MASTER'S THESIS 2026

Computationally Limited Product Type Classification

Melissa Engberg, Samuel Högfeldt

ISSN 1650-2884
LU-CS-EX: 2026-50

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2026-50

Computationally Limited Product Type Classification

Beräkningsmässigt Begränsad Produktklassificering

Melissa Engberg, Samuel Högfeldt

Computationally Limited Product Type Classification

Melissa Engberg
me8614en-s@student.lu.se

Samuel Högfeldt
sa2446ho-s@student.lu.se

July 1, 2026

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Noric Couderc, noric.couderc@cs.lth.se
Andreas Björklund, andreas.bjorklund@voyado.com

Examiner: Elizabeth Bjarnason, elizabeth.bjarnason@cs.lth.se

Abstract

Large Language Models (LLMs) are widely used today because of their ability to assist with a wide range of different tasks. This has led many companies and organizations to rely on LLM providers to deliver products and services. However, LLMs require considerable computational power and are often closed-source, hosted in external data centers, and require paid subscriptions to use.

This thesis investigates whether open-source Small Language Models (SLMs) with fewer than 500 million parameters, fine-tuned and run entirely on consumer grade hardware, can be used to classify Swedish fashion products into product types based on their title and description text. Three pre-trained embedding models and three zero-shot classification models were evaluated using standard performance metrics, hardware usage, and a set of custom metrics designed in this thesis to give a more nuanced assessment of misclassification severity. Once fine-tuned, all three embedding models achieved accuracies between approximately 89% and 96% depending on the dataset, while the zero-shot models performed considerably worse and required more computational resources. Of the three embedding models, MiniLM offered the most favorable trade-off between performance and hardware usage. Combining our standard and custom metrics, we estimate that these models produce an acceptable classification in around 98% of cases. These results suggest that small, fine-tuned, open-source SLMs offer a viable and considerably more lightweight alternative to LLMs for fashion product type classification.

Keywords: LLM, SLM, classification, confusion matrix, evaluation metrics, performance metrics, fine-tune

Acknowledgements

To make this project possible we are grateful for the help we received from multiple people. First and foremost, we want to thank Noric for being our supervisor at LTH and being a steady support throughout this whole project. We also want to give a huge thank you to Andreas who always challenged our thinking and methods in a way that enhanced not only our experience when doing the project but also the results. Lastly we want to thank Jessica at Voyado who helped us throughout the process of getting this project along with Voyado, who aided us in getting a project that enriched us in knowledge and a place to work at.

Declaration of AI Usage

During the course of this project the AI tools Claude and ChatGPT were utilized in a limited and supplementary capacity. The primary usage of AI was to find resources or keywords that we verified and evaluated independently. This is explained further in Section 3.1.1.

AI tools were in occasional instances used for language refinements. This involved using ChatGPT and Claude to find and point out grammatical errors or unprofessional writing in the text. We then reviewed the errors and fixed them accordingly to how we saw fit.

All analytical reasoning, methodology and conclusions presented in this thesis are entirely our own and the usage of AI tools did not interfere with our independent thinking or academic judgment.

Contents

1 Introduction **9**
- 1.1 Voyado . . . 10
- 1.2 The Problem . . . 10
- 1.3 Research Questions . . . 12

2 Background and Related Work **13**
- 2.1 Terminology . . . 13
- 2.2 Artificial Intelligence . . . 13
- 2.3 Performance Metrics . . . 14
- 2.4 Models . . . 16
- 2.5 Evaluation of a Model . . . 19
- 2.6 Classification Methods . . . 21
- 2.7 Related Work . . . 21
  - 2.7.1 Performance Trade-offs of Optimizing Small Language Models for E-Commerce . . . 21
  - 2.7.2 Small Language Models are Diligent Learners: Evaluation of Embedding- and Transformer-based Language Models for Text Classification and Recommendation . . . 22

3 Methodology **23**
- 3.1 Literature Review . . . 23
  - 3.1.1 Literature Study . . . 24
  - 3.1.2 Selecting Literature . . . 25
- 3.2 Identification and Selection of Models . . . 25
  - 3.2.1 Selection Criteria . . . 25
  - 3.2.2 Classification Technique Selection . . . 25
- 3.3 Model Evaluation Metrics . . . 26
- 3.4 Experiments . . . 26
  - 3.4.1 Experimental Design and Setup . . . 26
  - 3.4.2 Experiment 1: Pre-Trained SLMs . . . 30

3.4.3 Experiment 2: Fine-Tuning . . . 30
3.4.4 Experiment 3: Zero-Shot Classification . . . 31

4 Results **33**
4.1 Literature Review . . . 33
4.1.1 Chosen models . . . 33
4.1.2 Existing performance metrics . . . 34
4.1.3 Custom Design Metrics . . . 34
4.2 Experiments . . . 40
4.2.1 Evaluation of Pre-Trained and Fine-Tuned Embedding Models . . . 40
4.2.2 Evaluation of Zero-Shot Classification Models . . . 48
4.3 Misclassification Review . . . 48
4.3.1 E5 Small . . . 49
4.3.2 E5 Multi . . . 49
4.3.3 MiniLM . . . 49

5 Discussion **51**
5.1 Research Questions . . . 51
5.1.1 Evaluation Metrics (RQ1) . . . 51
5.1.2 Domain-Specific Evaluation Metrics (RQ2) . . . 52
5.1.3 Model Selection (RQ3) . . . 53
5.1.4 Model Performance (RQ4) . . . 53
5.1.5 Effect of Fine-Tuning (RQ5) . . . 54
5.2 Limitations and Threats to Validity . . . 55
5.2.1 Construct Validity . . . 55
5.2.2 Internal Validity . . . 55
5.2.3 External Validity . . . 56
5.2.4 Reliability . . . 56
5.2.5 Quality of Cited Sources . . . 56

6 Conclusions **57**
6.1 Future Work . . . 59

References **61**

Chapter 1
Introduction

In 2025, 78% of EU internet users were shopping online[1]. Being one of the most populated fields of the internet, online retail is a industry that has to handle a lot of traffic and manage a great deal of data. Managing the inventory for online retailers have become increasingly tedious as the field continues to grow and more products are uploaded online. For a field like fashion where inventory is constantly updated the task becomes unreasonable to perform by hand. There is therefore an incentive to automate the process of inventory management.

To power that automation many companies today use LLMs (Large Language Models). In online retail these models can be used for generating product descriptions, answer questions about products and automated product type classifications [2]. The problems with LLMs originate from their large size. Due to the computational cost of these models, they are usually hosted on external data centers, charging high token costs for larger systems, making them unsustainable to use for many companies. It is therefore worthwhile to investigate alternatives to LLMs for the online retail domain.

Current research indicates that SLMs (Small Language Models) are for many tasks as capable as LLMs [3]. SLMs are, as the name suggests, language models of smaller parameter sizes. The smaller size makes them less computationally intensive, allowing for training and inference without the need for large data centers. Many of them are available online with their code usually being open sourced.

Within the domain of online retail, SLMs have been proven to compete with state-of-the-art models [4]. However, as Gupta et al.[3] points out, there is no standard definition of what exactly constitute a SLM. The current research for the domain mainly covers SLMs with parameter sizes in the low billions, using data center enterprise grade GPUs such as the Nvidia T4 [4] to train the models. Whilst these models are smaller and more viable compared to LLMs they still require specialized hardware to be utilized. That is why researching if even smaller models are possible.

Furthermore, other research suggests that smaller SLMs, in the range of 100M parameters, perform well for semantic understanding and classification tasks [5]. This said, the research only covers general semantic cases and is not applied on the field of online retail. However, it would suggest that these models might be suitable for product classifications.

In this thesis we will, together with the retail solutions company Voyado, investigate the feasibility of classifying Swedish fashion products using SLMs with parameter sizes under 500M. Only utilizing consumer grade hardware, we explore the possibility of fine-tuning these models on domain specific data and how that affects performance. Additionally, we investigate what metrics should be used and how we can design additional custom metrics in order to get a nuanced perception of how the models would perform in practice. We see our contributions to the field as the following.

Firstly, we designed custom metrics inspired by Nearest Neighbor classification that can be used to evaluate a models clustering capabilities. These metrics can also be used to extract critical misclassifications for domains where label classifications is not necessarily binary. Our hope is that these metrics can be used to promote a more nuanced evaluation of models practical performances.

Secondly, we provide a look at using fine-tuned SLMs with parameter sizes under 500M parameters, trained on consumer grade hardware, for product type classifications in online retail. We aim for this contribution to serve as a groundwork for implementation of SLMs for online retail companies that may lack the hardware or economical resources to make full use of LLMs.

1.1 Voyado

Voyado is a leading company within AI retail solutions. Today the company works with over 500 brands in more than 100 markets [6], where one of the main focuses is on automating merchandising [7]. Working with Voyado for this thesis we were provided resources in the form of pre-labeled datasets, suitable computers for the tasks and an office to work at. Due to a non-disclosure agreement with Voyado, the underlying product data and the identities of the involved retailers are not going to be made publicly available in this thesis.

1.2 The Problem

In this thesis we investigate SLMs ability to classify products within fashion retail. We define the task of product type classification as, with knowledge of all possible product types, predicting the type of a product given a text description of the product. In this thesis, the input will consist of the title and descriptions of the product in Swedish.

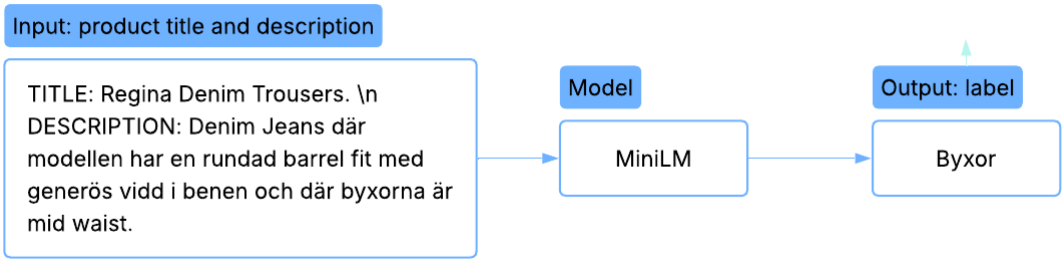

Figure 1.1: Example of product input and classification.

Previous research suggests that pre-trained models are well performing in the field because of their semantic understandings. Because of this we are limiting us to already built model architectures with adequate pre-training in contrast to building our own model. Pre-trained models are well performing, but there are several models in the same category, and there are no clear guidelines as to which model to choose to classify retail products.

Furthermore, research shows that larger pre-trained SLMs that have been further trained on domain specific data with methods such as fine-tuning perform well in tasks [4]. We want to find out if the models we chose, which are smaller, would perform better after fine-tuning when done within the domain of fashion retail.

As this thesis pertains to computationally limited classification it is important to measure the resource consumption. Licardo et al. [4] documents the resources for inference but not for fine-tuning. We believe that the fine-tuning process is important to measure in order to get a good idea on what hardware requirements are needed to train the models.

To evaluate the accuracy of a model, one should consider the context in which it is used. Licardo et al. [4] uses Exact Match Accuracy to evaluate the performance of the investigated models because they argue that "partial correctness in this e-commerce task can lead to significant functional errors in a production system" (p. 5). For our task we argue the opposite. Consider the scenario in Figure 1.2. One model predicts a T-shirt to be a tank top where as the other model predicts it to be a pair of jeans. We believe Model 1 to be more correct than Model 2. There could even exist edge products where humans have a hard time classifying whether it is a T-shirt or a tank top. Furthermore there could be a scenario where a fashion dataset is dominated by a certain category. A model that labels everything as t-shirts has a 90% accuracy for a dataset consisting of 90% t-shirts. With this thesis we therefore want to provide metrics that gives a nuanced view of a models performance in practice.

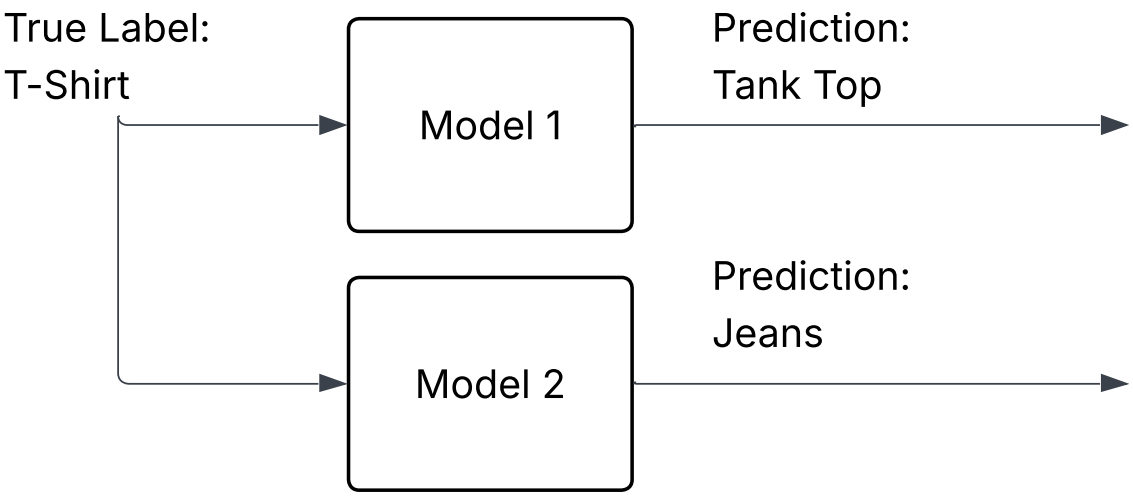

Figure 1.2: Two misclassifications that are not equally wrong.

1.3 Research Questions

To help navigate our work, we devised the following research questions. These concern how the performance of different models can be measured, using both existing metrics and the domain-specific ones we developed. Which SLMs are best suited for our end goal, with the aim of identifying the best-performing models. Additionally, we also wanted to see if we could improve some of the models when applicable.

RQ1: What existing metrics can be used to evaluate a model's performance, such as quality of predictions and hardware usage?

RQ2: How can a domain-specific metric be designed to give a better perception of the model's quality of predictions?

RQ3: What current license free, offline, SLMs are suitable for classification tasks based on popularity and model purpose?

RQ4: How do different license free SLMs perform in the context of classification of fashion products regarding quality of predictions and hardware usage?

RQ5: How can fine-tuning improve the model's performance when applicable?

Chapter 2
Background and Related Work

In order to get a better understanding of the technologies and strategies we will be using in our thesis, this section will be dedicated to explaining some background along with related work.

2.1 Terminology

AI	Artificial Intelligence
GAI	Generative Artificial Intelligence
LLM	Large Language Model
SLM	Small Language Model
NLP	Natural Language Processing
Model	A model here is defined as something which takes textual input and produces a label as output.
Label	The attribute that a classification model predicts, in this case most often referring to the product type.
True label	A label which can be seen as the ground truth.
Predicted label	The label the model predicts.

2.2 Artificial Intelligence

Artificial Intelligence (AI) encompasses a wide range of differentiating technologies, and is generally described as machines being able to act or think intelligently. What is defined as intelligent changes over time as computers become increasingly complex. The usage of AI

has increased and is utilized in tools such as search engines, robot vacuum cleaners, online shopping and social media [8]. That is why this section explains some of what AI encompasses that is related to this thesis.

