Universiteit Leiden
The Netherlands

Opleiding Informatica

Adapting the CoThought pipeline for training a BabyLM on Dutch data

Thijs Groeneweg

Supervisor:
Gijs Wijnholds

Second reader:
Max van Duijn

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

28/06/2001

Abstract

This research explores how the CoThought pipeline, which is designed to train smaller language models (BabyLMs) with limited data on English data, can be applied to Dutch language data. Just like the original CoThought pipeline, we utilized large language models to create instances of Natural Language Understanding tasks, which are then used to train BabyLMs. We explore the use of this pipeline on Dutch data by translating the English BabyLM challenge dataset into Dutch, experimenting with various prompting strategies for the LLM, and integrating originally Dutch data into the training process. Instead of fully fine-tuning the model like the original pipeline, adapters (a more parameter-efficient technique) are used on a BERT model. To evaluate the adapters, we evaluated their performance on two NLU tasks: Natural Language Inference and Sentiment Analysis. We found that the CoThought pipeline can indeed be adapted for Dutch data, delivering results comparable to existing Dutch models, particularly in Natural Language Inference. Furthermore, adjusting the prompting strategy and incorporating native Dutch data did not significantly impact the performance of the trained adapters.

Contents

1 Introduction

In this section, we introduce the problem addressed in this thesis and provide all the necessary background information.

1.1 Introduction and Contextual Overview

1.1.1 Large Language Models

According to Deng and Lui [DL18] the field of 'Natural Language Processing (NLP) investigates the use of computers to process or understand human (i.e. natural) languages for the purpose of performing useful tasks'. NLP aims to replicate and understand the cognitive processes involved in human comprehension and generation of languages [DL18]. Some applications of NLP include speech recognition, dialogue systems, machine translation and sentiment analysis [DL18]. One way to perform NLP is through the use of Language Models (LM). Language models are 'models that assign probabilities to sequences of words' [JM21]. In other words, they aim to predict the likelihood of words appearing together in a sentence. Recently, a form of language models, called Large Language Models (LLMs), has gained popularity [MMN+24]. One reason for their advance is that their size and performance have increased greatly in previous years [MMN+24]. A large language model (LLM) is a Language Model (LM) that is especially good at language generation and understanding [Ope19]. While using this definition, it is important to keep in mind the current debates surrounding the term "understanding" when applied to LLMs. For example, van Dijk et al. wrote about 'The Need for Nuance in Current Debates and a Pragmatic Perspective on Understanding' [vDKSvD23].

1.1.2 The BabyLM Challenge

Although these models are good at language generation and understanding, they have some drawbacks. For example, they need a lot of computational power to run properly [RWC+19], consume large amounts of electricity [SGM19], and need a large amount of training data [WMC+23]. To address these drawbacks, the "BabyLM challenge" [WCM+23] was introduced. This challenge encourages community members to create a smaller language model ("a BabyLM") on a limited amount of data. The motivation for this challenge was that while huge efforts have been made to optimize the pretraining of LMs at large scales in previous years [RSR+20] [BMR+20], almost no progress has been made in pretraining smaller models at 'human-like scales' [WMC+23]. Large-scale models are trained on large amounts of data, which this challenge aims to reduce. The smaller models trained in this challenge, called Baby Language Models (BabyLMs), are models that use 'Sample efficient pretraining on a developmentally plausible corpus' [WCM+23]. In other words, these models are trained on a more manageable dataset that reflects a baby's early exposure to language. Previous research on language acquisition has shown that humans acquire language for the most part during the first stages of life (for an overview, see [Cla22] and [Tom03]). Major advances in language comprehension and expression often occur during early childhood. Furthermore, Chang and Bergen [CB22], and Nikolaus and Fourtassi [NF21] have shown that language modelling shows similarities to children's language acquisition. They found that both children and language models use data to continuously update their knowledge of the outside world.

1.1.3 The CoThought Pipeline

Due to the small size of the BabyLM challenge dataset, sentences often lack strong semantic ties with each other, making it difficult for models to learn contextual representations[ZYM+23]. To address this problem, a submission [ZYM+23] to this challenge by Warstadt et al. [WCM+23] proposed a Cothought pipeline to prepare a BabyLM with smaller corpus data similar to humans, 'leveraging the LLM chain of thought feature and the child's cognitive learning ability' (for an overview of 'the child's cognitive learning ability', see [Cla22]). In this way, the LLM and the child are "co-thinking" during the training process'. This pipeline leverages a LLM to create instances of Natural Language Understanding (NLU) tasks, which are a subset of NLP tasks focused on reading comprehension [Sem12]. These tasks are used to restructure the data set on which the BabyLM is trained. To do this, they applied In-Context Learning (ICL), 'which means the model makes predictions by learning from a natural language prompt describing the language task or learning from (only a few) examples' [ZYM+23]. For the successful application of ICL, they use a method of prompting introduced by Wei et al. [WWS+23] called Chain-of-Thought prompting, in which a series of intermediate reasoning steps are used to improve the reasoning capabilities of the language model.

1.2 Background and theory

1.2.1 Adapters vs Fine-tuning

Zhang et al. [ZYM+23] used a technique called fine-tuning on an existing "RoBERTa" model, in which the weights of a pre-trained model are trained on new data. This technique has been shown to yield strong performance [HR18] [RNSS18]. However, a more (parameter) efficient technique has emerged in which new modules are added between the layers of the model [HGJ+19]. After adding these modules, only the weights of these modules are updated, while the rest of the layers remain frozen [HGJ+19]. These modules are called adapters. 'Adapter-based tuning requires training two orders of magnitude fewer parameters to fine-tuning, while attaining similar performance' [RNSS18]. For our research, we will be using adapters using the python package provided by AdapterHub [PSP+23].

1.2.2 Training and Validation Loss

To know the progress of pretraining our models we use, amongst others, the training and validation loss. They are examples of loss functions. A loss function, also known as a cost function or error function, is a mathematical tool that assigns a real number to an event or the values of certain variables. This number represents the "cost" associated with that event or those values, offering a measure of how well a model is performing or how much it deviates from the desired outcome [HTF01].

To calculate the training and validation loss, a loss function is applied to the results of a model on the training and validation set respectively. These loss values can than be used as an indication of the performance of a model [TCERPCU23].

1.2.3 Epochs

In training a machine learning model, the model is said to have completed one epoch when it has passed through the training data once [Kal23]. Zhang et al. fine-tuned their model for 5 epochs. However, as we started to see diminishing returns in validation loss around 2 epochs we have only trained our adapters for that amount (see section 3.2 for further explanation).

1.2.4 Learning Rate

One way we can influence the speed of decreasing our loss functions is by changing the learning rate. The learning rate determines the size of the steps taken at each stage of the optimization process [Mur12]. The learning rate is crucial for guiding the algorithm towards minimizing the loss function [Mur12]. For our research, we will use the standard learning rate of the Adapters library [PSP+23] (which is 1e-4).

1.2.5 Evaluation Metrics

To test the performance of our models, we calculate some evaluation metrics. We calculate the following metrics:

Accuracy
Accuracy measures the fraction of accurate predictions, for both true positives and true negatives, of all the cases evaluated [Met78]. In other words, accuracy indicates the proportion of correct predictions out of the total number of instances examined. According to Metz [Met78], it is defined by:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{1}$$

Because this metric only calculates the number of correct predictions, it might not give an accurate indication of a model's performance when a dataset with imbalanced classes is used.

Precision
The precision calculates how many of the instances predicted to belong to a class actually belong to that class. According to Taha and Hanbury [TH15], it is defined by:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2}$$

Recall
'True Positive Rate (TPR), also called Sensitivity and Recall, measures the portion of positive voxels in the ground truth that are also identified as positive by the segmentation being evaluated' [TH15]. In other words, it tells us how many of the actual positive items were correctly identified as positive by the test. According to Taha and Hanbury [TH15], it is defined by:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3}$$

F1 Score
F1-Measure is a trade-off between precision and recall. According to Taha and Hanbury [TH15], F1-Measure is given by:

$$\text{F1} = \frac{2 * \text{TP}}{2 * \text{TP} + \text{FP} + \text{FN}} \tag{4}$$

Because this measure is a trade-off between precision and recall, it gives a realistic indication of a model's performance, even if data used has a class imbalance.

1.2.6 Prediction Head

After pretraining our adapters, we will further fine-tune them on a downstream task. For this, we will add "prediction heads" to our adapters for each task. The prediction head is the final component in a Transformer Language Model (TLM). Its primary job is to generate the output predictions. It does this, by calculating the likelihood of each possible output [KKYI23]. In other words, it assigns a probability to each item in the model's vocabulary, indicating how likely it is to be the correct prediction.

