CHALMERS
UNIVERSITY OF TECHNOLOGY

Politecnico di Torino

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Informatica
A.a. 2022/2023
Sessione di Laurea Luglio 2023

Relevant Phrase Generation for Language Learners

SpeakEasy: A language model that makes it easy for students to practice their language skills by generating example sentences

Relatori:

Luca Cagliero

Candidati:

Davide Pinti

Master's thesis 2023

Relevant Phrase Generation for Language Learners

SpeakEasy: A language model that makes it easy for students to practice their language skills by generating example sentences

DAVIDE PINTI

Department of Computer Science and Engineering
Politecnico di Torino
Chalmers University of Technology
Torino, Italia 2023

Relevant Phrase Generation for Language Learners
SpeakEasy: A language model that makes it easy for students to practice their language skills by generating example sentences

Relatore: Luca Cagliero, DAUIN

Supervisors at Chalmers: Richard Johansson and Moa Johansson
Co-writer: Edvin Lidholm

Master's Thesis 2023
Department of Computer Science and Engineering
Politecnico di Torino &
Chalmers University of Technology

Typeset in LaTeX
Torino, Italia 2023

Relevant Phrase Generation for Language Learners
SpeakEasy: A language model that makes it easy for students to practice their language skills by generating example sentences
DAVIDE PINTI
Department of Computer Science and Engineering
Chalmers University of Technology and Politecnico di Torino

Abstract

In recent times Artificial Intelligence, Natural Language Processing (NLP) especially, has spread widely. Nowadays, most people use it, either directly or indirectly, and you can find it almost everywhere: from social networks, to generating images based on text prompts, to the automatic grammar checker of written text. In the field of NLP, the generation of text through large language models (LLMs) is becoming more and more dominant, especially since the release of the Transformer in 2017. The objective of this study was to leverage the powerful tools of NLP to generate contextually appropriate English sentences for language learners, when given a specific English keyword as input. Other papers in the field of Intelligent Computer-Assisted Language Learning (ICALL) have generated examples for language learners by either retrieval- or ranking-based models, but this is the first time generative language models have been used in this context. We claim that our model developed in this project, SpeakEasy, is useful for language learners that are trying to learn a new language. The generated sentences have three important characteristics: (1) They are relevant in context to the keyword; (2) They are in "simple" English suitable for language learners; (3) They always include the exact form of the keyword given in the prompt. We achieve this by first fine-tuning a GPT–2 model implemented by HuggingFace to a dataset of human-written sentences specifically tailored to language learners from Tatoeba.org. A decoding algorithm consisting of two steps was implemented. Initially, a shift to the probability distribution the context around the keyword was applied using Keyword2Text for controlled generation. Subsequently, the vocabulary is truncated to only include words that have an information content close to the language model itself, picked up during domain adaptation, using locally typical sampling. The generated sentences are similar to the human-written examples with a MAUVE score of 0.755 and an average cosine similarity of 0.577. Moreover, only 1.83% of sentences generated were identical to entries in the dataset and a sentence took on average 0.45 seconds to be generated. Qualitative human evaluation showed that examples generated by SpeakEasy did not only beat a fine-tuned version of GPT–2 with hybrid top-k and nucleus sampling scheme, but also out competing the human-written sentences with an average rank of 2.13 on a scale from 1 (best) to 4 (worst).

Keywords: Natural Language Processing, Automatic Text Generation, Language Learning, Transformer, GPT–2.

Acknowledgements

Innanzitutto sei bellissima,
Muchas gracias afición, esto es para vosotros.

First of all I have to thank Edvin Lidholm, who collaborated with me on this project and made significant contributions to its successful completion.
I would like also to express mine sincere gratitude to my supervisor Richard Johansson, from the Department of Computer Science and Engineering at Chalmers University of Technology, who has provided guidance, support, and valuable insights throughout the project. His expertise and constructive feedback have been invaluable in shaping the research. I also extend mine appreciation to the examiner, Moa Johansson, for her time and effort in evaluating the work and providing valuable suggestions for improvement.
Additionally, there are also every person that I met during my 10 months in Sweden, that helped me to overcome all the difficulties that going to another country alone has brought. I don't want to start saying names otherwise I'll never finish. Having spent 10 months together, I hope they are aware of how immensely significant they are to me.
There are also all my friends in Turin, that accompanied me during 4 years of my life, like the "sburacchio" group and my university friends which experience together a lot of great and tough moments "ringraziandoli per questa opportunità immensa". With them I spent an amazing period that I will never forget.
Without forgetting the "CRH" group that never leave me alone either if there was a thousand kilometers between us. Every time I come back it feels like Ive never been away.
Clearly my biggest gratitude goes to my family, always supporting me in all my decision and always be there for me.
Last but not least, I want to thank my brother, who helped me a lot with my university life always giving advice and be close to me, even though we may be geographically separated by vast distances.