Generative Artificial Intelligence

Generative Artificial Intelligence (GAI) refers to AI that generates content such as audio, code, images, text, simulations and videos. This can be seen in products such as DALL-E2 and ChatGPT, that generates images and text respectively based on text input. The GAI in this thesis only revolves around text and is built upon the field of Natural Language Processing (NLP), utilizing techniques such as transformer-based architectures. This thesis focuses mainly on SLMs, which are smaller LLMs [9].

Large Language Models

Large Language Models (LLMs) is a subdivision of GAI which has quickly in just a few years become wide spread [10]. LLMs can be seen in most chatbots today, such as ChatGPT and Claude. When interacting with the chatbots, the users questions gets sent to the LLMs which then generates a textual answer. LLMs generally have billions of parameters. Due to the large size of the LLMs they usually have more generalized capabilities than SLMs. LLMs usually require a lot of computational power to train along with a tremendous amount of data. For the customer, this can entail more latency. The query has to pass through a lot more layers in the LLM compared to the SLM. This is why LLMs usually perform better in a wider scope, but less so in more niche fields [11].

Small Language Models

Small Language Models (SLMs), as the name implies, are smaller language models compared to their LLM counterparts. SLMs operate on similar principles to LLMs. One key difference is given that SLMs has fewer parameters than LLMs, they are more often used when dealing with specific rather than broad tasks. Their smaller size enables them to run on resource-constrained devices. It also entails that they can run without the need of an external computation device or server, which can increase the speed at which a model can predict a label [12].

2.3 Performance Metrics

Performance metrics are often used to help evaluate a model, each giving a different insight into a model's performance. Performance metrics include both those that pertain to how well a model can predict a label correctly and others on the hardware usage needed to use the model. This thesis will be working on a multiclass task, which entails that there are more than two labels a fashion item can be labeled as. To estimate how well a model performs on the multiclass task, some of the standard metrics are accuracy, precision, recall, and F1-scores [13]. To understand how these metrics give insight, comprehending that classifying models categorizes their predictions into four different categories greatly helps. These can be seen in Table 2.1, where T stands for true, F for false, P for positive and N for negative. TP shows

how many of the predicted true labels are actually true. FP stands for the amount of labels which are predicted to be positive but which are not. FN are the number of labels which are predicted to be negative but are positive and finally TN are the labels which are predicted to be negative which they also are. These are important for understanding the different metrics which will be mentioned.

		Predicted label	
		Positive	Negative
True label	Positive	TP	FN
	Negative	FP	TN

Table 2.1: Predictions table explaining the different prediction categories and their associated shortened form.

Accuracy

Accuracy is a measure which can be used to get a first indication of how well the model performs. However, accuracy can also be misleading. It is sensitive to noise, and usually has a bias to the majority class when it comes to classification. This means that if the majority class performs well, there is a risk that the accuracy shows a favorable value, despite the model's risk of being horrible at classifying other classes. Accuracy can be calculated using the formula in (2.1).

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

Precision

Precision is the metric which calculates how many of the predicted true labels are actually true labels. It is a good metric when wanting a model's predicted true labels to be more correct than not. A higher value in precision means that the false positives are fewer, resulting in the model showing a higher performance in correctly classifying the positives as positives, whilst also being good at not classifying the negatives as positives. Precision is calculated using the formula in (2.2).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.2}$$

Recall

Recall measures how well the model captures all the positive cases. A low value therefore indicates that it classifies more positives as negatives. This measure is particularly relevant in medical contexts, such as hospitals. Here, a false positive is preferable to a false negative, since the former can be further investigated whilst the latter could be fatal. A higher recall is in these cases preferred. Recall is calculated using the formula in (2.3).

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.3}$$

F1-score

F1-score is a metric which utilizes the calculations of both the precision and the recall measurements. It is the harmonic mean of precision and recall and gives a result that is a more balanced value of the two. This is to ensure that neither metric is optimized at the expense of the other. The formula to calculate it can be seen in (2.4) [14].

$$\text{F1-score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.4}$$

Hardware Usage

Hardware constraints limit which device a model can be run or trained on. It is therefore important to understand the hardware usage of models. The most important hardware to consider is the CPU and GPU usage along with the process's Resident Set Size (RSS), which refers to the memory usage. These measurements together give an indication of how much hardware a model requires to operate [15].

2.4 Models

Embedding models

AI models often work with replicating complex patterns by constructing large networks of mathematical operations. To work with inputs such as text the model then needs some sort of numerical representation of this text. In early AI research one way to do this could be to represent one word as its index in a vocabulary [16].

In the report Efficient Estimation of Word Representations in Vector Space by Mikolov et al. (2013) [16] they propose representing words with numerical vectors. The idea is that by doing this the similarity between words can then be found by looking at the cosine distance within the vector space. It is then possible to analyze multiple dimensions of similarity between words rather than relying on one linear score. By training a model to place similar words close to one another in the vector space they were able to get a model which performed well in word similarity tasks.

In this thesis we will refer to these vectors as embeddings, a vector representation of text which keeps semantic information of the text. Most of the models we will cover in this section are embedding models, provided a text input they will provide an embedded output.

To then actually perform NLP tasks you would work with these embeddings. For our task of classification you would have to attach a classification head to the embedding model. This layer usually works within the principle that similar words and text are close to one another in the vector space and will be covered in Section 2.6.

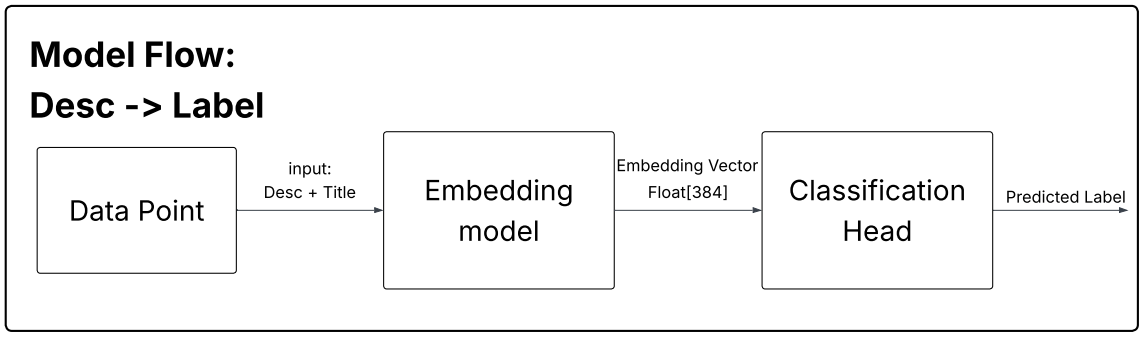

Figure 2.1: Structure of an embedding model combined with a classification layer.

Hugging Face

Hugging Face is a popular platform for open-source AI research. The website hosts a wide range of different AI models, datasets and leaderboards [17]. It also offers Python libraries that can be used to download models, run them and train them in your own local environment.

Fine-Tuning

Fine-tuning a pre-trained model entails that the pre-trained model's internal weights are adjusted to fit the provided training data. It is important that improving the model does not make it overfit the data, making it only memorize it. One can fine-tune a model in different ways. On Hugging Face you can find different hyperparameters for you trainer, which adjusts the weight of an embedding model based on its end goal.

BERT

Developed by Google Engineers in 2018, BERT is an open-source language model made to excel in language understanding. BERT is built on a bidirectional, encoder-only, transformer in contrast to left-to-right transformers like OpenAI's Generative Pre-trained Transformer (GPT) model. The transformer is trained on predicting masked words in a sentence. That training step is then followed up by a sentence prediction task to predict the following sentence.

The BERT model is mostly concerned about language understanding rather than text generation as in a GPT model. Because of this the transformer follows an encoder only structure. Consequently, the model performs well for tasks such as question answering along with classifications [18].

MiniLM

MiniLM was a model developed by Microsoft where the goal was to have a smaller model that replicates the performance of the BERT model in a more compact form. Early on a concern of the BERT model was that it at the time had a large parameter size. To achieve their goal,

Microsoft performed a process of distillation on the BERT model. This is a process where a model is trained on copying the outputs of a larger model. Distillation is not unique to the MiniLM model. The distillation of MiniLM is, however, unique as they only distilled the last transformer layer of the BERT model. Compared to other BERT distillations like TinyBert and DistillBERT, MiniLM excels in NLP tasks. This makes it a suitable candidate for text classification with small models. In this thesis we will this model will be reflected in the *sentence-transformers/all-MiniLM-L6-v2* model [19].

E5

E5, also developed by Microsoft engineers, is a model that focuses on improvements in the data collection phase. To achieve this, it uses the CCPair dataset consisting of queries and passages from sources such as Reddit, Stackoverflow, and Wikipedia.

The data is then filtered through a process called K-Top filtering. K-Top filtering ensures a higher level of uniqueness for queries and passages. In turn, the model is particularly good in definitive classification tasks.

The underlying model structure used for E5 depends on which size is used. In this thesis we will primarily cover the *intfloat/small-e5-v2* and *intfloat/multilingual-e5-small* models. The *intfloat/small-e5-v2* model is a BERT-based model and is trained on the English language. The *intfloat/multilingual-e5-small* model is similar but supports 100 languages, including the Swedish language [20] [21].

Zero-Shot Models

Zero-shot classification models involve using a model pre-trained on data independent from the user's data, to classify the input. This is especially advantageous if you lack a large dataset to train a model with or if you lack the hardware to train it.

On Hugging Face there are multiple different variations of zero-shot models. Most of the popular models are created by Facebook. The most common of these is the *facebook/bart-large-mnli* model, which is a BART model. BART is a model which uses a mixed approach to transformers, taking ideas from both the GPT and BERT models and has shown good results in zero-shot classification [22]. Another very popular model is the *roberta-large-mnli* model, also created by Facebook. This is a successor of the BERT model, where the main differences are the hyperparameters that were optimized on the BERT model . Both of these models are trained only trained on the English language [23].

Another popular zero-shot classification model is the *MoritzLaurer/mDeBERTa-v3-base-mnli-xnli* model created by Microsoft and is not only BERT-based but also tailored toward multilingual usage. It is trained on 15 different languages, some of which include English and German [24] [25].

In our thesis we will use zero-shot models to compare to the performance of models fine-tuned to our data. With this, we hope to be able to provide a good idea on the advantages

and disadvantages of fine-tuning a model.

2.5 Evaluation of a Model

There are different metrics and methods which can be used to evaluate a model. Because of this, we will go through the different ways in which we will evaluate the models used in this thesis.

Cross-Validation

Cross-validation is a data resampling method used for validating a model. Its purpose is to validate a model whilst minimizing the risk of either underfitting or overfitting. A model can be described as having an algorithm, which it uses on data to get an output. The algorithm can be seen as the weights in a neural network and models involve both parameters and hyperparameters. The parameters can be seen as the internal weights which are automatically adjusted in the neural network during the training process, whilst the hyperparameters are used to control the training process itself. The hyperparameters are manually set, and can for example be the k in the k-Nearest Neighbors algorithm or the number of layers in a neural network. For the optimal hyperparameters, cross-validation and random subsampling can be used.

In cross-validation there are different sets, where **L** is the learning set, **R** is the training set, **T** the test set and **V** the validation set. During training, **R** and **V** are usually split up into i disjoint sets.

For cross-validation, random subsampling is done to get these sets. To ensure that the training and test sets have no cases in common, $R \cap T = \emptyset$ and $L = R \cup T$ must be true. Performance of the final model is estimated using the resulting model $\hat{f}(x, R)$. In order to estimate the error, the loss function $\mathcal{L}(y_i, \hat{y}_i)$ is used. Here, y_i is the correct classification and $\hat{y}_i$ is the predicted classification made by the model. The loss function $\mathcal{L}$ can be used to evaluate the training along with the test error. The training error is a metric to evaluate the model's adaptation to the training set. The test error is used to estimate the true prediction error, and therefore the generalization ability of the model.

Prior to cross-validation, preprocessing steps such as feature selection must be applied to avoid data leakage. Since testing on training data is forbidden, due to the bias it causes, there are different data sampling methods which can be used. Some of these are *single hold-out random subsampling*, *K-fold random subsampling*, *K-fold cross-validation* and *nested cross-validation* to name a few. In our code we will be using *stratified K-fold cross-validation* [26]. We next describe these techniques in general and *stratified K-fold* in particular.

K-Fold Cross-Validation

K-Fold cross-validation is used to enhance a model's performance estimation. A model's performance on a single train and validation split can be insufficient. Splits can have some labels

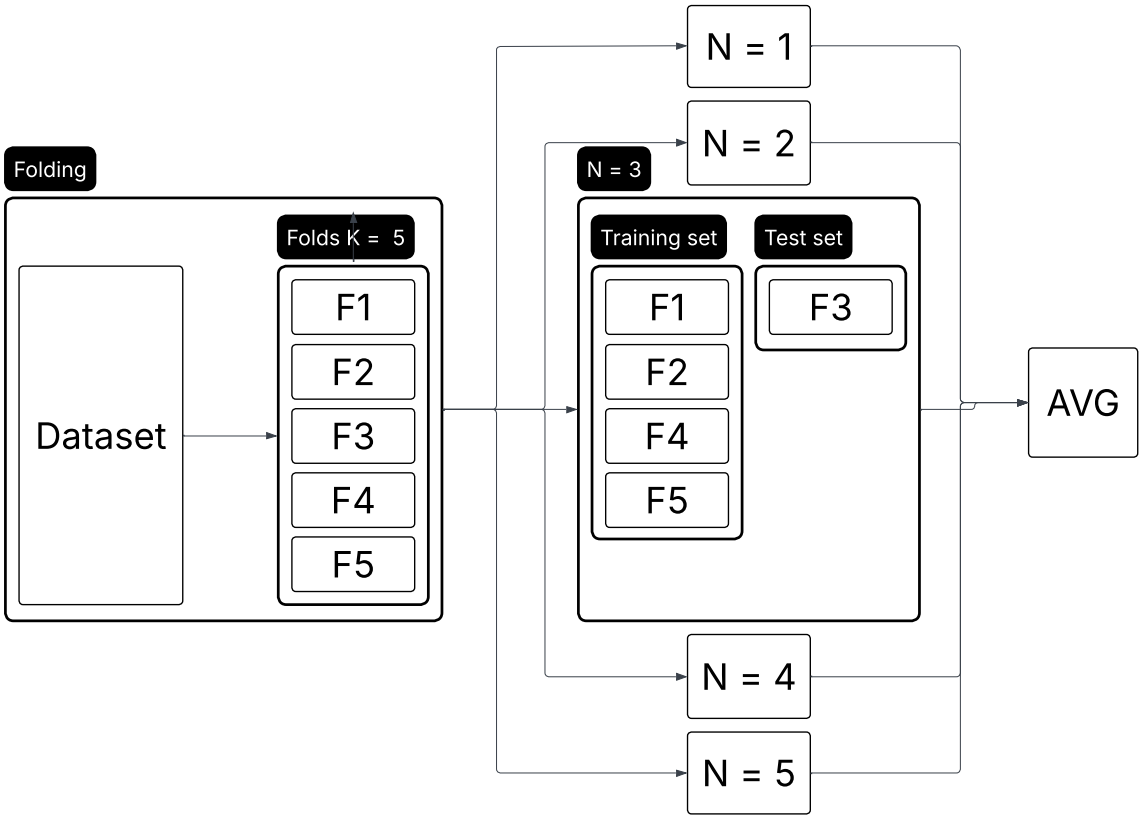

Figure 2.2: K-fold cross-validation for K = 5.

being exclusively in either the training or the validation sets. When that occurs, the performance estimate can become biased. To counter this, K-fold sampling can be used.