1.2.7 NLU Tasks

We fine-tuned our adapters on two different NLU tasks (for the implementation see section 3.3).

Natural Language Inference (NLI)
First of all, we fine-tuned our adapters for NLI, 'where the goal is to determine, for a given premise sentence whether it contradicts, entails, or is neutral with respect to a given hypothesis sentence' [WM21]. In other words, the task involves assessing the relationship between two sentences to classify the hypothesis as either being supported by the premise, in conflict with it, or neither.

Sentiment Analysis
Secondly, we fine-tuned our adapters for Sentiment Analysis. Sentiment analysis is the automated process of extracting attitudes, opinions, views, and emotions from various sources such as text, speech, tweets, and databases using Natural Language Processing (NLP) [AS16]. This process involves categorizing the opinions expressed in the text into classes like "positive", "negative", or "neutral" [AS16].

1.2.8 Different BERT Models

BERT
In our research, we added adapters to a BERT model. BERT is trained to understand the meaning of words in a sentence by considering the surrounding words on both sides, rather than just the words that come before or after [DCLT18]. It learns from text that has not been labeled with specific categories or tags, using this context from all layers of the model to build a more comprehensive understanding of the language [DCLT18]. This causes that after pretraining, BERT can be fine-tuned to create top-performing models for many tasks, like question answering and language inference, without needing major changes to the model's structure[DCLT18].

RoBERTa
The BERT model we used is different from the model used by Zhang et al. [ZYM+23]. They used a RoBERTa model. However, this model is not available for use with adapters [PSP+23]. RoBERTa builds upon the foundation laid by previous models like BERT, but it refines the training process to achieve better performance across various language understanding tasks [LOG+19]. Because Liu et al found that BERT was under-trained, they used extended training duration, larger batches of data, and a bigger amount of training data for RoBERTa [LOG+19].

BERTJE
In the discussion (see section 4) we compare our adapters with BERTJE and multilingual BERT. BERTJE is a Dutch variant of BERT. It utilizes the same transformer-based architecture and parameters as BERT but is trained on a much larger and more diverse dataset consisting of 2.4 billion tokens [dVvCB+19]. This dataset includes a wide range of Dutch texts, where BERT only used Dutch Wikipedia text [dVvCB+19].

Multilingual BERT
Multilingual BERT is a transformer-based model similar to BERT but unlike (the original English) BERT it is trained on Wikipedia articles from 104 languages, using a shared vocabulary of word pieces [PSG19].

1.3 Problem Statement

Zhang et al. [ZYM+23] have not explored the application of the CoThought pipeline to a Dutch dataset, which highlights a research gap. To address this, we aim to construct a Dutch dataset by translating the original English dataset from the BabyLM Challenge [WMC+23] using Google Translate via the python package "deep_translater"[1]. This raises the question of the impact of using translated data versus originally Dutch data. Furthermore, there is potential to optimize the pipeline's performance for Dutch data. To this end, we investigate the effects of modifying the prompting strategy for Dutch data.

When measuring the impact of the various changes we make to the CoThought pipeline, we will need to train multiple models. This process can be quite time-consuming. In a study by Bastian Bunzec et al. [BZ23] that was carried out as a response to the BabyLM challenge, it was observed that training the BabyLM took considerable time. They wrote 'Due to the small number of parameters, training times varied between 3–4 hours for the smallest models to 20h for the largest models.'. The largest model they used was trained on the strict-small dataset from the BabyLM challenge [WCM+23] for 10 epochs, this dataset contains 10 million words. However, Zhang et al. [ZYM+23] trained on the 100 million words dataset from the BabyLM challenge for five epochs. Because of this, they had to use more computing power. Therefore, it is crucial to tackle this computational challenge and investigate potential optimization methods such as using adapters.

[1]Available at: https://pypi.org/project/deep-translator/

1.4 Thesis Overview

This thesis was produced as a bachelor thesis by Thijs Groeneweg for LIACS, under supervision by Gijs Whijnholds and with as second reader Max van Duijn. This chapter contains the introduction of this thesis, key definitions and background information on related works. Section 1.3 describes the problem this thesis aims to solve. chapter 1.5 introduces the research questions. Chapter 2 describes the methods we used for adapting the CoThought pipeline to Dutch data. Chapter 3 describes the results of our research and some interpretations of those results. Chapter 4 discusses the implication of our results and the answers to our research questions. chapter 5 provides our final thoughts and a future outlook.

1.5 Research Question

Based on the previous research and limitations discussed above, we propose the following research question and sub-questions:

- Can we adapt the CoThought pipeline for training a BabyLM on Dutch data?
  - What is the impact of changing the prompting strategy used in the Cothought pipeline for training adapters on Dutch data?
  - What is the impact of including originally Dutch data in the pretraining data when using the Cothought pipeline for training adapters on Dutch data?

2 Methodology

In this research, we will produce adapters for a BERT model on a Dutch dataset using the CoThought pipeline as proposed by Zhang et al.[ZYM+23]. Zhang et al. [ZYM+23] used various prompts on GPT 3.5 turbo[2] to pre-process the 100M dataset from the BabyLM challenge. After this, they fine-tuned a RoBERTa base model [LOG+19] on the pre-processed data. Eventually, they compared their model against the RoBERTa base model using the tools provided by the BabyLM challenge [WMC+23]. To produce a BabyLM for Dutch, we need to change various parts of this process. After making these changes, we produce multiple BabyLMs and benchmark them against a baseline.

First of all, we need to produce a Dutch dataset, as the original dataset is in English. Secondly, due to resource limitation we need to use a different LLM. For this we will use Gemini 1.0 pro via the Vertex AI API[3]. We chose this model for its availability and high accuracy when initially testing our prompts. We also have limited resources for training the BabyLM, this becomes especially important because we need to train multiple models. To address this issue, we will be training adapters instead of pretraining and fine-tuning the whole model. For this we will be using AdapterHub [PSP+23] for its easy and flexible implementation. Because AdapterHub [PSP+23] currently does not support

[2] https://platform.openai.com/docs/models/

[3] Accessible from: https://console.cloud.google.com/vertex-ai/publishers/google/model-garden/gemini-pro

RoBERTa models, we will be using a BERT model to train adapters. Lastly, the benchmarks provided by the BabyLM challenge are designed for English models. To address this, we will be using some datasets from the DUMB benchmark [dVWN23] instead. The various changes we made to the CoThought pipeline are described in more detail below.

2.1 Including Originally Dutch Data in Pretraining Data

To generate the Dutch dataset, we start by translating the 100M dataset from the BabyLM challenge [WMC+23] using Google Translate via the Python package "deep_translator"[4]. This translated dataset is then transformed using the methods described in section 2.2 with prompt 6, resulting in our first dataset. This dataset will be used to train Model A1, serving as a comparison for Models B and C (see Table 1).
Next, to assess the impact of incorporating originally Dutch data into our translated dataset, we replace the children's stories data with ChiSCor, a corpus of freely-told fantasy stories by Dutch children, compiled by van Dijk et al. [vDvDVS23]. This dataset will be used to train Model C (see Table 1). To create a comparable baseline, we will also produce a dataset from the BabyLM data by reducing the children's stories dataset to match the number of generated NLU tasks in the ChiSCor dataset. This will be achieved by removing the final tasks that exceed the number in the ChiSCor dataset from a randomly shuffled version of the dataset used to train Model A1.
Finally, we will follow the steps described in section 2.3 to train and evaluate adapters on the various datasets we produced.

2.2 Changing the Prompting Strategy Used

Within the original Cothought pipeline by Zhang et al. [ZYM+23], a Large Language Model (LLM) is prompted to produce an NLU task for every five lines of data extracted from the BabyLM challenge dataset [WMC+23]. During this step, Zhang et al. [ZYM+23] used GPT 3.5-turbo[5]. As an alternative, we will use Gemini-pro via the Vertex AI API on Google Cloud[6], as this model has shown above average accuracy when testing our prompts and is easily available. As shown by Wang et al. [WCD+24], different combinations of models and languages perform well on different prompting strategies. As such, we will try to optimize the prompting strategy used by Zhang et al. [ZYM+23] for use on Gemini-pro in Dutch.