Davide Pinti, Torino, 2023-06-21

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ANN	Artificial Neural Network
ATG	Automatic Text Generation
BERT	Bidirectional Encoder Representations from Transformers
CBoW	Continuous Bag-of-Words
BLEU	BiLingual Evaluation Understudy
DL	Deep Learning
GloVe	Global Vectors for Word Representation
GPT	Generative Pretrained Transformer
GPU	Graphics Processing Unit
ICALL	Intelligent Computer-Assisted Language Learning
LM	Language Model
LSTM	Long Short-Term Memory
ML	Machine Learning
MLM	Masked Language Modeling
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
NSP	Next Sequence Prediction
PE	Positional Encoding
PPL	Perplexity
ReLU	Rectified Linear Unit
RL	ROUGE–L
RN	ROUGE–N
RNN	Recurrent Neural Network
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
SOTA	State-Of-The-Art

Contents

List of Figures

List of Tables

1

Introduction

This chapter provides an introduction to the research subject. First, the context for the project is presented in Section 1.1, followed by a description of the aim of the study in Section 1.2. Section 1.3 outlines the the research questions that our study aims to address. An overview of the related research in this field can be found in Section 1.4, and the delimitations of the project in Section 1.5.

1.1 Background

In today's era, the pursuit of learning new languages has become widespread due to the increasing demands of travel, work, and personal interests. Consequently, the methods of language acquisition have also evolved. The advent of advanced technology has brought about a revolution, enabling easy access to the internet and a vast array of resources at our fingertips. This progress has led to the development of language learning applications such as Duolingo, Babbel, and others. Here, the field of natural language processing (NLP) comes into play as a crucial component, addressing the need for tools that can generate words and sentences to assist students.

The domain of language learning holds immense significance and is rapidly expanding, with its applications spanning across education, business, and personal development [1]. Among the primary challenges in language acquisition is the ability to comprehend and produce relevant and coherent language within various contexts. Leveraging NLP techniques has the potential to transform the way we teach and learn languages by automating the generation of meaningful phrases and sentences for learners to practice with. This paradigm shift can enhance both the efficiency and effectiveness of language learning endeavours.

By employing a large language model, we aim to create an exemplary sentence generation system for language learners, surpassing existing retrieval or ranking-based methods, mentioned in Section 1.4. One reason to use NLP models to generate sentences exemplifying the use of a word is that the more advanced methods can grasp intricate grammar rules, idiomatic expressions, and cultural nuances, creating diverse and authentic sentences that closely resemble human language. This adaptability allows learners to encounter a wide range of sentence structures and vocabulary. Furthermore, Machine Learning (ML) models are highly scalable. They efficiently handle large volumes of data, enabling them to learn from extensive lan-

guage corpora and generate a plethora of example sentences. This scalability ensures access to diverse and relevant examples. Lastly, ML models facilitate continuous improvement. They can be trained and fine-tuned using user feedback, allowing them to adapt and enhance their sentence generation capabilities over time. By incorporating user preferences, corrections, and linguistic insights, the models become more accurate and better aligned with learners' specific needs and preferences.

1.2 Aim

The aim of this project is to achieve a method of generating English phrases of suitable level for language learners correlated with a specific keyword. A keyword is a single word working as an input for generating the sentences with context around it. Furthermore, these sentences should have some level of "quality", meaning they are not just trivial phrases, but with information relevant to the keyword. To illustrate this one could think of an example where a student wants to practice using the word "*bicycle*". In this case a sentence about riding bicycles, or falling while riding, like "*I learned how to ride a bicycle when I was 6 years old.*", would be useful, but "*It is a bicycle.*" would be quite uninteresting.

1.3 Specification of Issue Under Investigation

The following four research questions will be examined:

- Can an NLP-based system be trained to generate contextually appropriate English phrases and sentences for language learners, when given a specific English keyword as input?
- How can a decoding algorithm be developed to guide the text generation towards suitable phrases and sentences for said learners?
- How can the generated phrases and sentences be evaluated for "quality", in terms of their relevance and usefulness when learning a new language?
- Can the proposed method of generating phrases for language learners be applied in a novel way, beyond what has been previously considered in existing research?

1.4 Related Work

While we are not aware of any previous work that used language models to generate examples from scratch, several projects in the area of intelligent computer-assisted language learning (ICALL) have investigated the use of retrieval from large corpora for finding illustrative examples for purposes such as lexicon development, vocabulary exemplification for learners, and exercise generation. Pilán et al. (2016) [2] gives an overview of work in this area.