K-fold sampling creates K different splits, which are called folds, of equal size. Those K folds are then placed in K pairings of training and validation datasets where each fold appears in the training set $K-1$ times and once as the validation set. For each pair of training and validation sets, cross-validation is performed and the results are averaged across the K attempts [27].

Stratified K-Fold Cross-Validation

Stratified sampling is a sampling technique used on datasets whilst keeping the original proportions [27]. A stratified sample of 10% T-shirts and 90% pants would be a subset of the original dataset also containing 10% T-shirts and 90% pants. Combining this sampling method with K-fold cross-validation yields stratified K-fold cross-validation, which ensures that each fold maintains the class distributions that the original dataset contained.

2.6 Classification Methods

To be able to classify vector outputs of sentence embedding models, classification methods are used. Most of the models in the previous section are sentence embedding models. From the text-based input that the models are provided, they output vectors of numerical values that represent the features of the input. To classify the different inputs to the labels then requires a classifying layer. In the following section we will describe a few popular classification methods.

KNN

The K-Nearest Neighbor classifier predicts a label by observing the K closest neighbors in the vector space. K-Nearest Neighbors has been known to be costly on large datasets as finding the closest neighbors, especially in high-dimensional spaces, is a computationally expensive task. Furthermore the classifier must retain the entire dataset in memory for all the previously fitted data points. For our thesis we will investigate whether we can use KNN classification on top of sentence embedding models to get accurate classifications, across varying dataset sizes [28].

SVM

Support vector machines classify by setting up hyperplanes to divide data points into different labels. The technique is in its simplest form used for binary classification but by iterating you can use it for classifying multiple labels.

In some cases, the data points with different labels can not be separated by a plane. To circumvent this, non-linear kernel methods can be used to create more advanced divisions between the data points [29].

2.7 Related Work

We have identified two papers that we see as related to what we do in this thesis. In this section we will summarize them and explain what we can learn as well as what sets us apart from them.

2.7.1 Performance Trade-offs of Optimizing Small Language Models for E-Commerce

In the paper by Licardo and Tankovic [4] they claim that instead of relying on a single task-agnostic LLM, e-commerce companies could benefit from using a fleet of SLMs for each task they are trying to solve. To reach this claim, the paper measured different SLMs ability to extract intent from freeform text prompts describing a desired action to be performed to the shopping cart. It finds that a fine-tuned SLM called LLaMa 3.2 1B achieves 99% accuracy

when comparing it to the state-of-the-art LLM GPT 4.1.

Being mostly focused on the LLaMa model the paper also benchmarks the performances of popular SLMs Gemma 3 and 2 as well as Qwen 2.5, all in different parameter sizes. The smallest model in the test consists of 1 Billion parameters going up to 3 Billion for the largest. Whilst this would still constitute these models as relatively small, the largest LLaMA model reaches a parameter size of 405B [30]. The models are therefore still seemingly too large to train on consumer grade hardware as indicated by the Nvidia T4 enterprise server GPU used for fine-tuning in the paper. Our thesis investigates far smaller models performing everything on a consumer grade HP laptop.

For fine-tuning the paper produces a synthetic dataset by prompting the GPT 4.1 model to provide 3000 diverse examples of input. As pointed out in the limitations of study section, the performance of the model might actually vary when applied to real-world use. We are fortunate enough for our thesis to be provided with a large dataset of real products.

Another thing which the limitations of study brings up is the use of Exact Match Accuracy as the only metric. The paper uses Exact Match Accuracy as it argues that for its task, partial correctness could lead to significant errors. It does however acknowledge that even for this task, using only this metric leaves out nuance in answers that might be partially correct. We deem our task of product classification to leave a lot of room for partially correct classifications. A large part of our project is therefore to find and design the appropriate metrics to give a nuanced and more realistic insight on the performance of the models.

2.7.2 Small Language Models are Diligent Learners: Evaluation of Embedding-and Transformer-based Language Models for Text Classification and Recommendation

In the paper by Koutsomitropoulos and Skoulidis [5], they perform a survey on a range of different SLMs and measured their ability to embed and classify based on text. What the paper finds is that SLMs with a parameter size of 100M performs just 5% worse in comparison to a LLaMa model with 7B parameters. This paper does not concern itself with fine-tuning but suggests that even for the tasks we want to do in our thesis, models around 100M parameters could be of good use.

Chapter 3
Methodology

This chapter describes the methodology used throughout this thesis, which consists of two main parts. First, a literature review was conducted both to identify suitable models for our experiments (Section 3.2) and to identify and design metrics for evaluating them (Section 3.3). How this review was carried out is described in Section 3.1. Second, a series of three experiments was designed, run, and analyzed to compare the performance of the identified models on our classification task (Section 3.4). We worked with our supervisor at LTH and our supervisor at Voyado throughout both parts of this process.

Figure 3.1 gives a visual overview of this workflow. During the "Preparation" stage, we conducted the literature review described in Sections 3.1–3.3, gathering the knowledge needed to design our experiments and later interpret our results. In the "Experiments" stage, we used this review to choose which pre-trained and zero-shot models to evaluate, and fine-tuned the pre-trained models to obtain our fine-tuned models. These models were then incorporated into the code written during the "Implementation" stage, and tested using the data prepared during the "Data Preprocessing" stage. Finally, the "Testing" stage combined this code and data to produce the results analyzed in the "Analysis" stage and presented in Chapter 4.

3.1 Literature Review

To gain the background knowledge needed for this thesis, we conducted a literature review covering SLMs, evaluation methods, classification techniques, and zero-shot classification. This section describes how we searched for literature (Section 3.1.1) and how we selected which sources to use (Section 3.1.2). How this literature was subsequently used to identify candidate models and to identify and design evaluation metrics, both outcomes of the review rather than part of the search process itself, is described in Sections 3.2 and 3.3.

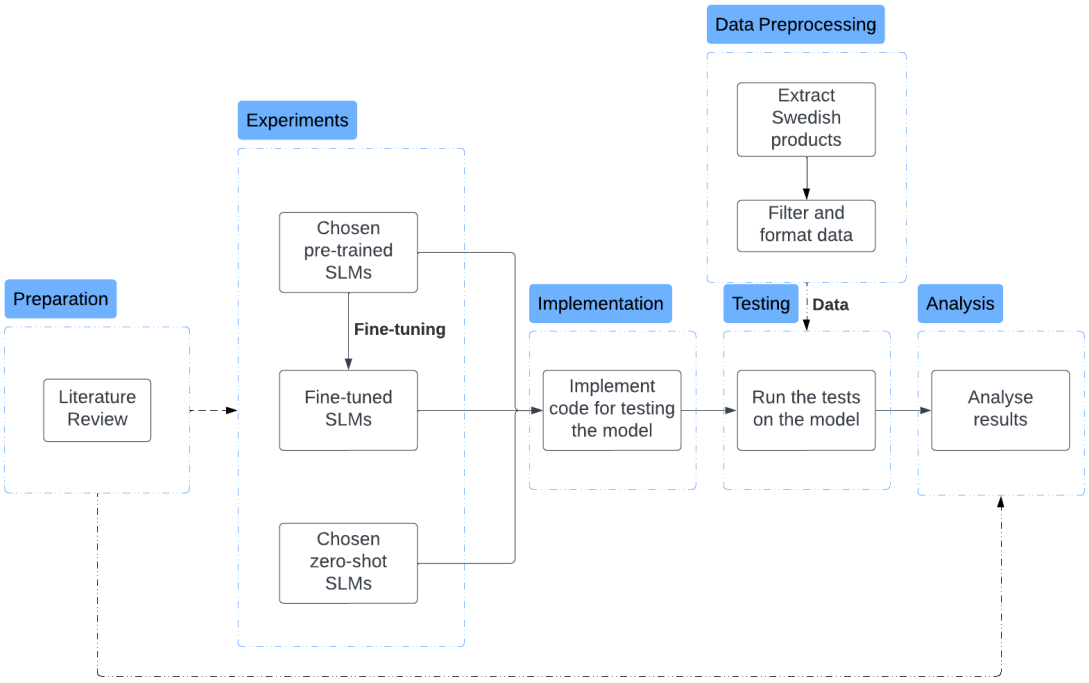

Figure 3.1: A chart of the methodology workflow.

3.1.1 Literature Study

Our main goal was to find known reliable sources along with peer-reviewed research papers. We utilized search services such as Finn and Google Scholar using keywords to obtain sources for what we searched for. These search services were chosen because of their large collection of peer-reviewed articles along with their filter functionality, making it easier to find the articles which are peer-reviewed. When we needed help with search queries, we used AI chatbots, such as Claude and ChatGPT, to get suggestions.

To find information about classifiers we turned towards scikit-learn documentation. Since scikit-learn implements a lot of popular machine-learning algorithms [31], is regularly maintained and is widely adopted, we believe it to be a good source to find popular classification techniques. By going to the classification section of the documentation we were presented with a handful of different classification techniques.

Finding literature for specific models was partly done through Hugging Face. By going to the platform and specifying the task to "sentence-similarity" or "zero-shot classification" and then sorting by trending or most downloads we were able to find current popular models that could be optimal for our tasks. To then find the literature behind the model we would scroll down to the papers section on the model's page.

3.1.2 Selecting Literature

When selecting literature, the most important factors were applicability and reliability. For reliability, most of the references in this thesis are peer reviewed. For applicability, we chose to select the sources which could be applied to the work done throughout this thesis. That also resulted in sources focusing on unrelated fields, for example multi-class labeling, meaning one item has multiple labels, were sometimes excluded.

When looking at literature for the models we chose to include research from established AI companies, including Microsoft, Google and Meta. When selecting the models the number of downloads along with the number of stars they had on Hugging Face also played a significant role. Additionally, since we are concerning ourselves with pre-trained models in this thesis, we believe these companies to have the most resources to produce the best pre-training.

3.2 Identification and Selection of Models

Having gathered and selected relevant literature, we analyzed it to identify which pre-trained and zero-shot models, and which classification technique, were best suited for our task. This section first describes our selection criteria (Section 3.2.1) and then how we selected a classification technique to pair with these models (Section 3.2.2).

3.2.1 Selection Criteria

We selected models based on their purpose, in our case text to text classification, along with their popularity. This was done both for the embedding models that were later fine-tuned along with the zero-shot classification models. Popular models tend to be more used and therefore more observed in their performance abilities.

Zero-shot classification means that the model makes predictions without any prior task-specific training data. We focused on these models, since they do not require any training at all, so they could be very economical. We chose the zero-shot models which are specialized towards classification because it aligns with our goal and how the data is structured. They were chosen for their properties and popularity, similarly to how the pre-trained SLMs were chosen.

3.2.2 Classification Technique Selection

As discussed in Section 2.6, the embedding models under consideration do not produce a label directly. A separate classification technique is therefore needed to assign labels based on the resulting embeddings. An important part of selecting our models was therefore also selecting a classification technique to pair with them. Through our literature review, we investigated different classification techniques to identify candidates that would be optimal for our task. Which technique we ultimately selected, and why, is presented as part of our literature review findings in Section 4.1.1.

3.3 Model Evaluation Metrics

To be able to determine a model's performance, good evaluation was needed. We therefore conducted research, using the process described in Section 3.1, into which metrics are commonly used to evaluate machine learning classifiers, identifying the metrics described in Section 2.3 as suitable existing metrics for our task. This understanding makes it possible to compare the different models.

As discussed in Section 1.2, however, we considered these existing metrics insufficient on their own, since they treat every misclassification as equally wrong even though some, such as confusing a T-shirt for a tank top, are more reasonable than others. We therefore also designed a complementary set of domain-specific metrics, inspired by Nearest Neighbor classification, intended to assess how well an embedding model clusters semantically similar products and to identify which misclassifications are likely to be critical. The full design and definition of these metrics, together with the resulting set of metrics chosen for our experiments, is presented in Section 4.1.3.

3.4 Experiments

With models and metrics identified, we designed three experiments to evaluate the performance of SLMs in classifying products within fashion. One using pre-trained SLMs, one using the same models after fine-tuning, and one using zero-shot models. As these experiments share a lot of overlap, this section first describes the experimental design and setup shared by all three, along with their individual differences and motivations (Section 3.4.1). Then we describe how the experiments were run and what data was collected. Finally, we describe how the resulting data was analyzed.

3.4.1 Experimental Design and Setup

This subsection covers the task shared by all three experiments, the tools and hardware used, how the dataset was prepared, and the specific design of each experiment.

Task Definition

All experiments share the same task. Every model should, provided the Swedish title and description of a product, output a label predicting its product type. An example of how this task looks can be seen in Figure 3.2.

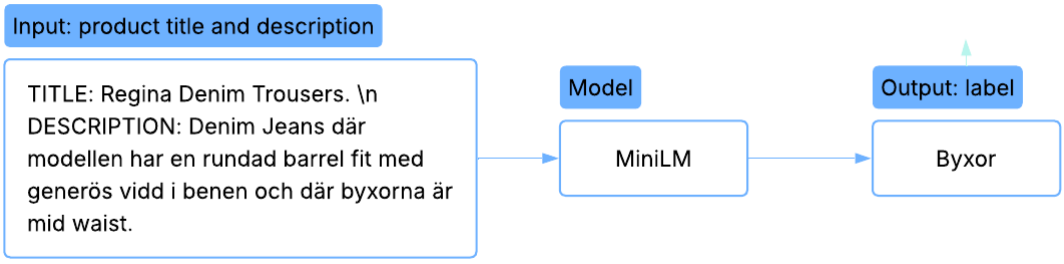

Figure 3.2: Example of product input and classification.

To achieve answers for the research questions, the number of models tested were limited. For the pre-trained SLMs we later fine-tuned, we limited the number to three candidates. For the zero-shot classification models the goal was to see a difference. To attain this, three zero-shot classification models were chosen. Additionally, the number of labels a product had was limited to one considering there are some products where more than one label could be applied. For example, 'top' and 'T-shirt' could both be applicable for one item, even though they are seen as different labels in this thesis.

Tooling: Hugging Face

This thesis utilizes pre-existing models, we will therefore be using Hugging Face to download and fine-tune pre-trained models. Our experiments will be implemented using the Hugging Face [32] library for Python together with Jupyter notebooks, scikit, numpy and PyTorch. This will allow us to construct scripts and notebooks supporting easy interchangeability between models allowing us to automate the training and testing processes.