First, we will test how the original prompting strategy used by Zhang et al. [ZYM+23] performs when translated into Dutch. They used the following prompt:

[4]Available at: https://pypi.org/project/deep-translator/
[5]https://platform.openai.com/docs/models/
[6]https://console.cloud.google.com/vertex-ai/publishers/google/model-garden/gemini-pro

Prompt 1: original prompt

Use the given sentences to create an example paragraph of an NLU task and its corresponding labels. The 5 sentences are: {input}.

Make a plan then write and determine. Your output should be of the following format:

Plan:
Your plan here.

Paragraph:
Your paragraph here.

Task:
[Only the NLU task name here, without additional information.]

Labels:
[Only the labels here, without additional information.]

At {input} they included five sentences from the BabyLM challenge dataset [WCM+23]. We used the following translated version of this prompt:

Prompt 2: original prompt translated to Dutch

Gebruik de gegeven zinnen om een voorbeeldparagraaf te maken van een NLU taak en de bijbehorende labels. De 5 zinnen zijn: {input}.

Jouw uitvoer moet het volgende formaat hebben:

Plan:
Jouw plan hier.

Paragraaf:
Jouw alinea hier.

Taak:
[Alleen de NLU taaknaam hier, zonder extra informatie.]

Labels:
[Alleen de labels hier, zonder extra informatie.]

When using this prompt, we noticed the LLM had difficulties in adhering to the format given. To improve compliance to this format, we changed the position of the sentences from the dataset so the prompt had a more logical order. After doing this, the LLM made tasks which followed the given format more closely (see prompt 3 in A.1).
When using the prompt with the changed order, we noticed that the model only made tasks for the

example tasks defined in the prompt (sentiment analysis and entity extraction). Because of this, we changed the prompt to include the specific NLU tasks the LLM can choose from (see prompt 4 in A.1). When using this prompt in our initial testing the model chose from the tasks we defined.
To generate relevant tasks, we include the NLU tasks we want to be generated in our prompt. For this, we input the NLU tasks used by the DUMB benchmark [dVWN23] in our prompt. These are the following: Part-Of-Speech Tagging, Named Entity Recognition, Word Sense Disambiguation, Pronoun Disambiguation, Causal Reasoning, Natural Language Inference, Sentiment Analysis, Abusive Language Detection and Question Answering. When we include these in prompt 4 (see prompt 4 in A.1), we get the following prompt:

Prompt 6: Include NLU tasks from the DUMB benchmark

Gebruik de gegeven invoerzinnen om een voorbeeldparagraaf te genereren voor een specifieke NLU-taak en de bijbehorende labels te identificeren.

Jouw uitvoer moet het volgende formaat hebben:

Plan:
Beschrijf in een paar zinnen hoe je de taak zult aanpakken.

Paragraaf:
Genereer een voorbeeldparagraaf op basis van de gegeven invoerzinnen.

Taak:
Specificeer de NLU-taak, kies uit: Part-Of-Speech Tagging, Named Entity Recognition, Word Sense Disambiguation, Pronoun Disambiguation, Causal Reasoning, Natural Language Inference, Sentiment Analysis, Abusive Language Detection of Question Answering. Geef alleen de taaknaam.

Labels:
Geef de alleen labels voor de gegenereerde paragraaf aan (bijv. positief/negatief sentiment, entiteitstypen).

De 5 te gebruiken invoerzinnen zijn: {input}

Zhang et al. [ZYM+23] used a separate prompt to ask the LLM for a coherency score. Then, the responses to the first prompt are filtered, so that we only use the responses where the coherency score is higher than seven. We translated the prompt used by Zhang et al. [ZYM+23] for use for the same purpose. After translating this prompt, we got the following:

Prompt 5: Filter by coherency score

Analyseer de volgende paragraaf, en eindig met op de laatste lijn: "Daarom is de coherentiescore s", waar s een integer van 1 tot 10 is. {input}

At {input} we included the responses to the first prompt. We will not be changing this prompt,

as it performed well when we tested it on some samples from the dataset. We will use regular expression to get the coherency score from the response.

To test how these prompts perform, we will generate two different datasets on which we will train the adapters. One of these will be generated using prompt 2, and one will be generated using prompt 6. These datasets are used to train model A2 and A3 respectively (see Table 1). The dataset used to train model A2 will be created by randomly selecting groups of 5 lines from the BabyLM challenge until we have selected 1/8th of the dataset, which are then processed using our prompts. The dataset used to train model A3 will be generated by randomly sampling 1/8th of the tasks generated for training model A1 in section 2.1. As performance is similar across the datasets used for Models A1, B and C (see section 3.1), we chose our baseline dataset. After generating these datasets, we will train and evaluate the adapters according to the methods described in section 2.3. In the Table below (Table 1) the different datasets and adapters are summarised.

Adapter	**Data used to train the model**	**Prompt**
Model A1	translated BabyLM datasets [WMC+23]	prompt 6
Model A2	1/8th translated BabyLM datasets [WMC+23]	prompt 6
Model A3	1/8th translated BabyLM datasets [WMC+23]	prompt 2
Model B	BabyLM datasets [WMC+23] children's stories resized to match the size of the ChiSCor dataset [vDvDVS23]	prompt 6
Model C	BabyLM datasets [WMC+23], children's stories replaced with the ChiSCor [vDvDVS23] dataset	prompt 6

Table 1: Explanation of the different adapters produced

2.3 Training and Evaluating the Adapters

In this section, we describe the process of training adapters, adding prediction heads, selecting evaluation metrics, and describing datasets for each NLU task.

2.3.1 Training Method

To train the BabyLMs, we start by adding adapters to a BERT model [DCLT18], which will then be pre-trained on the datasets outlined in the previous sections (see Table 1). Subsequently, prediction heads are added for each NLU task, enabling further fine-tuning of the adapters. While Zhang et al. [ZYM+23] used the BabyLM-challange evaluation pipeline [WCM+23] for evaluation of their model, we will need to adapt our evaluation strategy due to training on a Dutch dataset. For each NLU task, we will use a different dataset to fine-tune the adapter with the prediction head.

2.3.2 Evaluation Method

These datasets will also serve as the basis for evaluating the trained adapters and prediction heads, with specific evaluation methods tailored to each dataset. The datasets used are inspired by the DUMB benchmark [dVWN23]. We have limited the amount of tasks used for testing the models due to time limitations. Also, we have chosen two of the smaller datasets used by the DUMB

benchmark [dVWN23] because of resource limitations.

To asses the performance of the adapters, we calculate the following metrics: Accuracy, F1-score, Precision and Recall, alongside tracking training and validation losses. We have set the frequency of metric calculations based on the size of the datasets to ensure meaningful analysis without excessive computational burden. For instance, we calculate evaluation metrics for each epoch for smaller datasets, but for each 500 steps for larger datasets. In practise, this means that we calculated the evaluation metrics at each epoch when fine-tuning and at each 500 steps when pretraining. When we are done training, we will compare the checkpoints of the model by using the F1-score, because it provides a balanced measure of precision and recall, making it suitable for evaluating models on imbalanced datasets where the classes are not equally represented. Below, we will delve into the specific methods employed for each of the NLU tasks.

2.3.3 NLU Tasks

Natural Language Inference (NLI)
For NLI, we will use the SICK-NL dataset by Wijnholds et al. [WM21] to fine-tune the adapter. This dataset is already split in train, test and validation splits which we will use for their respective purposes. For fine-tuning on NLI we added a classification head.

Sentiment Analysis
For sentiment analysis we will use the DBRD v3.0 dataset by van der Burgh and Verberne [vdBV19]. This data set is divided into a test and train dataset, of which the training set is substantially larger. We will further split of 10 percent of the training dataset to be used as a validation set to prevent over-fitting. For fine-tuning on Sentiment Analysis we added a classification head.

3 Experiments and Results

3.1 Producing the dataset

First of all, we produced two datasets with instances of NLU tasks generated from the dataset of the BabyLM challenge [WMC+23], one generated from the whole dataset and one from 1/8th of the dataset. Furthermore, we produced a dataset with generated NLU tasks generated partly from the dataset of the BabyLM challenge [WMC+23] and partly from the ChisCor dataset [vDvDVS23]. The Dutch dataset constructed using the dataset from the BabyLM challenge consists of a total of 48454 generated NLU tasks after filtering by coherency score. This dataset consists of a total of 51979674 characters. In the table below (Table 2), you can see the proportion of NLU tasks generated for each of the datasets from the BabyLM challenge [WMC+23]:

Dataset	Proportion (%)
CHILDES	8.0
BNC	8.9
Children's Book Test	2.5
Children's Stories Text Corpus	8.9
Standardized Project Gutenberg Corpus	2.7
OpenSubtitles	49
QCRI Educational Domain Corpus	9.8
Wikipedia	2.4
Simple Wikipedia	6.6
Switchboard Dialog Act Corpus	1.6

Table 2: Proportion per Dataset of the BabyLM challenge

After removing the generated children's stories tasks that exceed the amount of tasks in the ChisCor dataset [vDvDVS23], we get a dataset containing 44543 NLU tasks that consists of 47646030 characters. In the Table below (Table 3), you can see the proportion of NLU tasks generated for each of the datasets from the BabyLM challenge [WMC+23].