Early work includes the GDEX algorithm [3], which retrieves examples to illustrate word use for lexicographers using a rule-based approach. Firstly, the GDEX algorithm prioritizes sentences within the range of 10 to 25 words, penalizing those that are longer or shorter. Moreover, it penalizes sentences containing uncommon or rare words, meaning they are not part of the 17,000 most frequently used English words. Sentences with pronouns and anaphors lacking self-contained meaning[1] receive penalties due to their need for additional context. Furthermore, preferred sentences begin with a capital letter and conclude with a full stop, exclamation mark, or question mark. The authors note that effective examples often introduce contextual information and position the keyword towards the sentence's end, enabling users to infer its meaning.

In ICALL specifically, Pilán et al. (2013) [4] proposed a ranker similar to GDEX for selecting examples suitable for learners. They extended the rule-based ranker by using a machine learning-based approach to classify sentences into their corresponding CERF (Common European Framework of Reference) language levels. Results showed that 70% of the sentences retrieved by the model were deemed to be of an appropriate language level by human evaluators. Furthermore, 60% of the retrieved sentences were suitable as an illustrative example of word usage.

Example selection has also been used in ICALL to generate exercises, typically in the form of cloze questions (fill-in-the-blank) [5], [6]. Some of these approaches used language models in order to rank examples according to the typicality of a word in a context; for instance, Wojatzki et al. (2016) [6] used an N-gram language model for this purpose. Results showed that their model's proposed exercises reduced the ambiguity in the "fill-in-the-blank"-type questions. Furthermore, the authors also introduced a disambiguation measure, which was proven to effectively discard exercises that were too ambiguous.

1.5 Delimitations

This thesis project has several limitations that should be considered. Firstly, it is only focused on English as the target language. This means that the model and decoding algorithm only is able to generate phrases and sentences in English, and does not support other languages.

Another limitation of the project is that it only considers one-word prompts for text generation. This means that the model only generates phrases and sentences based on a single word input, rather than multiple words or whole sentences. This may affect the model's ability to generate contextually appropriate phrases and sentences, as one-word prompts may not provide enough context for the model to generate accurate and meaningful output.

Additionally, we did not develop a way of adapting the level of the sentences towards the student proficiency in the target language. This means that the generated phrases and sentences are not tailored to the specific language level of the student,

[1] E.g., *this*, *that*, *it* or *one*.

thus making the generated phrases and sentences less relevant and useful for some students who are at different stages of language learning.

In summary, this thesis has limitations in terms of the target language, the type of prompts used, and the lack of adaptability to students' prior proficiency level.

1.6 Structure of the Thesis Report

In this thesis, Chapter 2 provides a comprehensive overview of the relevant background information, including the research field, relevant models, and previous studies. The methods employed to address the research question is outlined in Chapter 3, while the results of the study are presented and analyzed in Chapter 4. Finally, in Chapters 5 and 6, the methodology and results are thoroughly discussed along with suggestions for future research directions and conclusion, repectively.

2

Theory

This chapter provides the theoretical framework for the subject matter. In Section 2.1, an overview of NLP and its sub-field automatic text generation (ATG) is presented. Section 2.2 explores the field of deep learning (DL) and introduces commonly used models for NLP tasks, such as recurrent neural networks (RNNs), long short-term memorys (LSTMs), and the Transformer architecture. In Section 2.3, tokenization and word embedding are introduced, while Section 2.4 provides a description of the BERT model with some relevant adaptations of the core model. Section 2.5 provides a background for autoregressive language models used for generating text, finishing by introducing the GPT–2 model used in this project. Additionally, Section 2.6 delves into the algorithms used for decoding, which translate the probabilities generated by the decoder model into text. Finally, in Section 2.7, the tools used to evaluate the results are described.

2.1 Research Area

This section introduces the broader research domains of natural language processing, and the automated generation of text.

2.1.1 Natural Language Processing

NLP is a rapidly growing field that involves the use of computer algorithms to analyze, understand, and generate human language. It is a multidisciplinary field that combines the fields of computer science, linguistics, and cognitive psychology to enable computers to interact with human language [7].

The two primary subfields of NLP are natural language understanding (NLU) and natural language generation (NLG). NLU focuses on enabling machines to comprehend human language by analyzing and processing linguistic structures such as syntax, semantics, and pragmatics [8]. NLG, on the other hand, is concerned with producing language and involves generating human-like text that is coherent, grammatically correct, and semantically meaningful [9].