System Configuration

To ensure that all the results were comparable, all the tests and experiments regarding the hardware usage were run on the same computer. The computer used has an AMD Ryzen AI MAX+ PRO 395 processor with a base clock speed of 3.00 GHz with an integrated Radeon 8060S GPU. The installed RAM was 64.0 GB of LPDDR5x, shared between the CPU and GPU and operated on a 64-bit Windows operating system with an x64-based processor architecture. The OS version used was 24H2 with the Windows 11 Business edition. Because PyTorch uses the NVIDIA platform CUDA, which does not support AMD GPUs, we used the DirectML plugin to access the GPU in the tests [33]. To ensure that DirectML was usable, a virtual environment was created using Python 3.11.9.

Hardware Usage Measurement

As discussed in Section 2.3, we measured CPU time, GPU usage and the process's RSS to assess the hardware demands of each model. Which stage of the process these measurements correspond to differs between our experiments. For Experiments 1 and 2, the same three embedding models are evaluated first without fine-tuning and then after fine-tuning. Since fine-tuning is the computationally expensive step, and our interest lies in understanding the

resources required to adapt a pre-trained model to our task, we measured hardware usage only during fine-tuning itself, not during inference. The hardware values reported alongside our performance metrics in Tables 4.4-4.8 therefore reflect the cost of fine-tuning a given model on a given dataset. In contrast, the zero-shot models evaluated in Experiment 3 and shown in Table 4.9 are not fine-tuned. For these models, the hardware measurements we report instead reflect the cost of inference, since that is the only computational cost such a model incurs. To measure CPU usage, we used a background thread that periodically sampled CPU utilization throughout each run, from which we estimated the total CPU time consumed by the process. For peak memory usage, we used the *psutil* Python library to monitor the process's RSS throughout each run. GPU utilization and memory usage were similarly monitored throughout each run using a dedicated Python GPU-monitoring library, from which we computed the average and peak values reported in this thesis.

Dataset and Preprocessing

The data we have received consists of product catalogs from 6 different fashion retailers in the form of XML files. A product catalog contains information such as image URLs, product category and price. As stated earlier, we are only concerned with the title and description of the products. The chosen focus was on Swedish products, therefore the title and description from the Swedish catalog were extracted along with the product category. The product category was used as the label for the models to predict. The extraction is performed with the Python library Beautiful Soup and the output is stored in a serialized output file by using the serialization library pickle.

Prior to training the models on the data, it needed to be cleaned. Although most retailers primarily sell fashion products, they can still carry items in alternative categories such as home decorations, beauty products and electronics. We filter out all categories that do not pertain to wearable fashion. The products that lacked either title or description or both were also filtered out. The data was then normalized by converting HTML entities to their represented characters and removing any formatting errors before being serialized to a new file with pickle.

We decided to create two datasets from two different retailers' catalog. Due to a confidentiality agreement with Voyado, we cannot disclose the identities of these retailers or share the underlying data files. We will therefore refer to them anonymously as Customer A and Customer B. The sizes of these two customers individually can be seen in Table 3.1. We believe that using datasets from two different customers will give a more realistic idea on how the models would perform in reality.

Data Name	Data Size
Customer A	907
Customer B	1703

Table 3.1: Sizes of the two individual datasets.

A key objective is for the model to generalize so it can be used to classify fashion products for

multiple different retailers with different formats for their descriptions and titles. We therefore also construct general datasets in the different sizes XSmall, Small, Medium, Large and XLarge. Their specific data sizes can be seen in Table 3.2. They were created by combining the filtered data from each retailer while retaining the class label proportions to minimize the risk of overfitting.

Data Name	Data Size
XSmall	1621
Small	3243
Medium	9739
Large	16228
XLarge	32459

Table 3.2: Sizes of the 6 retailers combined datasets.

Formatting the Data

Formatting the data prior to processing is of significant importance. In information retrieval, data is commonly represented as query and passage pairs (q, p). These pairs can be seen as questions and answers of forums which the models used when creating their embeddings. In our case, the title and description of an item can be seen as the question, whilst the label can be seen as the answer. This resulted in the description becoming the passage and the label becoming the query.

For the fine-tuned models the split between training and testing was 70% training and 30% testing and was shuffled with a random state of 42. A consistent random state ensures that the model's shuffle will stay consistent between runs whilst also making sure the inputs to the model were shuffled.

Embedding and Classifying the Data

For experiment 1 and 2 we used embedding models and paired them with a suitable classifier based on the findings in Section 4.1.1. To produce the embeddings for a given description, we called on the model's encode function. In the encoding, we chose to normalize the vectors. The reason for normalizing is to ensure that the values stay in a reasonable range. Skipping the normalization step may cause the calculations to overflow and potentially take unnecessary computational power.

After producing the embeddings we feed the embeddings into a classifier head provided by the Hugging Face API and produce the prediction by running the classifiers *predict()* function.

Running Experiments and Collecting Data

To obtain performance metrics for the chosen models, the testing data was utilized. To be able to use embedding models for classification a classification layer with a specific classifi-

cation technique is needed. To evaluate how well the classifier and chosen embedding model performed together, the classifier was used to predict which labels corresponded to the test embeddings created earlier. This resulted in an array which was used to calculate the performance metrics and facilitate model evaluation.

To ensure the evaluation metrics provided a more reliable estimate of the model and to avoid results dependent on a single split, we repeated the process over 10 folds. In each fold, *StratifiedKFold* was used along with shuffling of the data once again, with the random state 42. In the first folds, we saved the data in order to test it on our own domain-specific metrics, described in Section 3.3.

Analysing Experiment Data

For each fold, we computed the standard performance metrics described in Section 2.3, averaged across all 10 folds to obtain a reliable estimate of each model's performance. We additionally applied our custom evaluation metrics to the data saved from the first folds, as described in Section 4.1.3, to assess each model's embedding clustering quality and to identify a set of likely critical misclassifications. For our final, optimal models, we manually reviewed a random sample of these critical misclassifications to assess their real-world severity; this process and its results are presented in Section 4.3. The full results of this analysis are presented in Chapter 4.

3.4.2 Experiment 1: Pre-Trained SLMs

Our first experiment concerned itself with how well the pre-trained SLMs we chose perform without any fine-tuning. These tests were only run on the datasets Customer A and B. This was done on the datasets both independently and combined. We do this since the models are not trained on our own data, we would therefore not see a significant performance increase from using one of the aggregated larger datasets. Having being already pre-trained on large external datasets, these models are capable of making predictions without any task-specific fine-tuning.

3.4.3 Experiment 2: Fine-Tuning

When fine-tuning the models the data that we use suddenly becomes much more important. Because of this we wanted to perform multiple different measurements to give a clear idea on how these models work in reality. Since fine-tuning is computationally expensive but a necessary step for adapting a pre-trained model to our task, we measured CPU time, GPU usage, and peak memory consumption specifically during this fine-tuning process. The hardware measurements reported alongside our performance metrics in Tables 4.4-4.8 therefore reflect the cost of fine-tuning a model on a given dataset, and do not include hardware usage during inference or evaluation. First we trained and ran the model on the same datasets as Experiment 1 to be able to compare the improvements of fine-tuning. Afterwards we trained the model on first Customer A and then Customer B and tested it on the combined dataset of A and B. Then we did the same thing but the other way round. Doing this will give a better idea on the impact of adding new customers to the model and whether or not the order

they are added in matters. Finally, we trained and ran the models on the Medium and Xlarge datasets to give measurements on how the models would perform in a larger scale scenario where the same model is used for multiple retailers.

Fine-tuning the pre-trained models meant that a balance between improving their performance and never overfitting on the data was important. For the fine-tuning we chose to use the Hugging Face API SentenceTransformerTrainer [34]. For the larger inputs, hard negatives were added to improve the models [35]. We did test it for some of the smaller inputs, but we could see neither improvement nor a decrease in performance. Therefore we chose not to have hard negatives when doing the tests for the smaller inputs. To validate the training loss, we used MultipleNegativesRankingLoss [36]. Our datasets without hard negatives rewarded positive pairs being close and with hard negatives also rewarded negative pairs being distant.

The chosen hyperparameters that were used reflected on both the data along with the computer used for computations. For the larger inputs, we chose to run 3 epochs whilst for the smaller ones we chose to run 5 epochs. In the larger models the learning stagnated at 3 epochs and having more epochs would consequently lead to the risk of overfitting the data. The value of the *per_device_train_batch_size* was 16. This value works best to create a balance between hardware and memory requirements for the computer which was used. A larger value would be difficult for the computer used and a lower value would heighten the risk of allowing noise to affect the results. The *warmup* ratio chosen was 10%, which makes sure that the model will learn but that the weights will not be drastically changed before the optimizer is able to adapt to the new data. Because the computer used does not support *bf16*, we chose to use *fp16* [37] [38]. To ensure that the model neither overfitted nor underfitted, the values of the training and validation losses were regularly printed during the fine-tuning's training stage.

3.4.4 Experiment 3: Zero-Shot Classification

The third experiment investigates whether zero-shot models are a viable option to use. If these models are viable, this would have great implications for parties who may lack a suitable dataset for fine-tuning. As zero-shot models require no fine-tuning step, the hardware measurements reported for these models reflect inference cost only, rather than the fine-tuning cost reported for the models in Experiments 1 and 2. For the zero-shot models we only used the dataset for Customer A. Since zero-shot classification requires scoring each input against every candidate label, we expected inference to be considerably slower per item than for our embedding models. Given the limited time available for this thesis, we therefore restricted zero-shot testing to Customer A, our smaller dataset, rather than the full set of datasets used in Experiments 1 and 2.

Chapter 4
Results

The results from experiment 1 and 2 overlap and will therefore be presented together. Experiment 3 will have the results in its own section.

4.1 Literature Review

4.1.1 Chosen models

Pre-Trained SLMs and classification layer

Following our literature review in Section 3.2, we selected the three pre-trained embedding models shown in Table 4.1 for their strong reported performance and fit for our task. All three are developed by Microsoft. The largest, *intfloat/multilingual-e5-small*, is the only one trained for multilingual use, while the other two are trained only on English data.

Model	Number of parameters
intfloat/small-e5-v2	33.4M
intfloat/multilingual-e5-small	117M
sentence-transformers/all-MiniLM-L6-v2	22.7M

Table 4.1: Number of parameters for the chosen embedding models.

All of these 3 models are embedding models. To be able to use the embedding models for classification a classification layer with a classification technique was attached to the embedding models. These classification techniques are used on a model's embedding to make predictions. When making the classification, the way in which it is made can be important

because it is the last layer before the models output and can therefore affect the model's perceived performance. This is why, when looking at the different classification techniques that can be found, we wanted to see which one would fit the models end goal.
To evaluate different models against one another, we chose a classifier which we wanted to test on the model's embeddings. After looking over the classifiers described in Section 2.6, we chose to use SVM. This choice was based SVMs efficiency on larger datasets, whereas KNN becomes costly for the larger datasets that created high-dimensional spaces.

Zero-Shot Classification models

After analyzing the literature existing for the zero-shot classification we chose to select the models in Table 4.2. The RoBERTa and BART models are both developed by Facebook, whilst the DeBERTa model is developed by Microsoft. The DeBERTa model is the only multilingual model out of these and was therefore chosen despite not being as popular as the other two models we chose, that are trained on only English.

Model	Number of parameters
facebook/bart-large-mnli	0.4B
roberta-large-mnli	0.4B
MoritzLaurer/mDeBERTa-v3-base-mnli-xnli	0.3B

Table 4.2: Number of parameters for the chosen zero-shot classification models.

4.1.2 Existing performance metrics

To evaluate the models, we used the metrics calculated from the 10 folds: accuracy, precision, recall, and F1-score, as defined in Section 2.3. These are reported in Section 4.2 alongside the hardware usage and our custom design metrics, introduced next.

4.1.3 Custom Design Metrics

Embedding Space and Classifiers

When producing embeddings for an array of input you often refer to the embeddings existing in an embedding space. Figure 4.2 shows a simple 2-dimensional visual example of how the embedding space might look. Each data point is labeled with its true label. Generally, most classifiers, such as KNN and SVM, benefit from common labels being clustered closely together. The idea is that closeness in the multi-dimensional space signifies similarities in the data points. A good embedding model should provide well-clustered embeddings to facilitate more accurate classification.

We want to use our embedding metrics to measure two things. The first aspect concerns the quality of the model's clustering. If the prediction accuracy of our model is low but the

embedding model is clustering well it suggests improvements should be made to the classification head rather than the embedding model. Otherwise, if the clustering of the embedding model is poor, it suggests that improvements should instead be made to the embedding model.

The second thing we want to measure is the criticality of our misclassifications. A product that is far away from its cluster is critically misclassified, meaning that it is likely that the model is significantly incorrect in its classification. In the following sections we will describe two metrics which we will use to evaluate our embedding models' clustering capabilities and then combine them to get a set of critical misclassifications.

The following metrics are based on looking at Nearest Neighbors in the embedding space. As written in Section 2.6, this is quite costly for larger datasets. Since these metrics are only used for testing we can use them here but it would be too costly to actually use in within the model during classification. We utilize the scikit-learn function NearestNeighbors to get the Nearest Neighbors of a point efficiently.

For both metrics we divide our test dataset into two equal halves. We call the first one $\boldsymbol{L}_f$ and the second one $\boldsymbol{L}_l$. Each point in both sets contain the title and description $\boldsymbol{x}$, the embedding $\boldsymbol{h}$, the true label y_t, and the label predicted by the model y_p. We then loop through each point in $\boldsymbol{L}_l$ and get its 10 Nearest Neighbors from the embeddings in $\boldsymbol{L}_f$ and call this set $\boldsymbol{N}_i$.

Step Accuracy

Since we are constructing metrics that do not have to produce a classification there is more flexibility to create a more lenient kind of accuracy. It is informative to examine how often the absolute closest neighbors contain the same label as the data point. We introduce an accuracy metric which we call step accuracy.

For step accuracy we iteratively step outwards from the absolute Nearest Neighbor to see how many steps are required before finding a neighbor with the same label. Looping over the points in $\boldsymbol{L}_l$ as described in the previous section, we first look at $\boldsymbol{K = 1}$ and we check whether the data point shares its true label with its absolute closest neighbor. Then $\boldsymbol{K = 2}$, we check if the data point has the same true label as one of its two closest neighbors and so forth.

For this thesis we will measure for $\boldsymbol{K \in 1, 2, 3}$. A low accuracy on $\boldsymbol{K = 3}$ would indicate the model clusters poorly. A significant increase between the steps would suggest poor clustering but the model still places similar items in close proximity. Individually for each point, having three Nearest Neighbors with differing labels may indicate that the point is poorly situated within its cluster.