Dataset	Proportion (%)
CHILDES	8.8
BNC	9.7
Children's Book Test	2.7
Children's Stories Text Corpus	0.92
Standardized Project Gutenberg Corpus	3.0
OpenSubtitles	53
QCRI Educational Domain Corpus	11
Wikipedia	2.6
Simple Wikipedia	7.2
Switchboard Dialog Act Corpus	1.7

Table 3: Proportion per Dataset of the BabyLM challenge

After replacing the children's stories dataset from the BabyLM challenge [WMC+23] with the ChisCor dataset [vDvDVS23] and running the prompts on this dataset, we get a dataset containing 44545 NLU tasks that consists of 47614064 characters. In the Table below (Table 4), you can see the proportion of NLU tasks generated for each of the datasets from the BabyLM challenge [WMC+23] and ChisCor [vDvDVS23].

Dataset	Proportion (%)
CHILDES	8.8
BNC	9.7
Children's Book Test	2.7
ChisCor	0.92
Standardized Project Gutenberg Corpus	3.0
OpenSubtitles	53
QCRI Educational Domain Corpus	11
Wikipedia	2.6
Simple Wikipedia	7.2
Switchboard Dialog Act Corpus	1.7

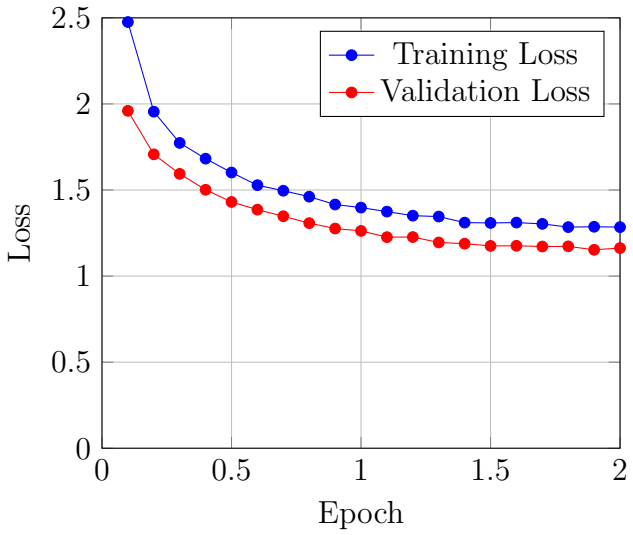
Table 4: Proportion per Dataset of the BabyLM challenge and ChiSCor [vDvDVS23]

In Table 4 we see that the proportion of the dataset we replace by ChisCor [vDvDVS23] is less then one percent of the dataset, which could affect our results.

Besides these three datasets, we have produced two datasets for testing how our changed prompt (prompt 6) performs against the original prompt translated to Dutch (prompt 2). These datasets are produced using of the translated dataset from the BabyLM challenge [WMC+23]. The dataset generated using prompt 2 contains 8086 instances of NLU tasks and consists of 6507898 characters. The dataset generated by randomly sampling the tasks generated using prompt 6 contains 6057 instances of NLU tasks and consists of 6489994 characters.
These two datasets contain similar amounts of characters, but vastly different amounts of instances of NLU tasks. This means that the individual instances of NLU tasks generated using prompt 6 are longer, but more of these tasks achieve a lower coherency score. A potential reason behind this phenomenon could be the LLM's tendency to produce multiple tasks when prompted with prompt 6, given the variety of tasks provided for selection. In contrast, prompt 2 typically results in the LLM generating a single task. Consequently, when multiple tasks are generated for each set of five sentences, it leads to longer text fragments. However, the inclusion of multiple tasks within these fragments may lead to a lower coherency score, which in turn produces less instances of NLU tasks.

3.2 Pretraining the Adapters

The datasets of the previous section (section 2.1) are used to train five adapters. A summary of the different datasets produced along with the adapters produced on them can be seen in Table 1.
In the graphs below, the training and validation loss calculated during pretraining of these adapters can be seen.



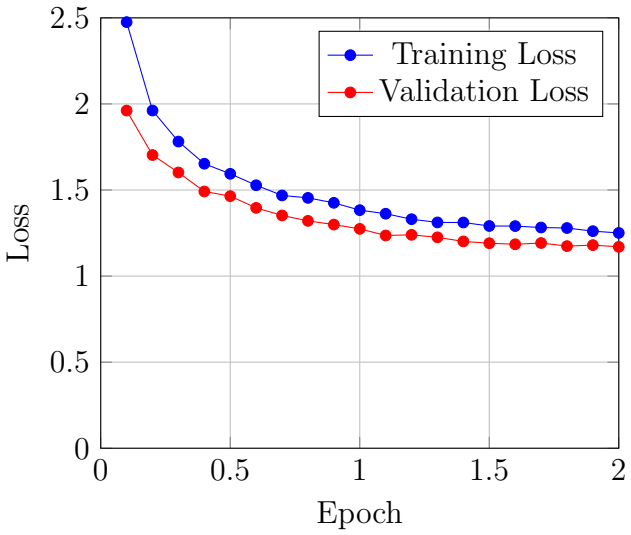


Figure 1: Training and validation loss for pretraining model A1 (left) and A2 (right)

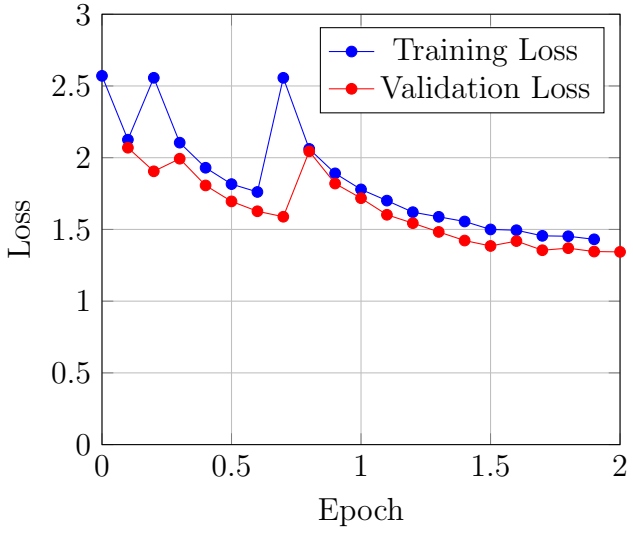


Figure 2: Training and validation loss for pretraining model A3

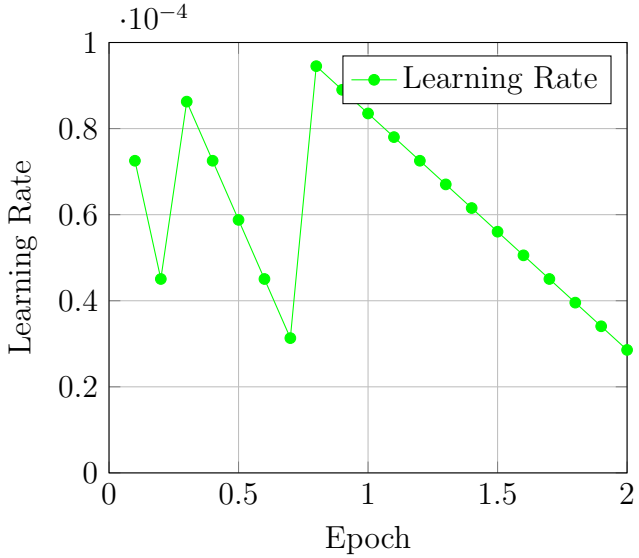
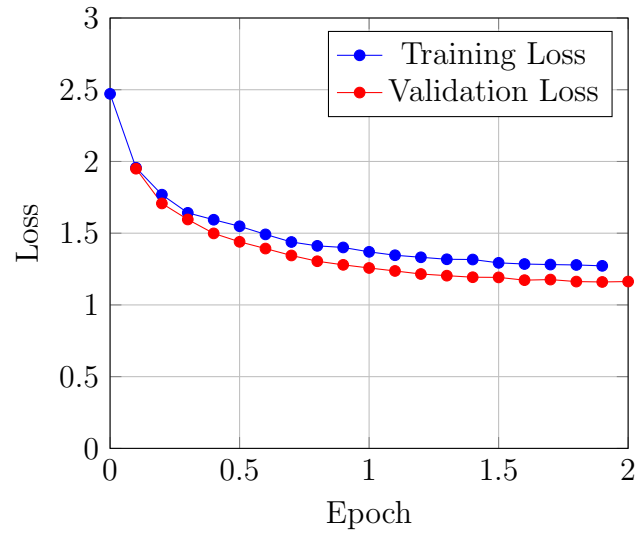


