20	.8590 ± .0056	.8386 ± .0101	.8478 ± .0075	.8420 ± .0091
	40	.8830 ± .0078	.8690 ± .0050	.8755 ± .0057	.8715 ± .0051
	60	.8904 ± .0078	.8774 ± .0067	.8835 ± .0061	.8797 ± .0063
	80	.9016 ± .0066	.8892 ± .0023	.8950 ± .0031	.8914 ± .0021
	100	.9043 ± .0074	.8872 ± .0077	.8951 ± .0067	.8902 ± .0072
norbert-large(os)	20	.8725 ± .0051	.8461 ± .0151	.8575 ± .0095	.8502 ± .0131
	40	.8814 ± .0046	.8693 ± .0032	.8750 ± .0033	.8715 ± .0031
	60	.8934 ± .0102	.8770 ± .0086	.8846 ± .0082	.8799 ± .0082
	80	.9002 ± .0054	.8860 ± .0079	.8927 ± .0063	.8886 ± .0072
	100	.8988 ± .0040	.8860 ± .0029	.8920 ± .0029	.8883 ± .0028
eltahier	20	.8653 ± .0072	.8489 ± .0051	.8565 ± .0060	.8518 ± .0054
	40	.8760 ± .0055	.8617 ± .0051	.8684 ± .0053	.8643 ± .0051
	60	.8855 ± .0031	.8736 ± .0060	.8792 ± .0043	.8758 ± .0054
	80	.8952 ± .0100	.8842 ± .0047	.8893 ± .0058	.8861 ± .0046
	100	.8950 ± .0073	.8830 ± .0027	.8886 ± .0023	.8851 ± .0014
eltahier(os)	20	.8635 ± .0102	.8427 ± .0087	.8518 ± .0059	.8461 ± .0072
	40	.8831 ± .0039	.8709 ± .0101	.8765 ± .0066	.8731 ± .0087
	60	.8859 ± .0039	.8726 ± .0099	.8787 ± .0064	.8749 ± .0085
	80	.8955 ± .0029	.8821 ± .0045	.8883 ± .0017	.8845 ± .0034
	100	.8927 ± .0057	.8815 ± .0034	.8868 ± .0041	.8835 ± .0035

Table A.51.: Intra cross project: Further fine-tuning on PROMISE FQ with custom p-fold and different amounts of internal (training) examples (classification: IsOnlyFunctional)

Approach	# Examples	IsOnlyQuality			
		P	R	F1	F2
norbert-base	20	.8472 ± .0128	.8493 ± .0102	.8480 ± .0115	.8488 ± .0107
	40	.8552 ± .0070	.8548 ± .0081	.8550 ± .0075	.8549 ± .0078
	60	.8647 ± .0077	.8629 ± .0083	.8636 ± .0078	.8632 ± .0080
	80	.8690 ± .0100	.8679 ± .0095	.8684 ± .0097	.8681 ± .0096
	100	.8699 ± .0039	.8686 ± .0040	.8692 ± .0039	.8688 ± .0039
norbert-base(os)	20	.8436 ± .0056	.8428 ± .0056	.8431 ± .0053	.8429 ± .0054
	40	.8523 ± .0081	.8510 ± .0099	.8516 ± .0090	.8512 ± .0095
	60	.8576 ± .0085	.8565 ± .0095	.8570 ± .0089	.8566 ± .0093
	80	.8693 ± .0064	.8680 ± .0062	.8686 ± .0061	.8682 ± .0061
	100	.8705 ± .0037	.8686 ± .0035	.8695 ± .0035	.8689 ± .0035
norbert-large	20	.8479 ± .0044	.8488 ± .0067	.8483 ± .0054	.8485 ± .0061
	40	.8629 ± .0080	.8582 ± .0092	.8603 ± .0085	.8590 ± .0089
	60	.8712 ± .0064	.8686 ± .0080	.8696 ± .0068	.8689 ± .0075
	80	.8844 ± .0122	.8842 ± .0107	.8843 ± .0115	.8842 ± .0110
	100	.8806 ± .0074	.8809 ± .0063	.8807 ± .0068	.8808 ± .0065
norbert-large(os)	20	.8425 ± .0073	.8424 ± .0094	.8423 ± .0083	.8424 ± .0089
	40	.8574 ± .0081	.8545 ± .0111	.8557 ± .0095	.8550 ± .0105
	60	.8686 ± .0078	.8663 ± .0083	.8673 ± .0079	.8667 ± .0081
	80	.8768 ± .0123	.8766 ± .0109	.8766 ± .0116	.8766 ± .0112
	100	.8802 ± .0065	.8785 ± .0079	.8793 ± .0071	.8788 ± .0075
eltahier	20	.8477 ± .0097	.8451 ± .0085	.8463 ± .0090	.8456 ± .0087
	40	.8601 ± .0061	.8577 ± .0061	.8588 ± .0060	.8581 ± .0060
	60	.8645 ± .0034	.8617 ± .0052	.8630 ± .0042	.8622 ± .0048
	80	.8704 ± .0058	.8682 ± .0060	.8693 ± .0058	.8686 ± .0059
	100	.8731 ± .0080	.8701 ± .0065	.8715 ± .0072	.8706 ± .0067
eltahier(os)	20	.8377 ± .0038	.8342 ± .0060	.8356 ± .0044	.8347 ± .0053
	40	.8520 ± .0032	.8480 ± .0024	.8498 ± .0024	.8487 ± .0023
	60	.8556 ± .0137	.8503 ± .0152	.8527 ± .0146	.8512 ± .0150
	80	.8652 ± .0064	.8607 ± .0063	.8627 ± .0060	.8615 ± .0061
	100	.8693 ± .0072	.8656 ± .0074	.8673 ± .0072	.8662 ± .0073

Table A.52.: Intra cross project: Further fine-tuning on PROMISE FQ with custom p-fold and different amounts of internal (training) examples (classification: IsOnlyQuality)