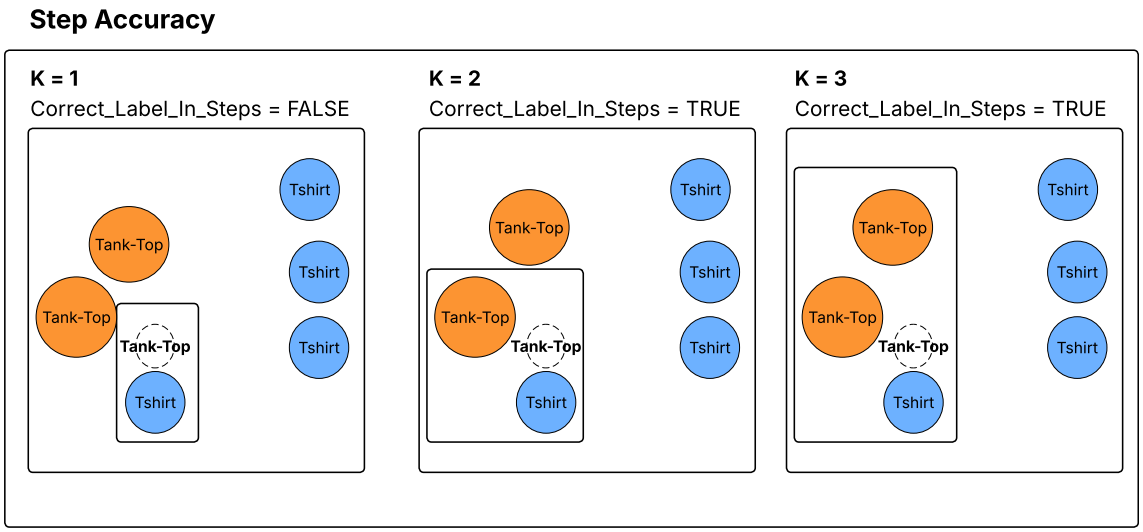

Figure 4.1: Step accuracy with example product.

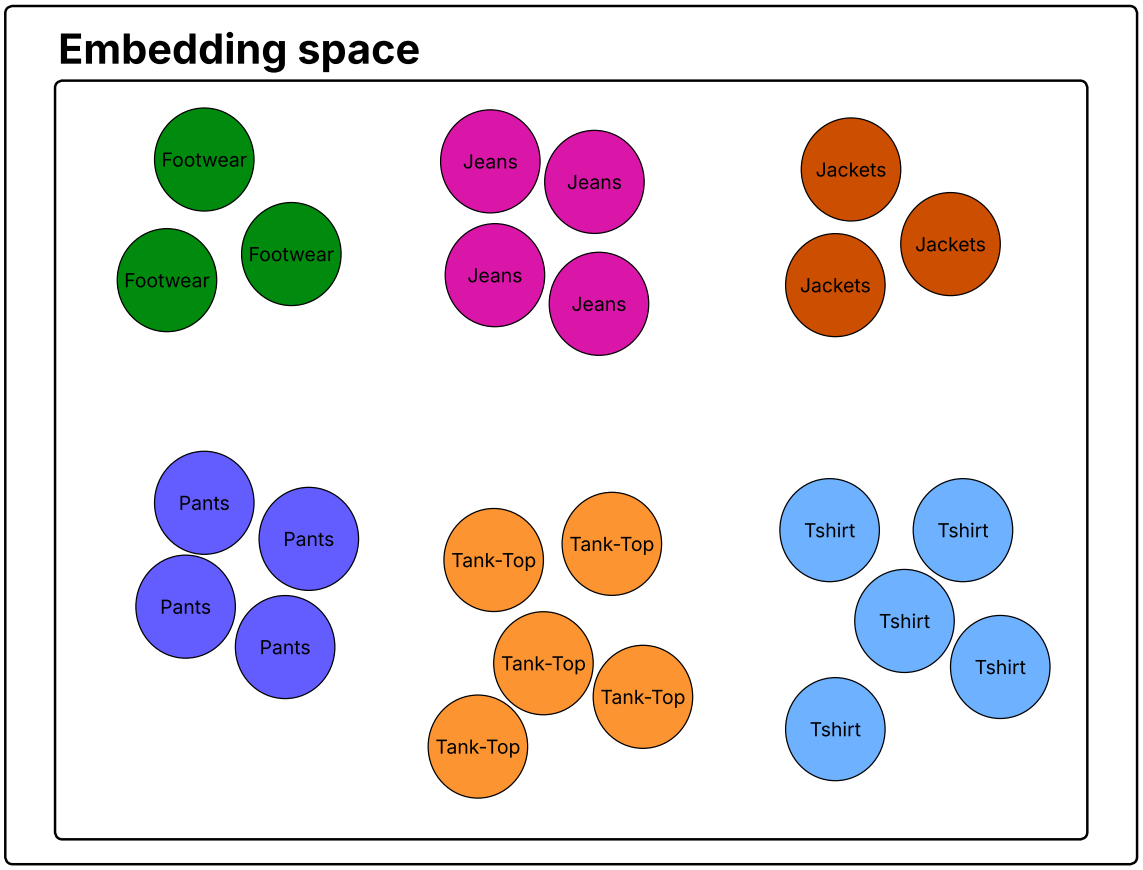

Figure 4.2: Example embedding space.

Nearest Neighbors

The next metric we examine is a more traditional Nearest Neighbor classification. Looking at N_i, we identify what the most popular label is. If the most popular label is the same as the

data point's it gives a hit. The accuracy measures the percentage of points in l_i that gets a hit. A high Nearest Neighbors accuracy should suggest that most inputs are being embedded close to each other in close clusters.

To further give a better insight of our models clustering skills we also deem it worthwhile to measure the Nearest Neighbor accuracy for when the most popular label is a majority. Since we only look at the 10 nearest neighbors that would mean this is the accuracy for all cases where more than 5 of the neighbors true label is the same as the most popular label. The idea behind using this metric is that a big jump between the normal Nearest Neighbor metric and the majority one could point to that there are some labels that are being confused with one another and are therefore clustered together.

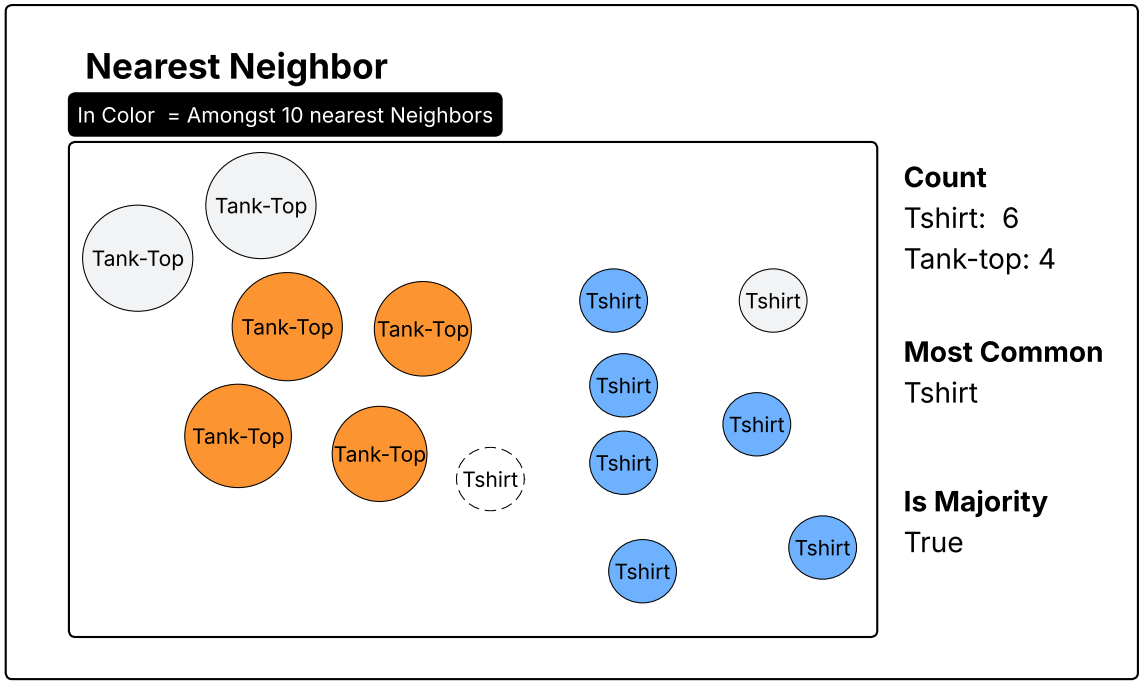

Figure 4.3: Nearest Neighbor example product.

Critical Misclassification

In the two previous subsections we described how a failure to meet the criteria of our metrics could suggest that a misclassification is critical. In Figure 4.4, 4.5, and 4.6 we can see two instances of a close classification and one critical misclassification. Notably, for case 1, the data point does not give a satisfying result for the step accuracy but it does for Nearest Neighbor. In case 2, the opposite occurs. In the third example, however, we see a critical misclassification. The embedding is a real outlier and we can see that here the data point does not satisfy either the step accuracy or Nearest Neighbor criteria. We construct our set of critical misclassifications with each data point that does not satisfy either step accuracy or Nearest Neighbors and the predicted label is not the same as the true label.

It is worth considering whether these critical misclassifications are entirely incorrect or

whether some ambiguity remains. Furthermore, it is interesting to examine if the misclassifications are incorrect to a degree that a customer could potentially be bothered.

We would like to review every one of them but there is a high likelihood that, due to the size of our dataset, there are too many products to manually go through. To circumvent this we will sample 10 different random critical misclassifications from each model. We go through each misclassification and conclude if we think it is a serviceable classification or not. The model then receives a score of x/10, where x is the number of classifications that we regard to be still serviceable. We give a short summary of each classification which we regard to not be serviceable and provide the reader with a better insight on how the model performs in the worst case scenarios. Since this is a manual and very time-consuming task we only provide these scores for the final run with our optimal models.

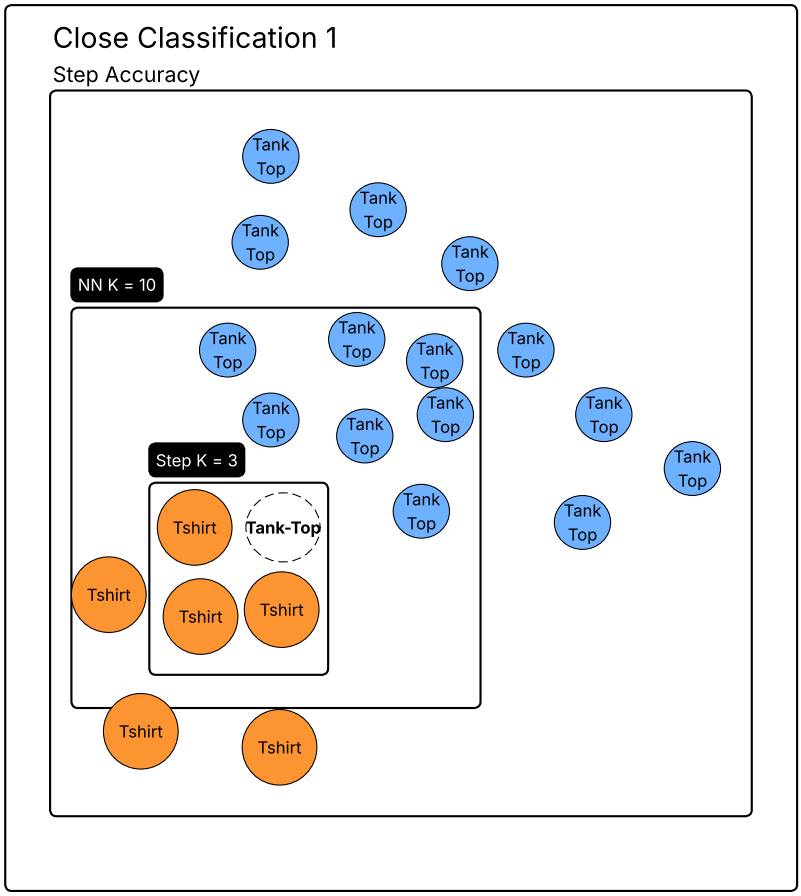

Figure 4.4: Close classification example 1.

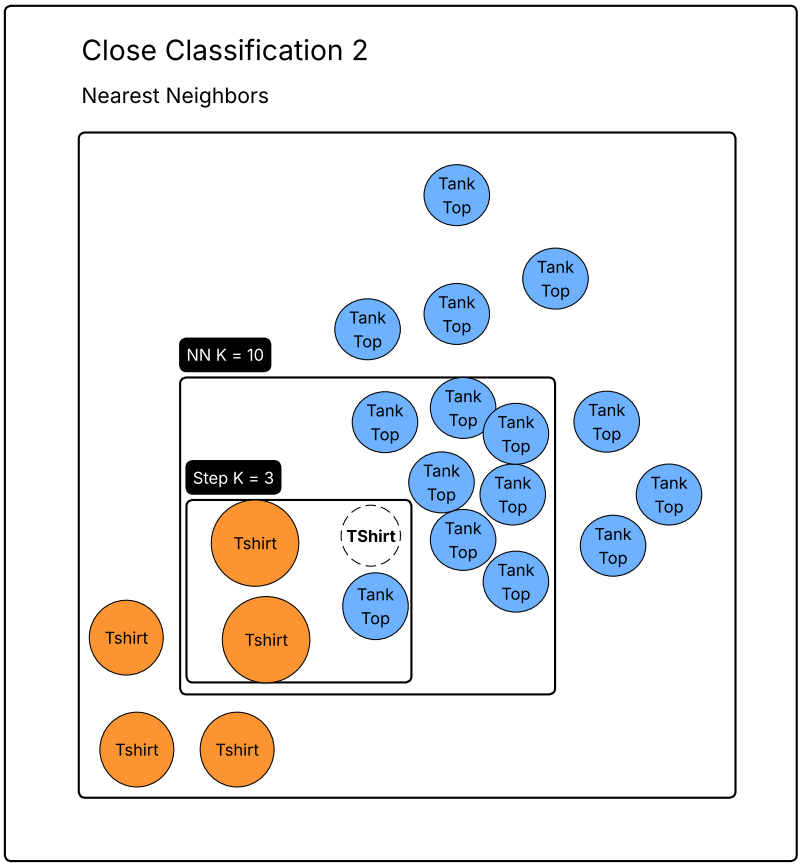

Figure 4.5: Close classification example 2.

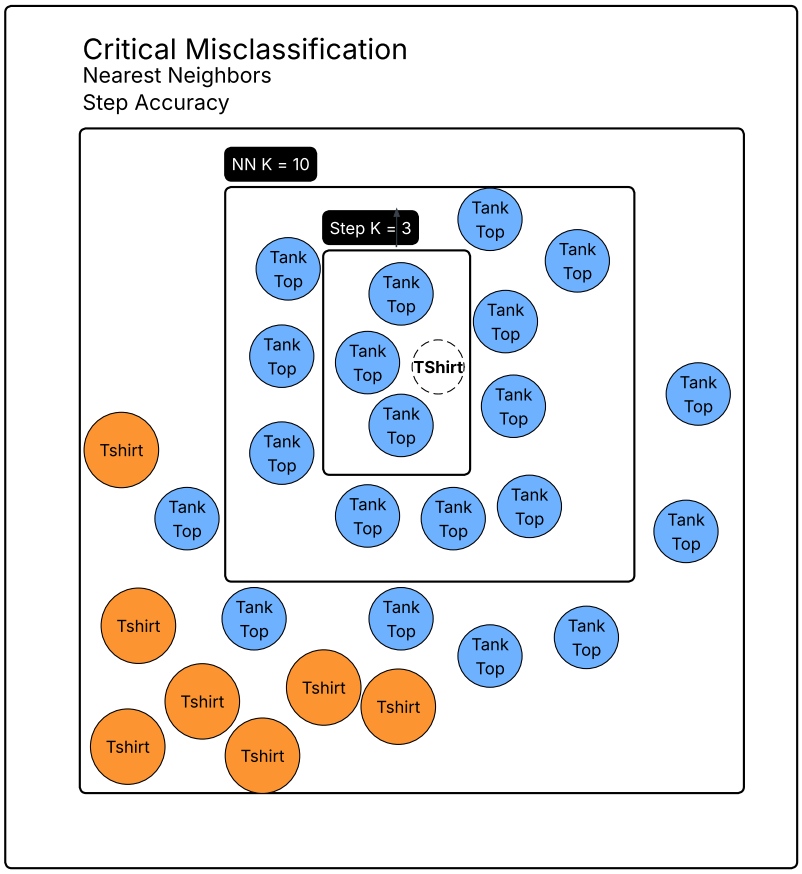

Figure 4.6: Critical misclassification example.