Figure 3: Learning rate for model A3



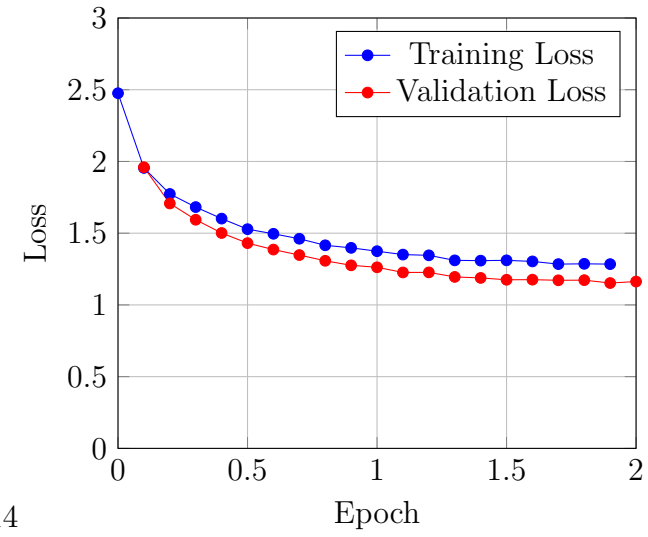


Figure 4: Training and validation loss for Model B (left) and C (right)

If we look at the graphs for the validation loss during pretraining (3.2, 2 and 4), it becomes clear that further pretraining of the model would give diminishing returns, as the graph flattens out. Furthermore, when looking at the graphs we can see decreasing loss values for both the training and validation sets, with comparable performance between the two. This suggests that the model is effectively learning from the data without over-fitting. The validation loss for all the models ends up similar, at below 1.20. However, for model A3 (see Figure 2), we see that there are some spikes in the loss values. These spikes come coupled with spikes in learning rate, as seen in Figure 3. There are two main reasons as to why this might happen. Firstly, the initialization of model parameters might be poor. Inappropriate initialization of parameters, either too small or too large, can lead to challenges such as vanishing gradients or over-amplification of certain dimensions, which in turn can contribute to spikes in the loss function during training [Sko20]. Secondly, using a learning rate that is too high could lead to overshooting [BL17], which could also be causing spikes in the loss function.

For fine-tuning, we have chosen to use the model checkpoints that achieved the lowest validation loss. These can be seen in the table below (Table 5).

Adapter	Validation loss	When
Model A1	1.1368	2 epochs
Model A2	1.1698	2 epochs
Model A3	1.2358	just before 2 epochs/ 9500 steps
Model B	1.1600	just before 2 epochs/ 9500 steps
Model C	1.1526	just before 2 epochs/ 9500 steps

Table 5: Lowest validation loss achieved by the models

3.3 Finetuning the Pretrained Adapters

After pretraining the adapters, they are all fine-tuned on two downstream tasks: Natural Language Inference and Sentiment analysis.

3.3.1 Natural Language Inference

During fine-tuning, we selected the best model checkpoints for further testing mainly based on the F1 score on the validation set. The metrics calculated during fine-tuning for NLI can be seen below.

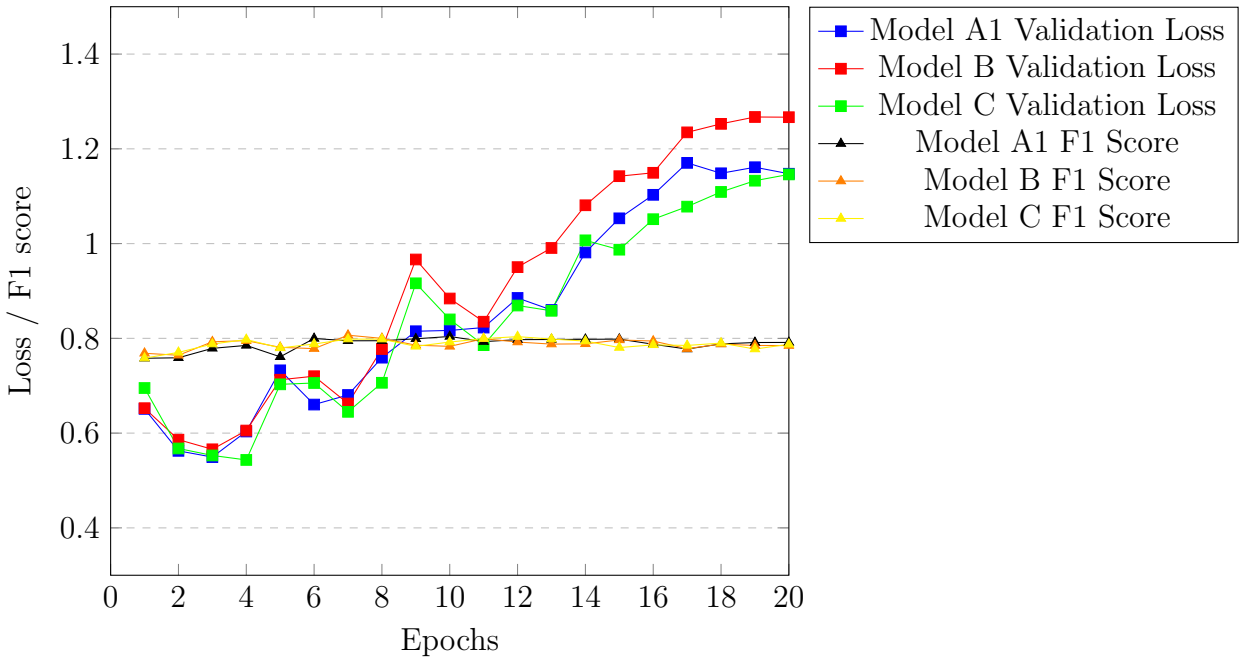


Figure 5: Comparison of Validation Loss and F1 Score for models A1, B, and C.

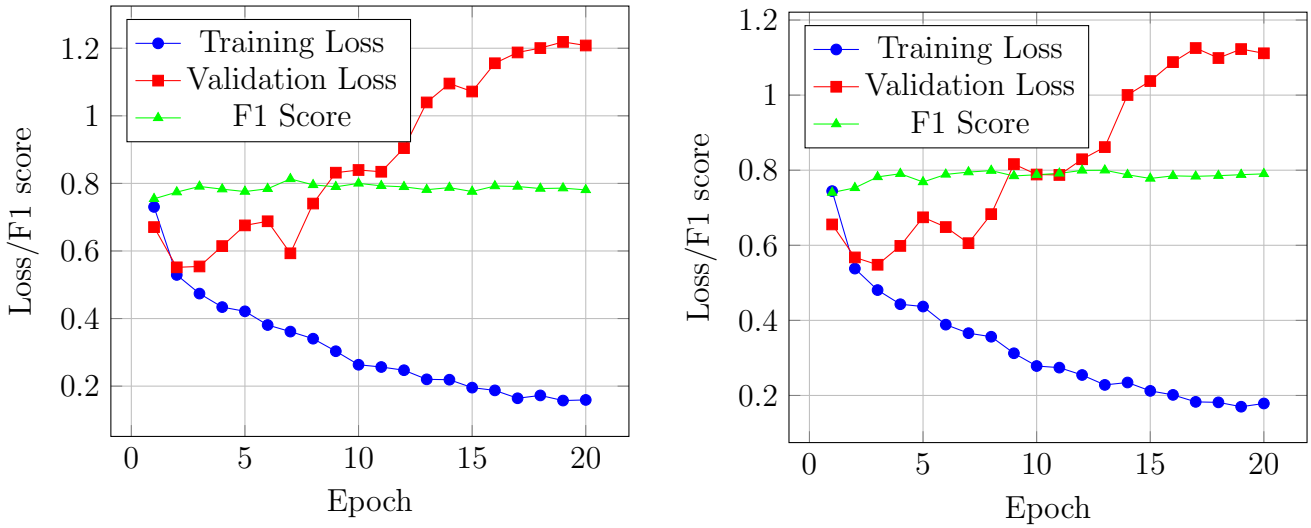


Figure 6: Training and Validation Loss, F1 Score when training model A2 (left) and model A3 (right) on NLI

For NLI, this method of selecting models solely based on F1 score can not be applied without a major downside. The models seem to reach a point where further training beyond this point increases the risk of over-fitting without significant gains in the F1 score, as demonstrated by their increasing validation loss (see Figure 5 and 6). This forms a problem for our selection procedure because the F1 score did increase slightly. As such, we will choose the checkpoints we use for further testing based on the F1 score and the validation loss. In the table below (Table 3.3.1), you can see the model checkpoints we have selected for testing on the test set. In the first column the F1 score

on the validation set is displayed, followed by the checkpoint at which the models achieved said validation score.