NLP has many applications and is used in a wide range of fields, including machine translation, text classification, question answering, sentiment analysis, and speech recognition. Machine translation is the process of translating one language into another [10], while text classification involves categorizing text into different classes

or categories [11]. Question answering systems allow machines to understand and respond to human language queries [12], while sentiment analysis involves analyzing the emotional content of text [13].

In recent times, DL has emerged as the most commonly used approach to create NLP models [14], which is a subset of machine learning (ML) that involves the use of artificial neural networks (ANNs) to analyze and process large amounts of data. DL algorithms have been shown to outperform traditional statistical methods in many NLP tasks, and have led to significant advancements in this field [15]. In particular, the use of autoregressive language models (introduced in Section 2.5) with transformer architectures has revolutionized the field of text generation, allowing for the creation of high-quality, human-like text.

Overall, NLP is an exciting and rapidly evolving field that has the potential to transform the way we interact with machines and with each other through language. Its applications are wide-ranging and continue to grow as new technologies and techniques are developed.

2.1.2 Automatic Text Generation

As a subfield of NLG, ATG focuses on generating human-like text automatically. It is a challenging task as it requires the computer to produce text that is coherent, grammatically correct, and semantically meaningful. Previous text generation models have been statistical-based [16], cased-based [17], and rule-based [18]. However, these models often fail to capture the nuances and complexities of human language, resulting in stilted and unnatural text [19].

Recently, the state-of-the-art (SOTA) technology in text generation is using deep learning and autoregressive language models with a transformer architecture [20]. These models are able to generate convincing natural text by training on large amounts of data and learning the patterns and structures of human language. The transformer architecture allows for capturing long-term dependencies and contextual information, leading to better text coherence and fluency [21].

Furthermore, recent advancements in pre-training techniques such as ChatGPT and GPT–4 have shown promising results in generating high-quality text that can mimic human-like writing style and tone [22], [23]. With these new technologies, ATG is becoming an increasingly important area of research, with potential applications in content creation, virtual assistants, chatbots, and more.

2.2 Deep Learning

This section provides an overview of the deep learning field and highlights the commonly employed models for ATG.

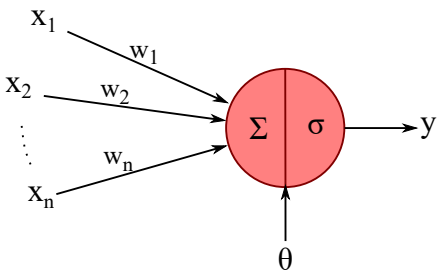


Figure 2.1: Schematic representation of a single neuron in an ANN

2.2.1 Background

Deep Learning is a sub-field in ML that uses ANNs. These networks are inspired by the structure of the human brain and consist of interconnected nodes that exchange information, also called neurons [14], [24]. Each node receives inputs from other nodes and generates a single output, which is then sent to multiple nodes. The connections between neurons are weighted and these weights determine the influence of inputs on the output. ANNs are composed of multiple layers, where deeper layers allows the network to learn more complex patterns. Typically, neurons in one layer are connected to all neurons in the preceding and following layers [25], also called a fully connected multi-layer perceptron. This structure allows DL algorithms to learn abstract representations of data, making it a powerful tool in addressing NLP challenges, such as automatic text generation [26].

Figure 2.1 provides a visualization of a single neuron in a neural network. The inputs $x_1, \ldots, x_n$ coming from neurons in the preceding layer are multiplied by the corresponding weights $w_1, \ldots, w_n$ and then summed up together with a bias, θ. The result of this addition is then passed through an activation function, σ, to produce the output $y = \sigma\left(\theta + \sum_{i=1}^{n} x_i \cdot w_i\right)$ that is transmitted to the next set of neurons in the network [25].

In Figure 2.2, an ANN with one hidden layer of neurons is illustrated. The input layer is responsible for receiving the initial data, while the output layer delivers the model's result. The hidden layer, positioned between the input and output layers, performs the computational operations.

In the training process of an ANN, the weights and biases are iteratively adjusted with the aim of reducing the value of the cost function, which measures the discrepancy between the predicted output of the network and the actual target output. This is typically achieved through the utilization of backpropagation [27], which allows for the calculation of the gradient with respect to each weight based on the difference between the target output and the actual output [28]. In the context of ATG, recurrent neural networks and long short-term memory networks have been widely used in the past, but the current SOTA models follow the Transformer architecture [14]. In the following sections, these various models will be discussed.