4.2 Experiments

4.2.1 Evaluation of Pre-Trained and Fine-Tuned Embedding Models

We evaluate our three pre-trained embedding models across several dataset combinations to assess both their raw performance and how that performance is affected by fine-tuning, dataset size, and the order in which retailers' data is introduced. Specifically, Table 4.3 reports performance on Customer A and Customer B individually, whilst Table 4.4 reports performance on the two combined. Tables 4.5 and 4.6 report performance when the model is fine-tuned on one customer and then the other, in each of the two possible orders. Tables 4.7 and 4.8 report performance on the larger, multi-retailer Medium and XLarge datasets described in Section 3.4.1. Throughout this section, NN refers to the Nearest Neighbor metric and NN (Majority) to the Nearest Neighbor majority metric, both defined in Section 4.1.3. RSS refers to Resident Set Size, meaning the amount of memory actively held in RAM. All the existing performance metric values, described in Section 2.3 are averaged across the 10 stratified folds described in Section 3.4.1.

The hardware usages in the Tables 4.4-4.8 are from the fine-tuning of the models. The best

performing model for each metric is highlighted and the average score from the existing metrics is calculated based on the mean of all metrics.

In Tables 4.3-4.8 the results from Experiment 1 and 2 are shown, with the different models and with different data as input. The parenthesis shows how the models perform prior to fine-tuning, where bold lettering displays the best result and underlined bold values display the best performance after fine-tuning in regards to a metric.

Table 4.3 shows how the models perform on on each of the two customers individually. Table 4.4 shows how the models perform when the combined data of the customers is used as input for the models.

All three models perform consistently better on Customer A than on Customer B, both before and after fine-tuning (Table 4.3), despite Customer B's larger dataset (Table 3.1). Before fine-tuning, MiniLM clearly outperforms both E5 variants on both customers, but this gap closes almost entirely after fine-tuning, with all three models reaching average scores within 2.4 percentage points of each other on each customer. The picture is noticeably different for hardware usage. The E5-Small model required the least CPU time to fine-tune on both customers, while MiniLM, despite having the fewest parameters of the three models, seen in Table 4.1, required the most CPU time on Customer A and the second-most on Customer B. E5-Multi consistently required the most peak memory usage of the three models on both customers, consistent with it being the largest of the three by parameter count.

On the combined dataset (Table 4.4), average scores for all three models fall between their respective scores on Customer A and Customer B alone, closer to the Customer B scores, roughly in proportion to Customer B making up the larger share of the combined data. CPU time on the combined dataset exceeds the sum of the individual customers' CPU times for all three models, most notably for MiniLM, which now requires more than three times the CPU time of E5-Small.

	Customer A			Customer B		
	E5-Small	E5-Multi	MiniLM	E5-Small	E5-Multi	MiniLM
Accuracy (%)	95.6 (73.1)	96.4 (78.1)	**96 (89.5)**	89.1 (70.4)	88.8 (73.2)	**89.4 (81.7)**
Precision (%)	85.9 (53.6)	87.7 (60.2)	**89.8 (82.3)**	70 (41.1)	**72.4** (40.3)	71 (**57.1**)
Recall (%)	87.9 (50.5)	89.3 (56.4)	**91 (81.1)**	73.1 (38)	**74.9** (38.8)	73.1 (**56.5**)
F1-score (%)	86.4 (50)	**88.1** (56.5)	88 (**80.6**)	70.6 (37.7)	**72.4** (38)	70.8 (**55.5**)
Average Score (%)	89 (56.8)	90.4 (62.8)	**91.2 (83.4)**	75.7 (46.8)	**77.1** (47.6)	76.1 (**62.7**)
step accuracies	[0.74, 0.763, 0.763]	[0.71, 0.763, 0.763]	[0.725, 0.74, 0.74]	[0.725, 0.748, 0.756]	[0.702, 0.771, 0.779]	[0.725, 0.748, 0.756]
NN:	0.656	0.656	0.603	0.595	0.679	0.595
NN (Majority):	0.748	0.74	0.692	0.709	0.736	0.709
CPU time (h)	**0.77**	0.87	3.53	**2.07**	5.8	4.63
Peak RSS memory (GB)	2.83	5.5	**2.7**	**2.84**	5.75	3.01
Average GPU usage (%)	4.5	**2.9**	3.2	3.5	**2.2**	3.2
Peak GPU usage (%)	17.8	11.9	**5.3**	11.8	**4.0**	5.6

Table 4.3: Metrics for Customer A and Customer B from the embedded models. In parenthesis the values before fine-tuning are displayed.

Note: E5-Small = intfloat/small-e5-v2; E5-Multi = intfloat/multilingual-e5-small; MiniLM = sentence-transformers/all-MiniLM-L6-v2.

NN and NN (Majority) refer to the Nearest Neighbor and Nearest Neighbor majority metrics defined in Section 4.1.3. RSS refers to Resident Set Size, the amount of memory actively held in RAM. All values are averaged across 10 stratified folds (Section 3.4.1). Values in parentheses show model performance before fine-tuning. Bold, underlined values indicate the best-performing model for that row.

	E5-Small	E5-Multi	MiniLM
Accuracy (%)	**91.2** (75.6)	91.1 (77.6)	91.1 (**83.6**)
Precision (%)	**76.9** (45.5)	76.8 (46.8)	76.1 (**65.1**)
Recall (%)	76.7 (44.3)	**77** (45.9)	75.9 (**63.2**)
F1-score (%)	75.6 (43.3)	**75.9** (44.7)	74.8 (**62.8**)
Average Score (%)	80.1 (52.2)	**80.2** (53.8)	79.5 (**68.7**)
step accuracies	[0.725, 0.763, 0.763]	[0.771, 0.786, 0.802]	[0.71, 0.74, 0.763]
NN:	0.656	0.649	0.664
NN (Majority):	0.712	0.738	0.765
CPU time (h)	**4.63**	9.92	12.27
Peak RSS memory (GB)	**2.75**	5.8	2.9
Average GPU usage (%)	**1.9**	2.5	2.0
Peak GPU usage (%)	13.1	18.8	**4.6**

Table 4.4: Metrics for Customer A & B from the embedded models. In parenthesis the values before fine-tuning are displayed.

Note: E5-Small = intfloat/small-e5-v2; E5-Multi = intfloat/multilingual-e5-small; MiniLM = sentence-transformers/all-MiniLM-L6-v2.

See Table 4.3 for definitions of NN, NN (Majority), RSS, and notation conventions.

Tables 4.5 and 4.6 present the results from fine-tuning on a pre-trained SLM on customers one at a time, where a model is trained on an initial customer and subsequently fine-tuned on a second customer. Table 4.5 show the metric results from having the models first fine-tuned on customer A and later on customer B. Contrary to this Table 4.6 displays the models results when they are fine-tuned on customer B and then customer A. As motivated in Section 3.4.3, this sequential fine-tuning setup allows us to assess the effect of onboarding a new customer onto an already fine-tuned model, and whether the order in which customers are introduced matters.

Comparing Tables 4.5 and 4.6 to the single-run combined result in Table 4.4, fine-tuning on Customer A followed by Customer B (Table 4.5) yields average scores within 1 percentage point of training on the combined dataset directly. Fine-tuning on Customer B followed by Customer A (Table 4.6), however, yields average scores roughly 2 percentage points lower than either alternative for all three models. Here, the hardware usage was measured and combined from both fine-tunings. The CPU time was consistently higher across all three models when Customer B was fine-tuned on first, which can be seen in Table 4.6, than when Customer A was fine-tuned on first, seen in Table 4.5.

	E5-Small	E5-Multi	MiniLM
Accuracy (%)	**90.7**	90.5	89.9
Precision (%)	76.1	**76.6**	76.5
Recall (%)	75.8	**77.1**	75.7
F1-score (%)	74.6	**75.7**	74.7
Average Score (%)	79.3	**80**	79.2
step accuracies	[0.698, 0.826, 0.849]	[0.791, 0.849, 0.86]	[0.733, 0.802, 0.849]
NN:	0.709	0.651	0.628
NN (Majority):	0.833	0.793	0.745
CPU time (h)	**3.91**	7.21	4.8
Peak RSS memory (GB)	2.91	5.5	**2.7**
Average GPU usage (%)	3.94	**2.64**	3.33
Peak GPU usage (%)	19.6	11.9	**6.7**

Table 4.5: Model first fine-tuned on Customer A, later being fine-tuned on customer B.

	E5-Small	E5-Multi	MiniLM
Accuracy (%)	**89**	88.4	88.3
Precision (%)	74.9	74.7	**75.2**
Recall (%)	74.4	**75**	74.2
F1-score (%)	73	**73.4**	73.1
Average Score (%)	77.8	**77.9**	77.7
step accuracies	[0.87, 0.87, 0.87]	[0.826, 0.87, 0.87]	[0.826, 0.826, 0.87]
NN:	0.435	0.5	0.435
NN (Majority):	0.531	0.515	0.667
CPU time (h)	9.96	14.5	**7.17**
Peak RSS memory (GB)	**2.86**	5.75	3.01
Average GPU usage (%)	2.63	**2.44**	3.06
Peak GPU usage (%)	11.8	**4.9**	5.6

Table 4.6: Model first fine-tuned on Customer B, later being fine-tuned on customer A.

Note: E5-Small = intfloat/small-e5-v2; E5-Multi = intfloat/multilingual-e5-small; MiniLM = sentence-transformers/all-MiniLM-L6-v2.
See Table 4.3 for definitions of NN, NN (Majority), RSS, and notation conventions.

In Table 4.7 we see how the chosen models perform on the Medium dataset and in Table 4.8 we see their performance on the Xlarge dataset, which comprises all available data after filtering.

Average scores improve for all three models from the Medium to the XLarge dataset (Tables 4.7, 4.8), most notably for MiniLM, whose average score improves by 4.4 percentage points compared to 1.4 and 0.9 points for E5-Small and E5-Multi respectively. On the Medium dataset, MiniLM also stands out on the clustering-based metrics, reaching the highest NN and NN (Majority) scores of the three models despite having the lowest average score on the standard metrics. Following the reasoning in Section 4.1.3, this divergence suggests that MiniLM's embeddings cluster comparatively well even where its classification head performs comparatively poorly.

Hardware usage does not scale consistently with dataset size between Medium and XLarge. CPU time is lower on the larger XLarge dataset than on Medium for all three models, most noticeably for MiniLM, whose CPU time nearly halves. Average GPU usage, conversely, increases for all three models from Medium to XLarge, while peak RSS memory and peak GPU usage change in different directions for different models. We do not have a clear explanation for this and return to it in Section 5.2.

	E5-Small	E5-Multi	MiniLM
Accuracy (%)	91.5	**92.1**	91.3
Precision (%)	**72.4**	71.7	67.4
Recall (%)	69.1	**69.2**	64.7
F1-score (%)	68.8	**69.4**	65.1
Average Score (%)	75.5	**75.6**	72.1
step accuracies	[0.813, 0.885, 0.92]	[0.809, 0.901, 0.924]	[0.821, 0.899, 0.938]
NN:	0.797	0.803	0.854
NN (Majority):	0.848	0.846	0.885
CPU time (h)	138.01	**112.16**	120.39
Peak RSS memory (GB)	3.25	3.78	**3.16**
Average GPU usage (%)	2.3	2.5	**2.1**
Peak GPU usage (%)	**5.1**	25.0	**5.1**

Table 4.7: The models trained on the Medium dataset.

	E5-Small	E5-Multi	MiniLM
Accuracy (%)	93.8	**94.1**	93.4
Precision (%)	73.5	73.1	**73.7**
Recall (%)	**69.7**	69	68.9
F1-score (%)	**70.5**	69.8	69.8
Average Score (%)	**76.9**	76.5	76.5
step accuracies	[0.917, 0.957, 0.966]	[0.914, 0.954, 0.971]	[0.906, 0.943, 0.963]
NN:	0.929	0.929	0.92
NN (Majority):	0.935	0.94	0.932
CPU time (h)	118.92	98.11	**65.51**
Peak RSS memory (GB)	2.55	4.19	**2.26**
Average GPU usage (%)	4.0	3.4	**3.3**
Peak GPU usage (%)	11.3	**5.0**	15.7

Table 4.8: The models trained on the Xlarge dataset.

Note: E5-Small = intfloat/small-e5-v2; E5-Multi = intfloat/multilingual-e5-small; MiniLM = sentence-transformers/all-MiniLM-L6-v2.

See Table 4.3 for definitions of NN, NN (Majority), RSS, and notation conventions.

4.2.2 Evaluation of Zero-Shot Classification Models

The implementation of using the zero-shot classification models required less code than that of the pre-trained SLMs. We began by creating a zero-shot classification pipeline, specifying the model used. Then the data was loaded and duplicated labels were removed, because duplicates would require more run time without improved performance. Zero-shot models were given an input and a list of possible answers. Given an input, the model outputs a probability for each label. We selected the label with the highest probability. This could then be used to calculate how well it performed based on existing metrics we had chosen previously to evaluate our models. Table 4.9 presents the result from running the chosen zero-shot classification models on customer A and the hardware usage of it.

	BART	RoBERTa	MoritzLaurer
Accuracy (%)	16.4	16.5	**34.1**
Precision (%)	31.4	23.2	**40.3**
Recall (%)	18.6	20.2	**31.7**
F1-score (%)	17.3	17.3	**31.4**
Average Score (%)	21	19.3	**34.4**
CPU time (h)	**14.24**	24.31	36.28
Peak RSS memory (GB)	2.02	1.83	**0.88**
Average GPU usage (%)	0.9	**0.1**	4.0
Peak GPU usage (%)	**11.1**	19.2	17.4

Table 4.9: Metrics for Customer A from the zero-shot classification models.

Note: BART = facebook/bart-large-mnli; Roberta = roberta-large-mnli; MoritzLaurer = MoritzLaurer/mDeBERTa-v3-base-mnli-xnli

RSS refers to Resident Set Size, the amount of memory actively held in RAM. Bold, underlined values indicate the best-performing model for that row.

Of the three zero-shot models, MoritzLaurer clearly outperforms BART and RoBERTa on every standard metric, which we can see in Table 4.9. However, it also requires the most CPU time of the three, roughly 1.5 and 2.5 times that of RoBERTa and BART respectively. Even the strongest zero-shot model performs substantially worse than any of the three pre-trained embedding models on Customer A before any fine-tuning was applied (Table 4.3), with an average score roughly 22 percentage points lower than even the weakest pre-trained model's pre-fine-tuning score.