	F1 score (validation)	When	Accuracy (test)	F1 score (test)
Model A1	80.40 %	10 epochs	77.09 %	77.31 %
Model A2	80.01 %	10 epochs	81.35 %	81.38 %
Model A3	79.95 %	12 epochs	81.61 %	81.59 %
Model B	80.67 %	7 epochs	77.70 %	77.84 %
Model C	80.36 %	12 epochs	77.05 %	77.18 %

Table 6: Comparison of Evaluation Metrics for all trained adapters on NLI

For model A1, we chose to use the checkpoint at 10 epochs. After this point, we see a large increase in validation loss with no significant increase in F1 score. For model A2, A3, B and C we chose the checkpoint at 10, 12, 7 and 12 epochs respectively, using the same line of reasoning.

After fine-tuning the models and selecting the models that perform best on the validation set, we test these models on the test set. The results for this can be seen in the last two columns of Table 3.3.1. In this Table (Table 3.3.1) we can see models A2 and A3 perform best, with almost no difference in accuracy and F1 score between them. The other models perform similar to each other. Also, the accuracy and F1 score for all models is similar, despite the dataset not having a balanced class distribution.

3.3.2 Sentiment Analysis

In the graphs below, you can see the metrics calculated during finetuning of the models for Sentiment Analysis.

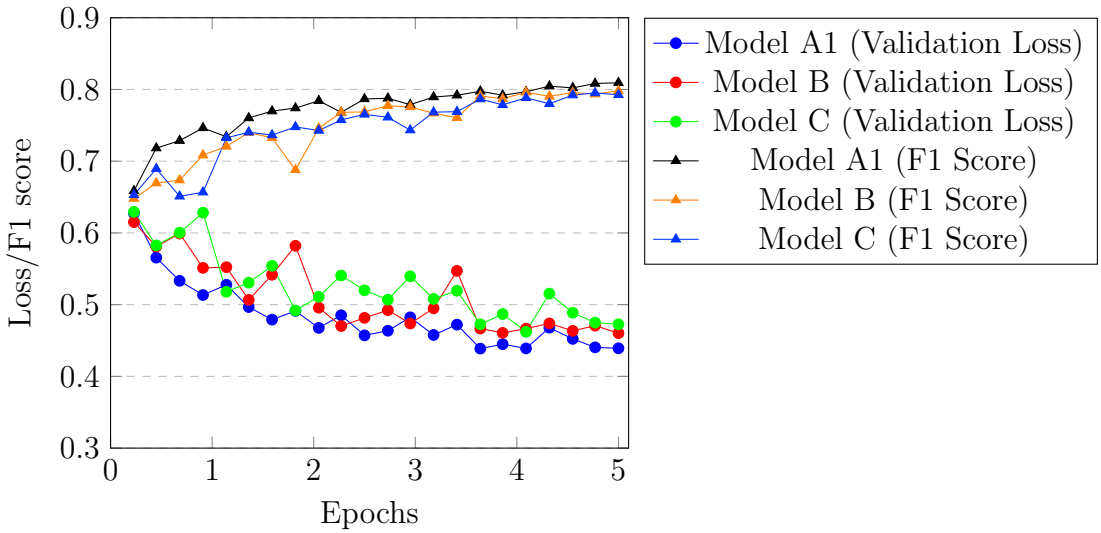


Figure 7: Comparison of Validation Loss and F1 Score for Models A1, B, and C when training on sentiment analysis

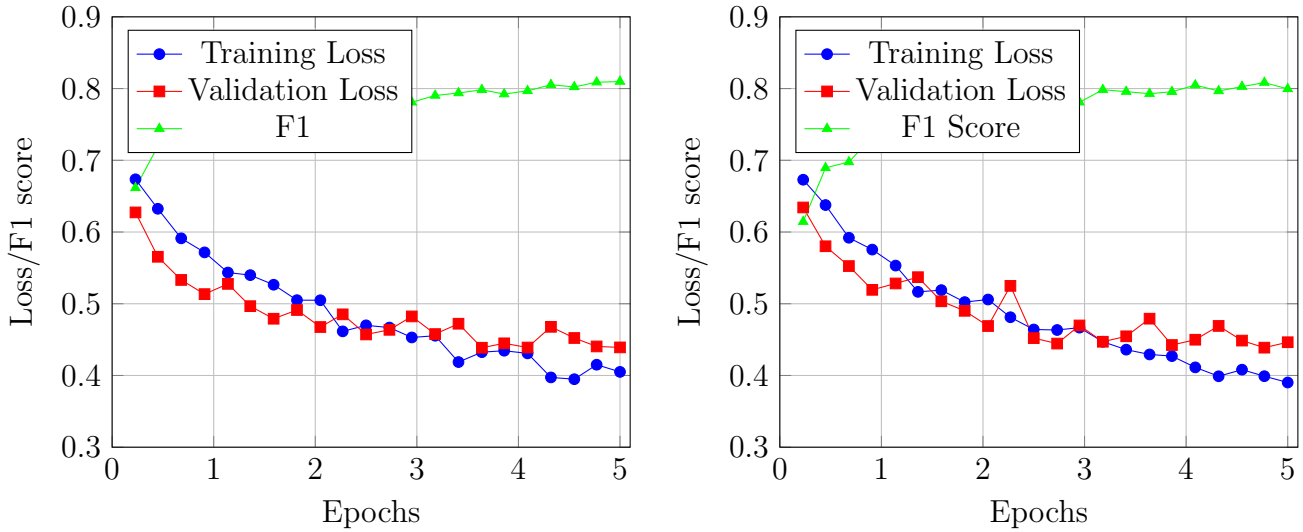


Figure 8: Training and Validation Loss and F1 score when training model A2 (left) and A3 (right) on Sentiment analysis

In contrast to when we were fine-tuning on NLI, we continue to experience an increasing F1-score coupled with a decreasing validation loss during almost the entire fine-tuning process (see Figure 7 and 3.3.2). Because of this, we select the models with the best F1 score on the validation set (without looking at the validation loss as we did for NLI) for further testing on the test set. In the

first two columns of the table below (Table 3.3.2) the selected model checkpoints along with their F1 score on the validation set are displayed:

	F1 score (validation)	When	Accuracy (test)	F1 score (test)
Model A1	80.93 %	5 epochs	77.09 %	77.31 %
Model A2	79.92 %	5 epochs	80.26 %	80.22 %
Model A3	80.83 %	just before 5 epochs/ 10500 steps	81.25 %	81.25 %
Model B	79.82 %	5 epochs	77.70 %	77.84 %
Model C	79.52 %	just before 5 epochs/ 10500 steps	77.05 %	77.18 %

Table 7: Comparison of Evaluation Metrics for all trained adapters on Sentiment Analysis

After fine-tuning the models and selecting the models that perform best on the validation set, we test these models on the test set. The results for this can be seen in the last two columns of Table 3.3.2. Here we see the same models performing well as for NLI (see Table 3.3.1). In this case, model A3 performs the best, followed by model A2. Lastly, model B performed slightly better than model A1 and C. Just like with NLI, the accuracy and F1 score for all models are similar. However, unlike the dataset we used for NLI, this dataset has a balanced class distribution.

4 Discussion

In the previous section (section 3), we found that generally model A2 and A3 performed best on Sentiment Analysis and NLI. However, all models achieve an F1 score of between 77% and 82%. Furthermore, when training the models on NLI, the models seem to reach a point where further training beyond this point increases the risk of over-fitting without significant gains in the F1 score. Interestingly, this does not happen when training on sentiment analysis.

In this section, we will be analyzing the impact of our results for answering the different research questions. To answer our main research question, we will start by answering our sub-questions.

4.1 Answering the Research Questions

Our first sub-question was:

What is the impact of changing the prompting strategy used in the Cothought pipeline for training adapters on Dutch data?