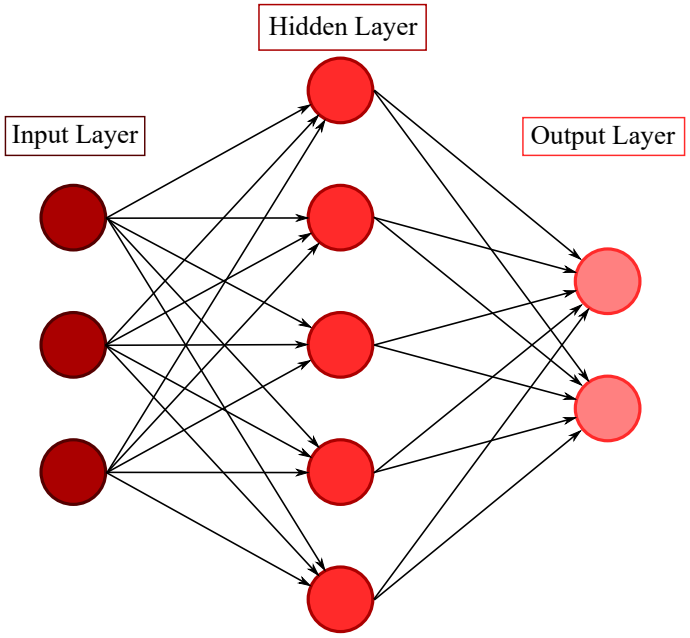


Figure 2.2: A multi-layer perceptron neural network with one hidden layer of five neurons.

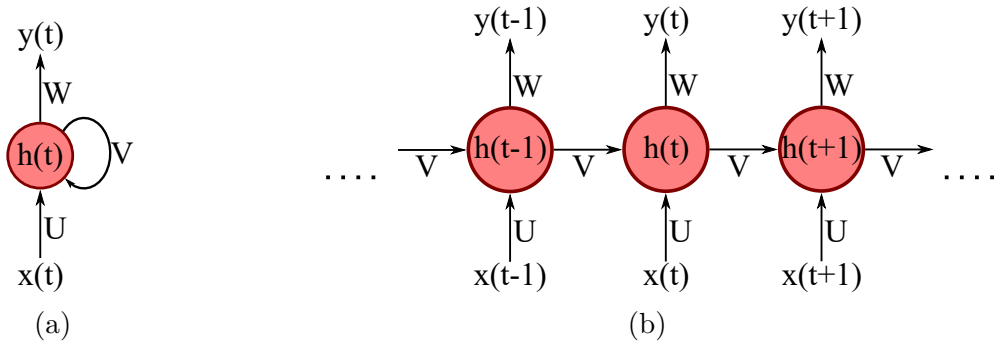


Figure 2.3: a) shows a recurrent neural network and b) shows the same RNN unfolded in time.

2.2.2 RNN and LSTM

An RNN is a type of neural network that utilizes feedback loops to incorporate information from previous time steps into its computation. This structure is depicted in Figure 2.3, where U, V and W are edge weights; And $x\,(t)$ and $y\,(t)$ are input and output states, respectively. At each time step, the hidden state $h\,(t)$ is passed back into the network to be processed again, enabling the network to handle inputs of varying length and to preserve the order of input sequences, which is crucial in the field of ATG where sentences or word sequences have variable length and the arrangement of words affects their meaning [26]. However, RNNs face challenges in dealing with long-term dependencies, such as gradient vanishing or gradient explosion, where the gradient can become very small or very large over time [29].

An LSTM unit is a type of RNN that utilizes gates to control the memorization

process. This unit has three distinct gates, the input, output, and forget gate, which regulate the flow and modification of information through the neuron. This advancement in RNN architecture addressed the issues of vanishing gradients and long-term memory loss [30], and is depicted in Figure 2.4. c_t, h_t and x_t are the cell-, hidden- and input states at time t, respectively.

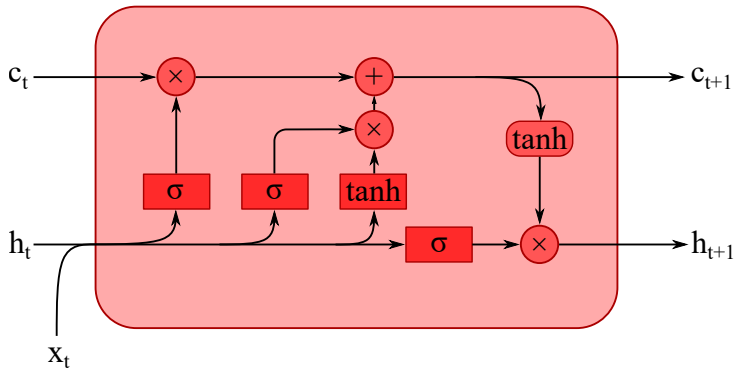


Figure 2.4: Schematic representation of the LSTM architecture.