4.3 Misclassification Review

The following are the scores for each model, as described in Section 4.1.3. The final run is on the XLarge dataset, corresponding to Table 4.8. The product type labels referenced below

originate from each retailer's own product category taxonomy (Section 3.4.1). We did not impose a unified taxonomy across retailers beyond filtering out non-fashion categories. Following each score is a list of summarized descriptions of each unserviceable misclassification.

4.3.1 E5 Small

Score: 8/10

- Shirt for swimming, Prediction: T-shirt, True: Swimwear
- Knitted shirt, Prediction: T-shirt, True: Cardigan & Shirts

4.3.2 E5 Multi

Score: 7/10

- Shirt for swimming, Prediction: T-shirt, True: Swimwear
- Knitted shirt, Prediction: T-shirt, True: Cardigan & Shirts
- Leggings, Prediction: Pants & Jeans, True: Socks & Tights

4.3.3 MiniLM

Score: 7/10

- Children Boots, Prediction: Accessories, True: Footwear
- Scrunchie for hair, Prediction: Shoes, True: Accessories
- Loungewear pants, Prediction: Pants & Jeans, True: Loungewear

Chapter 5
Discussion

This chapter discusses the results presented in Chapter 4 in relations to the research questions introduced in Section 1.3, interpreting what the results imply for the utilization of SLMs in fashion product type classification. Section 5.1 discusses our finding for each research question in turn, after which Section 5.2 reflects on the limitations and threats to validity that should be considered when interpreting these results.

5.1 Research Questions

This chapter discusses our results in relation to each of the five research questions presented in Section 1.3. We will focus on interpreting what the results imply, how confident we are in them and how they relate to current work in the field.

5.1.1 Evaluation Metrics (RQ1)

RQ1: What existing metrics can be used to evaluate a model's performance, such as quality of predictions and hardware usage?
We used accuracy, precision, recall and the F1-score from Section 2.3 because taken together, they present both overall correctness and the balance between false positives and false negatives that accuracy alone can not do. This stands in contrast to to Licardo et al. [4], who only relied on Exact Match Accuracy, arguing that partial correctness leads to functional errors in production. As we argued in Section 1.2, we believe this is a less suitable choice for fashion product type classification, where the boundary between adjacent categories, such as a T-shirt and a tank top, is often genuinely ambiguous even for a human. Therefore, a single exact-match metric can not capture this nuance, which is part of our motivation for the domain-specific metrics discussed under RQ2 (Section 5.1.2).

For hardware, measuring CPU and GPU time alongside peak memory usage let us reason about deployability rather than predictive quality alone. To our knowledge, this is an angle largely missing from related work at this model scale, Licardo et al.[4] measures inference cost but not fine-tuning cost, even though fine-tuning is what enables the performance gains we discuss under RQ5 (Section 5.1.5). The hardware measurements in Tables 4.3-4.8 addresses this gap directly, though, as discussed in Section 5.2, they are tied to the available hardware and limited manual experimentation and should be interpreted with that limitation in mind.

5.1.2 Domain-Specific Evaluation Metrics (RQ2)

RQ2: How can a domain-specific metric be designed to give a better perception of the model's quality of predictions?

By examining the embedding space directly, rather than only the final predicted label, we designed metrics, step accuracy and Nearest Neighbor majority (Section 4.1.3), that capture how well an embedding model clusters semantically similar products, independent of the classifier placed on top of it. This is a separation which is useful in practice because it helps us knowing which part of the classifier, which includes both the embedding and classification head, is problematic. A model can have a low accuracy but good clustering points toward improving the classification head, whereas poor clustering points toward the embedding model itself, is a distinction the standard metrics discussed under RQ1 (Section 5.1.1) can not make on their own.

Using these clustering metrics, we identified the subset of misclassifications that are likely critical, meaning the predicted label is far from the product's true cluster rather than merely confused with a neighboring category. Reviewing a random sample of these for our three final fine-tuned models, in Section 4.3, we found between 7 and 8 of every 10 sampled critical misclassifications to still be serviceable in our judgment, in other words plausible enough that we believe a customer would not be bothered by them. When averaged across the three models' individual scores approximates to 73%. Combining the average miss rate of approximately 6% for these models on the XLarge dataset, seen in Table 4.8, with the average 73% serviceability rate found in our review, we estimate that roughly 1.7% of all classifications are both incorrect and unserviceable, implying an overall acceptable-classification rate of around 98%. We present this as an estimate rather than a precise figure, since it assumes that misclassifications outside our sampled critical set behave similarly to those inside it, an assumption we did not test directly.

Many of the misclassifications we reviewed originated from the XLarge dataset, seen in Table 4.8, which combines several retailers that sometimes use different formulations for similar product categories. Our three final models, E5-Small, E5-Multi, and MiniLM, treated these as entirely separate categories where a human reviewer could plausibly categorize the item as either, or both. We discuss what could be done to better account for this in Section 6.1. Overall, however, our results suggest these models are reliable at producing acceptable classifications, even where a small number of genuinely unsuitable ones remain.

5.1.3 Model Selection (RQ3)

RQ3: What current license free, offline, SLMs are suitable for classification tasks based on popularity and model purpose?
Our selection criteria, popularity on Hugging Face and a model's stated purpose (Section 3.2), were a practical way to navigate a large and fast-growing landscape of open-source SLMs without exhaustively benchmarking every candidate. When looking at the results of the pre-trained embedding models discussed under RQ4 (Section 5.1.4), the reasonable proxy for all three models is that they performed within a few percentage points of one another once fine-tuned, which can be seen in Table 4.3. This suggests that popular, purpose-built embedding models in this size range may be broadly interchangeable for tasks similar to ours.

The picture is less favorable for our zero-shot model selection. *facebook/bart-large-mnli*, *roberta-large-mnli* and *MoritzLaurer/mDeBERTa-v3-base-mnli-xnli* were all popular models explicitly built for classification, yet none performed adequately on our task, as can be seen in Table 4.9. This suggests that popularity and stated purpose alone are insufficient indicators of suitability for a narrow, specific language and domain-specific classification problem such as ours. Furthermore, having exposure to languages similar to the target language, such as the MoritzLaurer model had, appears to matter considerably more than general popularity.

5.1.4 Model Performance (RQ4)

RQ4: How do different license free SLMs perform in the context of classification of fashion products regarding quality of predictions and hardware usage?
All three pre-trained SLMs performed similarly across the standard metrics once fine-tuned, with average scores within roughly two percentage points of each other on every dataset we tested, which can be seen in the Tables 4.3–4.8. Given this similarity in predictive quality, hardware usage became the deciding factor. The MiniLM model required less peak memory on the largest dataset and substantially less CPU time than the multilingual E5 model (Table 4.8). Our results therefore suggest MiniLM offers the most favorable trade-off between performance and resource usage for our task, although the difference in predictive quality between the three models is small enough that this should not be read as one model being categorically superior.

Comparing pre-trained SLMs to the zero-shot models, we found a much larger gap. Before doing any fine-tuning, the embedding models outperformed all three zero-shot models on every metric we tested, which can be seen when comparing Table 4.3 and Table 4.9. The best-performing zero-shot model, *MoritzLaurer/mDeBERTa-v3-base-mnli-xnli*, also required the most CPU time of the three zero-shot models, suggesting that for this task, the added flexibility of zero-shot classification does not translate into a favorable trade-off relative to a small, fine-tuned, or even non-fine-tuned, embedding model.

Our headline accuracy of around 94–96% (Tables 4.3 and 4.8) is not directly comparable to the 99% Exact Match Accuracy that Licardo et al. [4] report for a fine-tuned 1-billion-parameter LLaMA 3.2 model, since the tasks, metrics, and datasets differ substantially. We note the comparison primarily to highlight that, despite using models one to two orders of

magnitude smaller and consumer-grade rather than data center enterprise grade hardware, our results do not appear far behind in absolute terms, which is broadly in line with the claim by Koutsomitropoulos and Skoulidis [5] that SLMs in the 100M-parameter range perform only marginally worse than substantially larger models on classification and embedding tasks.

All three models' scores improved somewhat between the Medium and XLarge datasets, seen in Table 4.7 and Table 4.8, with MiniLM improving the most. Hardware usage, however, did not scale predictably with the size of the datasets. The CPU time was, somewhat counterintuitively, lower on the larger XLarge dataset for all three models, and peak memory usage showed no consistent direction either. We do not have a clear explanation for this and flag it as worth further investigation rather than assuming any general principle about how hardware cost scales with dataset size for these models.

Splitting fine-tuning across customers sequentially (Tables 4.5 and 4.6) consistently performed worse than fine-tuning on the combined dataset directly, seen in Table 4.4 and performed worse still when Customer B preceded Customer A than the reverse. We did not investigate why this asymmetry occurs, but it suggests that, at least for these two retailers, the order in which new data sources are introduced during incremental fine-tuning is not arbitrary. The differences in results are however not massively different, which is relevant for a practitioner considering onboarding a new customer onto an already fine-tuned model.

5.1.5 Effect of Fine-Tuning (RQ5)

RQ5: How can fine-tuning improve the model's performance when applicable?
Fine-tuning produced substantial improvements across all three models on both individual customer datasets, with average scores roughly doubling relative to their pre-trained baselines for Customer B and growing by 30–37 percentage points for Customer A, as can be seen in Table 4.3. This supports our motivation in Section 1.2 that, given access to representative labeled data such as provided to us by Voyado, even comparatively small SLMs (22.7M–117M parameters) can be adapted to a specific domain without the billion-parameter scale used in related work such made by Licardo et al. [4].

Contradictory to what we though, the size of this improvement was not uniform, gains were consistently larger for Customer A than for costumer B, which can be seen in Table 4.3, despite Customer B's larger dataset (Table 3.1). This suggests dataset size alone does not determine how much a model benefits from fine-tuning. The composition or difficulty of the underlying label set could play a role. However, this was not directly investigated.

Fine-tuning was also considerably more resource-intensive than simply running the pre-trained models, particularly on larger datasets (Tables 4.7 and 4.8), where CPU time exceeded 100 hours for several models. This is a practical consideration for any organization considering this approach. Although the resulting model is relatively cheap to run for inference, the one-off cost of fine-tuning on a single commercial laptop, as we did (Section 3.4.1), may not be negligible for a smaller organization, even though it remains far less demanding than the data center hardware used to fine-tune the billion-parameter models evaluated by Licardo et

al. [4]. For the larger combined datasets, we also observed that the smaller MiniLM model required less peak memory than both E5 variants, as can be seen in Table 4.8, consistent with the general expectation that fewer parameters require less memory to fine-tune. For the comparatively small individual customer datasets, this difference was negligible, suggesting the choice between these three models matters most for memory-constrained hardware when fine-tuning on larger, combined datasets rather than on a single retailer's data alone.

5.2 Limitations and Threats to Validity

Our results are subjected to a number of limitations and threats to validity. We discuss these in terms of construct, internal, external, and reliability validity, following the categorization commonly used when evaluating empirical studies in software engineering [39]. We will also discuss the quality of the sources used in this thesis.

5.2.1 Construct Validity

Construct validity concerns whether what we measure actually reflects what we intend to measure. The "serviceable" misclassifications were judgments made manually by us authors, based on our own interpretation of whether a customer would be bothered by what the model classified an item as. The resulting figure of around 98% should therefore be read in light of the fact that no external parties or actual customers were consulted, and no inter-rater agreement was measured. Similarly, the custom metrics we designed approximate the severity of a misclassification, but are not themselves independently validated against a ground truth of our datasets. They instead reflect our own assumptions about what makes one misclassification worse than another.

5.2.2 Internal Validity

Internal validity concerns whether observed effects, such as the performance gains from fine-tuning, can be attributed to the methodological choices made rather than to confounding factors. The hyperparameters we chose were based on our own research, the available hardware, and limited manual experimentation, rather than a systematic search such as grid search. This means that the upper bounds that can be found in the models we tested may not reflect their true potential. Likewise, having hard negatives for the larger datasets and not the smaller ones was a choice made based on an informal observation rather than a controlled comparison. Furthermore, SVM classification was used on all three embedding models. The classifier and embedding space work together and therefore part of the differences observed between models could stem from this fixed choice rather than from embedding quality alone.

5.2.3 External Validity

External validity concerns how well these findings generalize beyond the data studied. All data used in this thesis originates from fashion product descriptions written in Swedish. There is therefore a concern that our findings would not transfer to other languages or product categories. As can be seen in Table 4.3, there are differences between customers when it comes to performance metrics such as accuracy, even when the same model is used. What the exact differences stem from is unclear, since it can pertain to the sizes of the datasets, the data itself or other factors which are not explored in this thesis. Furthermore, all the models that were chosen in this thesis are based on our own research, which is explained in Section 3.2. These models represent a small selection from a much larger and fast-moving open-source model landscape, meaning that other models could perform differently.

5.2.4 Reliability

Reliability concerns whether the study would yield the same results if repeated. Exact numerical replication of our results would be difficult, since the underlying datasets were provided to us under a confidentiality agreement with Voyado and are not publicly accessible. Furthermore, the differences in the report between the models are sometimes within a few percentage points and should therefore not be assumed to statistically be reliable differences.

5.2.5 Quality of Cited Sources

A number of the sources cited in this thesis, particularly those used to motivate our research gap and provide background context, such as Palen-Michel et al. [2], are preprints hosted on arXiv rather than peer-reviewed publications. As such, the claims drawn from these sources have not undergone formal peer review, and could in principle later be revised, retracted, or found to contain errors not yet identified through that process. We used these sources primarily for contextual and motivational claims rather than for findings underpinning our core methodological choices or key conclusions.

Chapter 6
Conclusions

LLMs are capable of impressive results across a wide range of tasks, but our results suggest that, for text-based fashion product type classification, SLMs can offer a viable and considerably lighter alternative. By searching for relevant models on Hugging Face (Section 3.2), we identified three pre-trained embedding models, E5-Small, E5-Multilingual, and MiniLM, that performed well on our task. We also evaluated three zero-shot classification models, but found that none performed adequately, which can be seen in Table 4.9. Embedding models fine-tuned on domain-specific data were clearly preferable for our task. Measured using standard metrics such as accuracy and F1-score, all three embedding models performed well once fine-tuned, reaching accuracies between approximately 89% and 96% depending on the dataset, these results can be seen in Tables 4.3–4.8.

While fine-tuning required a significant amount hardware, it was performed entirely on a single, consumer grade laptop (Section 3.4.1), making our experiments reproducible by others with access to a comparable dataset and similar hardware, though, as discussed in Section 5.2, the underlying retailer data itself cannot be shared due to a confidentiality agreement with Voyado. We also found that fine-tuning a model sequentially on a new customer, after it had already been fine-tuned on a different one, was possible without retraining from scratch. This did however come at a small cost to accuracy compared to fine-tuning on the combined data directly, and this cost was larger when the customers were introduced in one order rather than the other (Section 5.1.4). This suggests that, while accommodating a new customer without full retraining is feasible and can possibly save computational resources, the order in which customers are introduced can have a small negative effect on accuracy.