To answer this question, we need to compare model A2 and A3 (see Table 1). For both Sentiment Analysis (see Table 3.3.1) and NLI (see Figure 5) model A3 performed slightly better. The differences are not big however. As model A3 is the model with the original prompt translated to Dutch, we

can conclude that the impact of changing the prompting strategy used in the Cothought pipeline for training adapters on Dutch data is insignificant.

Our second sub-question was:

What is the impact of including originally Dutch data in the pretraining data when using the Cothought pipeline for training adapters on Dutch data?

To answer this question, we need to compare models B and C (see Table 1). For both Sentiment Analysis (see Table 3.3.1) and NLI (see Figure 5) the models performed similarly. Model B performed better for both tasks, but only by less than one percentage point. Model B is the model trained on the dataset that does not include originally Dutch data. As such, we can conclude that the impact of including originally Dutch data in the pretraining data when using the Cothought pipeline for training adapters on Dutch data is insignificant. However, like discussed in section 2.1, the proportion of the dataset we replace by ChisCor [vDvDVS23] is less then one percent of the dataset. This could mean, that changing this part of the dataset only has a small effect on the pretraining of the adapters.

Lastly, our main research question was:

Can we adapt the CoThought pipeline for training a BabyLM on Dutch data?

We were able to adapt the CoThought pipeline for training a BabyLM on Dutch data. For training the BabyLM we have used adapters, instead of pretraining and fine-tuning the whole model. In the table below (Table 8), we compare the performance of our best adapters with the performance of the fine-tuned models based on the same dataset.

Task	**Model**	**Accuracy (%)**
NLI	BERTJE [WM21]	83.94
	Adapters on BERT (Ours)	81.61
Sentiment Analysis	BERTJE [dVvCB+19]	93.00
	Multilingual BERT [dVvCB+19]	89.10
	Adapters on BERT (Ours)	81.25

Table 8: Comparison of Accuracy between adapters and fine-tuning

For both tasks, our model is model A3 (see Table 1). The results for BERTJE on NLI are from the paper associated with the SICK-NL dataset by Wijnholds and Moortgat [WM21]. The results for both Multilingual BERT and BERTJE on Sentiment Analysis are from the paper in which de Vries et al. produced BERTJE [dVvCB+19]. For NLI, our model performed fairly close to the implementation by Wijnholds and Moortgat [WM21], falling behind by only about two percentage points. For sentiment analysis, the difference between our implementation and the implementation by de Vries et al. [dVvCB+19] is larger, falling behind by about 8 and 12 percentage points for Multilingual BERT and BERTJE respectively. Besides our implementation of the Cothought pipeline, these results can be affected by the difference in models used and how long the models are trained for. As such, the differences in this comparison can not be attributed solely to changes we

made to the CoThought pipeline.

Next, we will compare our implementation to the original implementation of the Cothought pipeline by Zhang et al. [ZYM+23]. They used the GLUE benchmark by Wang et al. [WSM+19] amongst others to benchmark their model. This benchmark also contains some NLI tasks, but no Sentiment Analysis tasks. As such, we will only be comparing our model with theirs for NLI. However, since the benchmarks used by Zhang et al. [ZYM+23] are based on different data sets than ours, the comparison lacks some robustness. The NLI tasks from the GLUE benchmark by Wang et al. [WSM+19] are based on two different datasets:

MNLI
This task is based on Multi-Genre Natural Language Inference (MNLI) Corpus. The MNLI Corpus is a crowd-sourced dataset of sentence pairs annotated for textual entailment. The premises of this dataset are sourced from diverse genres such as transcribed speech, fiction, and government reports [WSM+19].

QNLI
For this task they used the Stanford Question Answering Dataset (SQuAD). They adapted SQUAD into a Question-answering NLI (QNLI) format. QNLI involves determining if a context sentence contains the answer to a question. For this, original question-paragraph pairs from Wikipedia are converted into sentence pairs, filtering out pairs with low lexical overlap. This transformation changes the task to sentence pair classification, removing the requirement to select exact answers and the assumption that answers are always present [WSM+19].

Below you can see the accuracy of the model by Zhang et al. [ZYM+23] compared with our best adapters (again model A3) for NLI.

Task	**Accuracy**
MNLI [ZYM+23]	73.73
MNLI-mm [ZYM+23]	74.76
QNLI [ZYM+23]	76.86
NLI (ours)	81.61

Table 9: Comparison of Accuracy between our adapters and the model by Zhang et al. [ZYM+23]

Here, we achieve a higher accuracy score. This could be due to the fact that our implementation of the CoThought pipeline performs better, because we used a different dataset, but also because we used adapters. Like discussed in section 1.2.1, previous research has shown that using adapters instead of fine-tuning has a small impact on model performance. Using different datasets can influence results in varying ways. For example, having an imbalanced fine-tuning dataset can negatively influence model performance [KKP05].

4.2 Limitations of Current Experiments

Our research has some limitations. First of all, we could only run our experiments once due to time and computational limitations, which means that our results might be affected by an (un)lucky initialization of weights. One possible occurrence of this is model A3 (see figure 2), which we discussed in section 3.2.
Secondly, due to time limitations, we could only fine-tune and test the adapters on two NLU tasks. Further testing would allow us to give a more accurate representation of the performance of our adapters. The adapters might perform worse or better on some specific NLU tasks.
Furthermore, when replacing part of the translated data by originally Dutch data, we replaced only one dataset from the BabyLM challenge which was 0.92 % of the total dataset (see Table 4). In the end, we did not see any large differences between the adapters trained on the dataset with the originally Dutch data and without. It could be the case that using originally Dutch data has no effect on the produced adapters, or the part of the dataset we replaced was too small to notice any significant difference.
Also, because we have only trained adapters instead of pretraining and fine-tuning the whole model we cannot compare our models with fine-tuned models using the same methods, which could give further insights of the performance impact of using adapters for our methods. However, previous research has shown minimal performance impact when using adapters instead of fine-tuning the whole model [PSP+23].
Lastly, because we did not use the same datasets for testing as Zhang et al. [ZYM+23], we could not directly compare the performance of our adapters.

5 Conclusions

In conclusion, this research successfully adapted the CoThought pipeline for training a BabyLM on Dutch data. Our results showed that the impact of changing the prompting strategy used in the CoThought pipeline for training adapters on Dutch data was insignificant, as was the impact of including originally Dutch data in the pretraining data. While the adapters achieved promising results for both Sentiment Analysis and NLI, they performed comparatively better on NLI. However, due to the limited scope of our experiments, there is room to further assess the pipeline's potential for Dutch language processing.

5.1 Further Experiments and Future Work

First of all, we have compiled some suggestions for future work based on the limitation discussed in section 4.2:

- Run the experiments multiple times, to be sure of the results.
- Test our methods on more NLU tasks, to get better insight in the performance of our models.
- Further test the impact of using originally Dutch data could be further studied by pretraining adapters on only originally Dutch data versus using translated data, either for each separate NLU task or for the whole dataset.

- Test how using adapters affects the performance of the models trained using our methods.

Furthermore, here are some additional research directions based on our methods:

- Apply our methods across various languages to evaluate their impact on the results.
- Compare the effectiveness of different large language models when using our methods.
- Investigate the effects of modifying various hyper-parameters during training.
- Investigation into the application of adapters across various models to be able to better compare with other studies and to observe performance disparities among these models.

References

[AS16] Vishal A. and S.S. Sonawane. Sentiment analysis of twitter data: A survey of techniques. *International Journal of Computer Applications*, 139(11):5–15, April 2016.

[BL17] Nikhil Buduma and Nicholas Locascio. *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms*. O'Reilly, 2017.

[BMR+20] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. 2020.

[BZ23] Bastian Bunzeck and Sina Zarrieß. GPT-wee: How small can a small language model really get? In Alex Warstadt, Aaron Mueller, Leshem Choshen, Ethan Wilcox, Chengxu Zhuang, Juan Ciro, Rafael Mosquera, Bhargavi Paranjabe, Adina Williams, Tal Linzen, and Ryan Cotterell, editors, *Proceedings of the BabyLM Challenge at the 27th Conference on Computational Natural Language Learning*, pages 35–46, Singapore, December 2023. Association for Computational Linguistics.

[CB22] Tyler A. Chang and Benjamin K. Bergen. Word Acquisition in Neural Language Models. *Transactions of the Association for Computational Linguistics*, 10:1–16, 01 2022.