2.2.3 Transformer Architecture

Vaswani et al. introduced the Transformer network [31] as a type of deep neural network that has become the SOTA model for language modeling and NLP tasks [20]. This model allows for greater parallelization, making efficient use of modern GPU capabilities for parallel computation. The Transformer architecture has proven effective in automatic text generation as well. Figure 2.5 depicts the Transformer architecture as presented by Vaswani et al. (2017), and the subsequent paragraphs provides a detailed description of the various layers of the model, based on the original publication.

2.2.3.1 Encoder

This section describes the encoder block, the components in the left part of Figure 2.5.

Input Embedding: This component involves converting the input sequence of words into *embeddings*, which are numerical vectors that represent each word in the sequence. The embeddings are created in such a way that words with similar meanings are represented by similar vectors. In the paper [31], 512-dimensional embedding vectors are used, but other implementations may use embeddings of different dimensions.

Positional Encoding: This step is added to encode the position of each word in the input sequence, which is essential for NLP tasks as word order conveys meaning, and this information is not naturally preserved when computations are done in parallel. Vaswani et al. achieves this by generating a 512-dimensional vector for each position in the sequence as in Equation 2.1 [31].

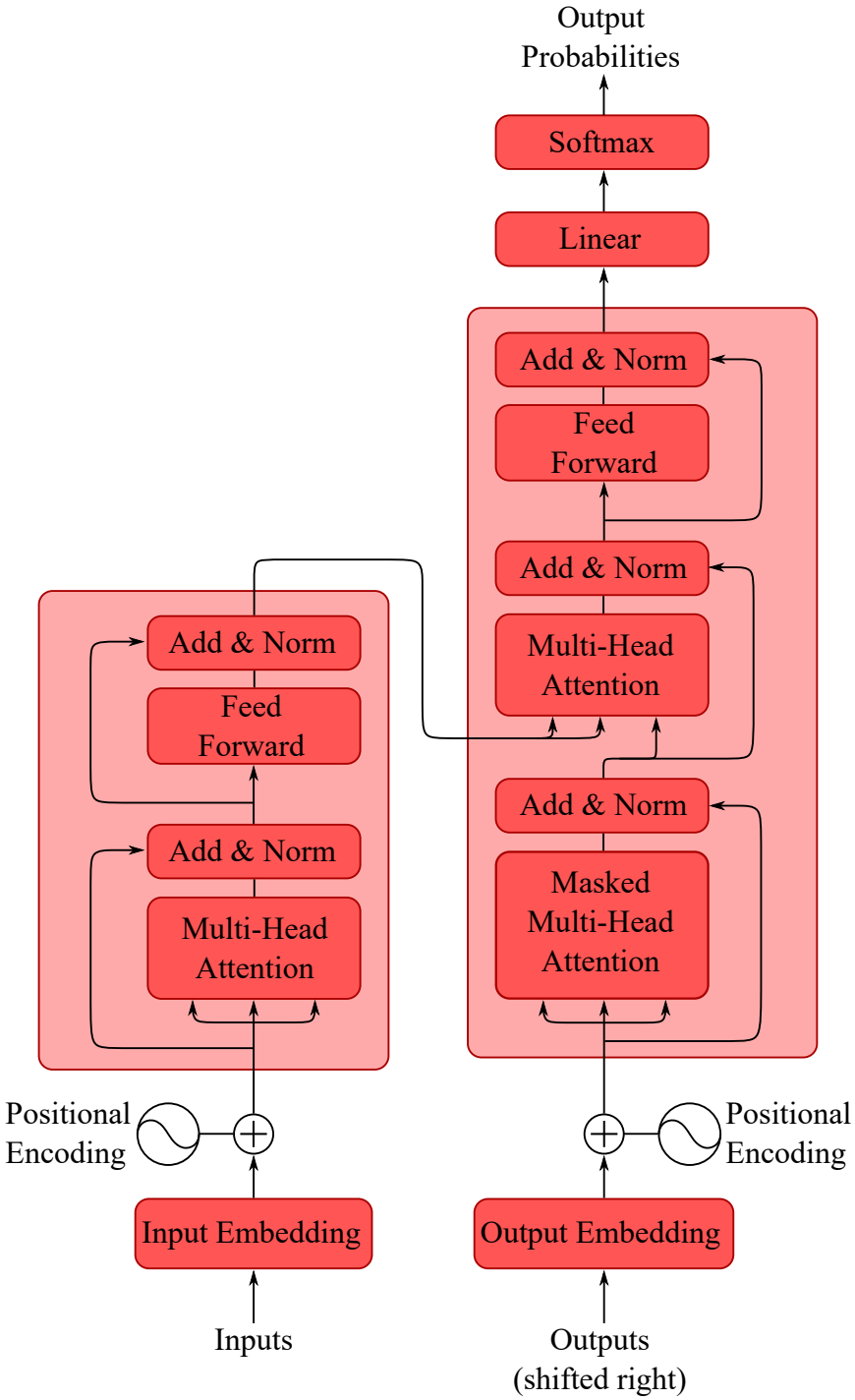


Figure 2.5: The Transformer model architecture according to the paper by Vaswani et al. (2017) [31].