By introducing custom metrics inspired by Nearest Neighbor classification (Section 4.1.3), we were able to identify and examine troublesome misclassifications more closely than would be possible using standard metrics alone. As discussed in Section 4.3, even among misclassifications, our models produced classifications we judged to still be serviceable in the majority of cases. Combined with the overall miss rate of our final models, this suggests an estimated

overall acceptable-classification rate of around 98%, though, as noted in Section 5.2, this estimate rests on a small manually reviewed sample and should be interpreted with appropriate caution rather than as a precise figure.

Considering its lower training time, smaller parameter size, and performance comparable to the other two embedding models we tested, our results suggest MiniLM offers the most favorable trade-off for practitioners seeking a competent SLM for text-based fashion product classification. We suggest considering Microsoft's MiniLM model from Hugging Face, fine-tuned on the target dataset using the MultipleNegativesRankingLoss function with hard negatives, with a scikit-learn SVM classifier as the final layer. We do not, however, have grounds in this thesis to claim that this approach matches the performance of current large language model implementations directly, since we did not perform a head-to-head comparison on the same task and dataset. Instead, our results indicate that a small, fine-tuned embedding model of this kind can be a practical and considerably cheaper alternative worth evaluating before defaulting to a larger model.

Returning to the research questions posed in Section 1.3, our results suggest the following. For RQ1, we found that accuracy, precision, recall, and F1-score, together with CPU time, GPU usage, and peak memory, provide a reasonably complete picture of both predictive quality and hardware cost, though, as discussed in Section 5.1.1, what counts as the relevant hardware cost differs between fine-tuned and zero-shot models. For RQ2, our custom step accuracy, Nearest Neighbor, and critical misclassification metrics let us look beyond the predicted label to the structure of the embedding space itself. This separated issues in the embedding model from issues in the classification head, and let us estimate how often our models' errors would actually bother a customer in practice. For RQ3, popularity and stated purpose on Hugging Face proved a reasonable basis for selecting pre-trained embedding models, but a poor predictor of suitability for our zero-shot models, none of which performed adequately despite being popular and purpose-built for classification. For RQ4, our three fine-tuned embedding models performed similarly to one another on standard metrics, with MiniLM offering the best trade-off once hardware usage is taken into account, while all three clearly outperformed the zero-shot models we tested. For RQ5, fine-tuning improved performance substantially over the pre-trained baselines across all three models, though the size of this improvement varied by customer and was not fully explained by dataset size alone.

These conclusions should be read alongside the limitations discussed in Section 5.2. In particular, our findings are based entirely on fashion product data in Swedish, and we cannot say with confidence whether they would generalize to other product domains, other languages, or retailers with substantially different data. Our hardware measurements are similarly tied to the single laptop configuration used throughout this thesis. We see addressing this generalizability as one of the more important directions for future work (Section 6.1).

For research, our results add to the growing body of evidence that small, openly available language models can perform competitively with much larger models on narrow, well-defined classification tasks, and our custom metrics offer one way to evaluate such models with more nuance than exact-match accuracy alone allows. For practice, our results suggest that organizations with access to labeled product data, but without the computational resources or

budget for large language models, can achieve strong product type classification performance using small, fine-tuned embedding models on consumer grade hardware.

6.1 Future Work

This section will present some possible future work that can be made based on the work and results from this thesis.

Multiple Labels

A single product could possess multiple different labels, which would affect how the model is used and evaluated. An example would be a product being classified as both a T-shirt and a top. Because of this, it would be interesting to see how well the model performed if a correct guess would be any or all of the labels a product possessed. Examining how the models would be evaluated and how much those results would differ from those presented in this thesis would also be of value.

Other Domain-Specific Performance Measures

The correctness of a label assigned to a product could be evaluated against different criteria. It would be interesting to create an ontology, where the model is not as heavily penalized for mistaking jeans for pants. The ontology could be created based on the body parts they are worn on, or in what type of weather they are commonly worn. Creating a hierarchy of misclassification severity based on this ontology would most likely yield interesting findings compared to those found in this thesis.

Cross-Domain and Cross-Lingual Generalization

As discussed in Section 5.2, this thesis is limited to fashion products described in Swedish. It would be valuable to investigate whether our findings, including the relative performance of the three embedding models and the effectiveness of fine-tuning with hard negatives, generalize to other e-commerce domains such as electronics or home goods, and to other languages beyond Swedish.

Real-World Deployment

This thesis evaluated model performance and hardware usage in a controlled, offline setting. Future work could investigate how a fine-tuned SLM performs when deployed in a live production pipeline, for example regarding inference latency under real traffic, how performance changes as new products and categories are added over time, and how the model should be maintained or re-fine-tuned as a retailer's product catalog evolves.

Further Development of the Custom Metrics

The custom metrics introduced in this thesis (Section 4.1.3) were designed and applied within the scope of this thesis but, as discussed in Section 5.2, were not validated against an indepen-

dent ground truth of misclassification severity. Future work could investigate this validation directly, for example by having multiple independent human raters score a larger sample of misclassifications, and could explore whether the same metrics generalize to domains beyond fashion retail.

References

[1] Eurostat, "E-commerce statistics for individuals," 2026, accessed: 2026-06-13. [Online]. Available: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=E-commerce_statistics_for_individuals

[2] C. Palen-Michel, R. Wang, Y. Zhang, D. Yu, C. Xu, and Z. Wu, "Investigating LLM Applications in E-Commerce," 2024, arXiv:2408.12779 [cs.CL].

[3] A. Gupta, B. Thomas, H. Asnani, P. R. Madduru, S. Feroze, S. Subramanian, V. Elango, and M. Gungor, "Small Language Models (SLMs) Can Still Pack a Punch: A Survey (Updated 2026)," 2026, arXiv:2501.05465 [cs.CL].

[4] J. T. Licardo, N. Tanković, I. Osman, I. Lorencin, and S. Baressi Šegota, "Performance Trade-Offs of Optimizing Small Language Models for E-Commerce," *Big Data and Cognitive Computing*, vol. 10, no. 5, p. 155, 2026, doi: 10.3390/bdcc10050155.

[5] D. A. Koutsomitropoulos and S. Skoulidis, "Small language models are diligent learners: Evaluation of embedding-and transformer-based language models for text classification and recommendation," in *2025 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*. IEEE, 2025, pp. 1–5, doi: 10.1109/INISTA68122.2025.11249552.

[6] Voyado, "About Voyado - Voyado," 2026, accessed: 2026-06-14. [Online]. Available: https://voyado.com/about-voyado/

[7] ——, "CX for Fashion Apparel - Voyado," 2026, accessed: 2026-06-14. [Online]. Available: https://voyado.com/solutions/fashion/

[8] Tableau, "Everyday examples and applications of artificial intelligence (AI)," 2026, accessed: 2026-06-20. [Online]. Available: https://www.tableau.com/data-insights/ai/examples

[9] F. Kalota, "A primer on generative artificial intelligence," *Education Sciences*, vol. 14, no. 2, p. 172, 2024.

[10] W. Liang, Y. Zhang, M. Codreanu, J. Wang, H. Cao, and J. Zou, "The widespread adoption of large language model-assisted writing across society," *Patterns*, vol. 6, no. 12, 2025, doi: 10.1016/j.patter.2025.101366.

[11] Örpek, Zeynep, Tural, Büşra, and Destan, Zeynep, "The language model revolution: LLM and SLM analysis," in *2024 8th International Artificial Intelligence and Data Processing Symposium (IDAP)*. IEEE, 2024, pp. 1–4.

[12] Z. Lu, X. Li, D. Cai, R. Yi, F. Liu, X. Zhang, N. D. Lane, and M. Xu, "Small language models: Survey, measurements, and insights," 2025, arXiv:2409.15790 [cs.CL].

[13] scikit-learn, "Classification metrics," 2026, accessed: 2026-03-17. [Online]. Available: https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics

[14] K. M. Sujon, R. Hassan, K. Choi, and M. A. Samad, "Accuracy, precision, recall, f1-score, or mcc? empirical evidence from advanced statistics, ml, and xai for evaluating business predictive models," *Journal of Big Data*, vol. 12, no. 1, p. 268, 2025, doi: 10.1186/s40537-025-01313-4.

[15] J. Liu, J. Liu, W. Du, and D. Li, "Performance analysis and characterization of training deep learning models on mobile device," in *2019 IEEE 25th international conference on parallel and distributed systems (ICPADS)*. IEEE, 2019, pp. 506–515, doi: 10.1109/ICPADS47876.2019.00077.

[16] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[17] J. Boudier, "Hugging face enterprise: Why you need it, how to get it," 2025, accessed: 2026-05-26. [Online]. Available: https://huggingface.co/spaces/huggingface/how-to-upgrade-to-enterprise

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019, pp. 4171–4186, doi: 10.18653/v1/N19-1423.

[19] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5776–5788, 2020.

[20] L. Wang, N. Yang, X. Huang, L. Yang, R. Majumder, and F. Wei, "Multilingual e5 text embeddings: A technical report," 2024, arXiv:2402.05672 [cs.CL].

[21] C. Jansen, M. Nalenz, G. Schollmeyer, and T. Augustin, "Statistical Comparisons of Classifiers by Generalized Stochastic Dominance," *Journal of Machine Learning Research*, vol. 24, no. 231, pp. 1–37, 2023.

[22] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural

language generation, translation, and comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 7871–7880, doi: 10.18653/v1/2020.acl-main.703.

[23] Tung.M.Phung, "A review of pre-trained language models: from BERT, RoBERTa, to ELECTRA, DeBERTa, BigBird, and more," 2021, accessed: 2026-05-24. [Online]. Available: https://tungmphung.com/a-review-of-pre-trained-language-models-from-bert-roberta-to-electra-deberta-bigbird-and-more/

[24] A. Conneau, R. Rinott, G. Lample, A. Williams, S. Bowman, H. Schwenk, and V. Stoyanov, "XNLI: Evaluating cross-lingual sentence representations," in *Proceedings of the 2018 conference on empirical methods in natural language processing*, 2018, pp. 2475–2485, doi: 10.18653/v1/D18-1269.

[25] P. He, J. Gao, and W. Chen, "DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing," 2021, arXiv:2111.09543 [cs.CL].

[26] D. Berrar, "Cross-validation," 01 2024, doi: 10.1016/B978-0-323-95502-7.00032-4.

[27] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," vol. 14, no. 2, Montreal, Canada, 1995, pp. 1137–1143.

[28] P. Cunningham and S. J. Delany, "K-nearest neighbour classifiers-a tutorial," *ACM computing surveys (CSUR)*, vol. 54, no. 6, pp. 1–25, 2021, doi: 10.1145/3459665.

[29] scikit-learn, "Support Vector Machines - scikit-learn 2.8.0 Documentation," 2026, accessed: 2026-05-28. [Online]. Available: https://scikit-learn.org/stable/modules/svm.html

[30] Meta, "Introducing Llama 3.1: Our most capable models to date," 2024, accessed: 2026-06-15. [Online]. Available: https://ai.meta.com/blog/meta-llama-3-1/

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011.

[32] Hugging Face, "Hugging Face – The AI community building the future." accessed: 2026-05-27. [Online]. Available: https://huggingface.co/

[33] Microsoft, "DirectML," 2024. [Online]. Available: https://github.com/microsoft/DirectML/tree/master/PyTorch

[34] Hugging Face, "Trainer," 2026, accessed: 2026-05-20. [Online]. Available: https://sbert.net/docs/package_reference/sentence_transformer/trainer.html

[35] ——, "Sentence Transformers," 2026, accessed: 2026-05-20. [Online]. Available: https://github.com/huggingface/sentence-transformers/blob/main/sentence_transformers/util/hard_negatives.py

[36] ——, “Losses,” 2026, accessed: 2026-05-15. [Online]. Available: https://www.sbert.net/docs/package_reference/sentence_transformer/losses.html

[37] P. Qi, Z. Liu, X. Zhou, T. Pang, C. Du, W. S. Lee, and M. Lin, “Defeating the Training-Inference Mismatch via FP16,” 2025, arXiv:2510.26788 [cs.LG].

[38] Hugging Face, “Trainer - Training Arguments,” 2026, accessed: 2026-05-15. [Online]. Available: https://huggingface.co/docs/transformers/v5.9.0/en/main_classes/trainer#transformers.TrainingArguments

[39] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009, doi: 10.1007/s10664-008-9102-8.

EXAMENSARBETE Computationally Limited Product Type Classification
STUDENTER Melissa Engberg, Samuel Högfeldt
HANDLEDARE Noric Courdec(LTH), Andreas Björklund(Voyado)
EXAMINATOR Elizabeth Bjarnason(LTH)

Stor prestanda i ett litet paket

POPULÄRVETENSKAPLIG SAMMANFATTNING **Melissa Engberg, Samuel Högfeldt**

Du använder säkert AI varje dag, men vems dator är det egentligen som svarar på dina frågor? Kan nuvarande AI-appliceringar från stora datacenter köras lokalt? Detta arbete undersöker hur AI-appliceringen av produktklassificieringar presterar på små, gratis, open source modeller.

I vardaglig teknikanvändning förekommer AI alltmer. Google sökningar ger nu en AI overview längst upp, Microsoft office är helt integrerat med deras Copilot system och Meta har en AI du kan chatta med bland resten av dina kontakter. Många av de här funktionerna drivs av stora, resursintensiva LLM:er som företag betalar höga token-konstnader för att använda. Dessa modeller möjliggör stora framsteg inom industrin. Frågan är om det alltid krävs en LLM för att lösa alla problem, kanske skulle vissa lösningar kunna åstadskommas med mindre modeller.

I vårt examensarbete undersöker vi, i samarbete med Voyado, möjligheten att använda gratis, open source, Small Language Models, ofta kallade SLMs, för att utföra produktklassficieringar inom mode e-handel. Modellerna hämtar vi från AI-forsknings plattformen Hugging Face som vi därefter tränar och kör på en komersiell HP Zbook Laptop.

I arbetet finner vi 3 relevanta embedding modeller, MiniLM, E5-Small och E5-small multilingual. Dessa modeller är tidigare tränade på större dataset bestående av diverse frågor och svar vilket gör dem optimala för språkförståelse. Dessa modeller tar vi sedan och fine-tunar på vårt eget dataset för att se hur det påverkar prestandan.

Vidare valdes 3 relevanta zero-shot modeller i vår undersökning. Förtränade modeller vars mål är att utföra uppgifter utan att träna på data specifik till uppgiften. Dessa kan vara intressanta om man saknar lämpliga datasets eller rätt hårdvara för att träna modeller. Vi finner dock att dessa modeller presterar väldigt dåligt och inte är användbara för vårt syfte.

Det slutgiltiga resultatet är att vi kommer fram till att alla 3 embedding modellerna presterar med en noggrannhet på ungefär 90% efter fine-tuning. När vi sedan applicerar våra egendesignade mätningar, skräddarsydda för domänen, hittar vi att modellerna levererar dugliga klassificieringar 98% av tiden. Vi konstanterar att produktklassificieringar inom mode är möjligt med betydligt mindre modeller, där vi anser att MiniLM är den optimala modellen att utgå från baserat på att den har kortast träningstid och minst parameterstorlek.