[Cla22] Eve Clark. *Language development in the early years*, pages 1–13. 11 2022.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[DL18] Li Deng and Yang Liu. *Deep learning in natural language processing*. Springer, Singapore, 2018 - 2018.

[dVvCB+19] Wietse de Vries, Andreas van Cranenburgh, Arianna Bisazza, Tommaso Caselli, Gertjan van Noord, and Malvina Nissim. Bertje: A dutch BERT model. *CoRR*, abs/1912.09582, 2019.

[dVWN23] Wietse de Vries, Martijn Wieling, and Malvina Nissim. DUMB: A benchmark for smart evaluation of Dutch models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7221–7241, Singapore, December 2023. Association for Computational Linguistics.

[HGJ+19] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019.

[HR18] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018.

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning*. Springer, New York, 2001.

[JM21] Dan Jurafsky and James H. Martin. *N-gram Language Models*. Speech and Language Processing, 3rd edition, 2021. Archived from the original on 22 May 2022. Retrieved 24 May 2022.

[Kal23] Jugal Kumar Kalita. *Machine learning : theory and practice*. CRC Press, Boca Raton, Florida ;, 2023.

[KKP05] Sotiris Kotsiantis, D. Kanellopoulos, and P. Pintelas. Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30:25–36, 11 2005.

[KKYI23] Goro Kobayashi, Tatsuki Kuribayashi, Sho Yokoi, and Kentaro Inui. Transformer language models handle word frequency in prediction head. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4523–4535, Toronto, Canada, July 2023. Association for Computational Linguistics.

[LOG+19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[Met78] CE Metz. Basic principles of roc analysis. *Semin Nucl Med*, 8(4):283–298, 1978. Archived from the original on 2022-10-09.

[MMN+24] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey, 2024.

[Mur12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, Cambridge, 2012.

[NF21] Mitja Nikolaus and Abdellah Fourtassi. Modeling the interaction between perception-based and production-based learning in children's early acquisition of semantic knowledge. In Arianna Bisazza and Omri Abend, editors, *Proceedings of the 25th Conference on Computational Natural Language Learning*, pages 391–407, Online, November 2021. Association for Computational Linguistics.

[Ope19] OpenAI. Better language models and their implications. Archived from the original on 2020-12-19, February 2019. Retrieved 2024-03-25.

[PSG19] Telmo Pires, Eva Schlinger, and Dan Garrette. How multilingual is multilingual BERT? In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4996–5001, Florence, Italy, July 2019. Association for Computational Linguistics.

[PSP+23] Clifton Poth, Hannah Sterz, Indraneil Paul, Sukannya Purkayastha, Leon Engländer, Timo Imhof, Ivan Vulić, Sebastian Ruder, Iryna Gurevych, and Jonas Pfeiffer. Adapters: A unified library for parameter-efficient and modular transfer learning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 149–160, Singapore, December 2023. Association for Computational Linguistics.

[RNSS18] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[RSR+20] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[RWC+19] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[Sem12] Paul Semaan. Natural language generation: An overview. 2012.

[SGM19] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy, July 2019. Association for Computational Linguistics.

[Sko20] Maciej Skorski. Revisiting initialization of neural networks. *CoRR*, abs/2004.09506, 2020.

[TCERPCU23] Juan Terven, Diana M. Cordova-Esparza, Alfonso Ramirez-Pedraza, and Edgar A. Chavez-Urbiola. Loss functions and metrics in deep learning, 2023.

[TH15] A. A. Taha and A. Hanbury. Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool. *BMC medical imaging*, 15:29, 2015.

[Tom03] Michael. Tomasello. *Constructing a language a usage-based theory of language acquisition*. Harvard University Press, Cambridge, Mass, 2003.

[vdBV19] Benjamin van der Burgh and Suzan Verberne. The merits of universal language model fine-tuning for small datasets - a case with dutch book reviews. *CoRR*, abs/1910.00896, 2019.

[vDKSvD23] Bram M. A. van Dijk, Tom Kouwenhoven, Marco R. Spruit, and Max J. van Duijn. Large language models: The need for nuance in current debates and a pragmatic perspective on understanding, 2023.

[vDvDVS23] Bram M. A van Dijk, Max J van Duijn, Suzan Verberne, and Marco R Spruit. Chiscor: A corpus of freely told fantasy stories by dutch children for computational linguistics and cognitive science. 2023.

[WCD+24] Linyi Wang, Xin Chen, Xinghua Deng, Hao Wen, Mengxia You, Weihua Liu, Qianqian Li, and Jianshu Li. Prompt engineering in consistency and reliability with the evidence-based guideline for llms. *NPJ Digital Medicine*, 7(1):41, 2024.

[WCM+23] Alex Warstadt, Leshem Choshen, Aaron Mueller, Adina Williams, Ethan Wilcox, and Chengxu Zhuang. Call for papers – the babylm challenge: Sample-efficient pretraining on a developmentally plausible corpus. *Computing Research Repository*, arXiv:2301.11796, 2023.

[WM21] Gijs Wijnholds and Michael Moortgat. Sick-nl: A dataset for dutch natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics*, Online, April 2021. Association for Computational Linguistics.

[WMC+23] Alex Warstadt, Aaron Mueller, Leshem Choshen, Ethan Wilcox, Chengxu Zhuang, Juan Ciro, Rafael Mosquera, Bhargavi Paranjabe, Adina Williams, Tal Linzen, and Ryan Cotterell, editors. *Proceedings of the BabyLM Challenge at the 27th Conference on Computational Natural Language Learning*, Singapore, December 2023. Association for Computational Linguistics.

[WSM+19] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.

[WWS+23] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. 2023.

[ZYM+23] Zheyu Zhang, Han Yang, Bolei Ma, David Rügamer, and Ercong Nie. Baby's cothought: Leveraging large language models for enhanced reasoning in compact models. 2023.

A Additional Figures

A.1 Prompts

Prompt 3: Changed order

Gebruik de gegeven invoerzinnen om een voorbeeldparagraaf te genereren voor een specifieke NLU-taak en de bijbehorende labels te identificeren.

Jouw uitvoer moet het volgende formaat hebben:

Plan:
Beschrijf in een paar zinnen hoe je de taak zult aanpakken.

Paragraaf:
Genereer een voorbeeldparagraaf op basis van de gegeven invoerzinnen.

Taak:
Specificeer de NLU-taak (bijv. sentimentanalyse, entiteitsextractie). Geef alleen de taaknaam.

Labels:
Geef de alleen labels voor de gegenereerde paragraaf aan (bijv. positief/negatief sentiment, entiteitstypen).

De 5 te gebruiken invoerzinnen zijn: {input}

Prompt 4: Specific NLU tasks

Gebruik de gegeven invoerzinnen om een voorbeeldparagraaf te genereren voor een specifieke NLU-taak en de bijbehorende labels te identificeren.

Jouw uitvoer moet het volgende formaat hebben:

Plan:
Beschrijf in een paar zinnen hoe je de taak zult aanpakken.

Paragraaf:
Genereer een voorbeeldparagraaf op basis van de gegeven invoerzinnen.

Taak:
Specificeer de NLU-taak, kies uit: sentimentanalyse, natural language intference of entiteitsextractie. Geef alleen de taaknaam.

Labels:
Geef de alleen labels voor de gegenereerde paragraaf aan (bijv. positief/negatief sentiment, entiteitstypen).

De 5 te gebruiken invoerzinnen zijn: {input}

B Supplementary Materials

B.1 Code

The code used in this research can be found at:
https://github.com/ThijsGroeneweg/Adapting-the-CoThought-pipeline-for-training-a-BabyLM-on
In this Github repository we also included some documentation on how to use our code. The code in this Github repository consist of python files which are used for the creation of the dataset and generating the NLU tasks, and Jupiter Notebook files which are used for pretraining and finetuning the adapters.

B.2 Datasets

The datasets we generated can be found at:
https://drive.google.com/drive/folders/1EW2bgdH0OVriijsXU6JOG3fZUcjmHjbs?usp=sharing.
For calculating the proportions of each dataset you can find the code in the Github repository mentioned above (section B.1). Each dataset is included as a combined excel file, and seperate files for each part of the BabyLM challenge dataset [WMC+23]. The folders are labeled with the models trained on the dataset as seen in Table 1.

B.3 Adapters

The adapters we trained during this research can be found at:
https://drive.google.com/drive/folders/1Xa83i4y4Ut6d6NBV4W-TlLBNyx1GZZnk?usp=sharing.
This includes the pre-trained adapters, the fine-tuned adapters, and the various metrics calculated during training. The folders are labeled with the model names as seen in Table 1.