$$\begin{cases} PE_{pos,2i} = \sin\left(pos/10,000^{2i/d_{model}}\right) \\ PE_{pos,2i+1} = \cos\left(pos/10,000^{2i/d_{model}}\right) \end{cases} \tag{2.1}$$

Multi-Head Attention: The self-attention mechanism is a crucial part of the Transformer architecture that helps the model to understand the relationships between words in a sentence. Specifically, this mechanism allows the input sequence to attend to itself. With the help of trainable weight matrices W_i^Q, W_i^K, W_i^V and the input X the model obtains three representations of the same word for each attention-head i as such:

- Queries: $Q_i = XW_i^Q$
- Keys: $K_i = XW_i^K$
- Values: $V_i = XW_i^V$

From these vectors you calculate the self-attention by multiplying the queries and keys vectors and dividing by the square root of their dimension. The softmax is then applied and the values vector is multiplied to the result, see Equation 2.2 [31].

$$Attention\left(Q_i, K_i, V_i\right) = softmax\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i \tag{2.2}$$

To pass the attention scores through the feed-forward network the model concatenates the output of all heads and multiplies by an output weight matrix W^O, as

$$MultiHead\left(Q, K, V\right) = concat\left(head_1, head_2, \ldots, head_h\right) W^O, \tag{2.3}$$

where

$$head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right). \tag{2.4}$$

Add and Norm: The word embeddings from the previous layer are added with the output from the Multi-Head Attention mechanism and normalized to ensure that the mean and variance of each word representation are standardized to 0 and 1, respectively [31].

Feed Forward: The feed-forward network is composed of two linear layers separated by a ReLU activation function [32]. The output of this layer is added to its input, and then the result is normalized using a new Add & Norm layer as above [31].

2.2.3.2 Decoder

This section describes the components in the right part of Figure 2.5, the decoder block. Only the layers that differ from their respective counterpart in the encoder are described.

Masked Multi-Head Attention: The decoder of the Transformer architecture generates the output sequence word-by-word, and its output at each step should only depend on the previous words in the sequence. To achieve this, the decoder applies a mask to the future tokens. This mask is a matrix, M, filled with 0's for the unmasked tokens and $-\infty$ for the masked tokens.

During the computation of the attention score, masking is performed to serve two purposes. Firstly, masking is used to zero out attention outputs where there is padding in the input sentences, to prevent padding from contributing to self-attention. Secondly, it is used to prevent the decoder from "peeking" ahead at the rest of the target sentence when predicting the next word. To achieve this, the decoder masks out input words that appear later in the sequence. The masked elements are set to negative infinity just before the softmax calculation, see Equation (2.5), to ensure that softmax turns those values to zero [31].

$$MaskedAttention\,(Q, K, V) = softmax\left(\frac{QK^T + M}{\sqrt{d_k}}\right)V_i \tag{2.5}$$

Encoder-Decoder Attention: In the decoder block's self-attention layer, the encoder output serves as both the values and keys, whilst the Masked Multi-Head attention outputs function as queries. This allows the decoder to focus on the relevant encoder inputs, leading to a more efficient and precise decoding process. The encoder-decoder attention mechanism is similar to that of sequence-to-sequence models and enables every decoder position to attend over all positions in the input sequence [31].

Linear: The Linear layer in the Transformer architecture is responsible for projecting the decoder output vector into word scores, with a score value assigned to each unique word in the target vocabulary, at every position in the sentence. This means that for an output sentence with n words and a target vocabulary with V unique words, we generate V score values for each of the n words. These score values indicate the likelihood of occurrence for each word in the vocabulary at that specific position of the sentence. Essentially, the Linear layer acts as a classifier, producing a vector of the same length as the vocabulary [31].

Softmax: At the end of the decoder block, there is a layer that produces a probability distribution for each token in the vocabulary, with each token assigned a score between 0 and 1 (which all add up to 1.0). This is accomplished by applying the softmax function to the output of the previous linear transformation layer [31].

2.2.3.3 HuggingFace

HuggingFace is an open-source NLP technologies company that provides a platform for researchers to upload pre-trained models for public use. The *Transformers* library[1], created by HuggingFace, contains various well-known Transformers archi-

[1]The library is available at: `https://github.com/huggingface/transformers`.

tecture variants [33]. The library's primary objective is to make pre-trained models more accessible and more comfortable to read, develop, and deploy. The community around HuggingFace has contributed thousands of fine-tuned models that researchers can download and use for research and educational purposes.

Researchers can download a model's architecture and pre-train it from scratch or download pre-trained models by defining a checkpoint, which contains the model's current state of weights and tokenizer.

In this thesis, we use the GPT–2 model, one of the models implemented in the Transformers library, as an autoregressive language model. The library also contains tokenizers that handle the specific encoding and decoding of tokens for each model. Our specific model is "`GPT2LMHeadModel`", which is a GPT–2 implementation specifically for language modeling.

2.3 Word Tokenization

In NLP, tokenization is a crucial preprocessing step that breaks down text into smaller pieces, or *tokens* [34]. The tokenization process can group tokens into words, subwords, or even as granular units as characters, depending on the chosen *tokenizer*, and studies suggest that these strategies can significantly affect model performance [35], [36]. Tokens are mapped to token-IDs and tracked in a vocabulary.

While word- and character-based tokenization methods are relatively straightforward to grasp, they have some issues. Word-based tokenization can lead to unmanageably vocabularies, as every word has its own token. To combat this issue, a vocabulary of only the most common words can be created, with an "`unknown`" token assigned for words not included. However, this approach can lead to a loss of performance as information is lost for every used unknown token. Furthermore, semantically similar words (e.g. "*bicycle*" and "*bicycles*") can have different representations in the model, and will therefore falsely be treated as completely separate entities.

While character-based tokenization mitigates the problem of large vocabularies, it fails to capture the emergent properties of language where a single character may not carry as much meaning as a word[2]. Therefore, the model has to look at several tokens to interpret the meaning of a word. Additionally, character-based tokenization requires the model to handle larger inputs, as a word-based input of only a small number of tokens can be split up into a large number of characters.

The subword-based tokenization method is a combination of the word- and character-based approaches and is the most common method used by current SOTA models in NLP [37]. This method splits uncommon words into subwords while leaving common sequences of characters intact. The model can build every word in the document by stitching together the subwords. Subword-based tokenization can learn pre- and suffixes and grammatical word endings, allowing the model to see the similarity between the aforementioned singular contra plural example [38]. However,

[2]For example Japanese Kanji characters can carry a lot of meaning, but this is mostly true for languages using the Latin alphabet (e.g, English).

the partitioning of subword tokens depends on the data used to train the tokenizer and may not always result in an optimal partition.

The WordPiece tokenizer is a common approach that aims to express the input corpus with a fixed-size vocabulary [37]. If a word is not found in the vocabulary, it is split into subwords in such a way as to minimize the number of tokens needed. However, this recursive process of splitting can result in excessive splitting and significantly lengthen input text if it is dense in out-of-vocabulary words and subwords.

2.3.1 Word Embeddings

Word embeddings are used in NLP models to represent words as real-valued vectors instead of one-hot encodings, which are often insufficient for deeper understanding [39]. The goal of word embeddings is to map words with semantic similarities to similar vectors in a high-dimensional space, where distance between vectors indicates the degree of semantic relatedness between words and cosine similarity as the measure of closeness (see Section 2.7.1). This is not possible with one-hot encodings because each word is represented by a binary vector with identical distance to every other word. The larger the dimension of the space, the better the representation, but the trade-off is that it requires more data or is slower to train. By using word embeddings, NLP models can capture semantic relationships between words, enabling them to perform more accurate and sophisticated language processing tasks.

2.3.1.1 Word2Vec

Word2Vec is an embedding model which uses a neural network comprising one hidden layer and can be trained using two different methods. Developed by Mikolov et al. (2013) [40], [41] at Google, this algorithm is used to predict words close to the word to be embedded via a two-layer neural network, the structure of which is illustrated in Figure 2.6, and then uses the obtained parameters as embeddings.

The first method used is the Continuous Bag-of-Words (CBoW) method, which learns the word embeddings by implementing a context window around a word, and then the network tries to predict the word in question. Similarly, the Skip-gram method also uses a context window, but trains the model to predict the surrounding words in the training set.

2.3.1.2 GloVe

GloVe (Global Vectors for Word Representation) is a well-known algorithm for obtaining word embeddings from large text corpora [42]. Unlike other techniques such as Word2Vec, GloVe not only considers local context-based information but also leverages global co-occurrence information to create embedding vectors that capture both semantic and syntactic properties of words. The training of the model is done by constructing a co-occurrence matrix that stores the frequency of adjacent words occurring together in a given window size for every word in the training corpus.

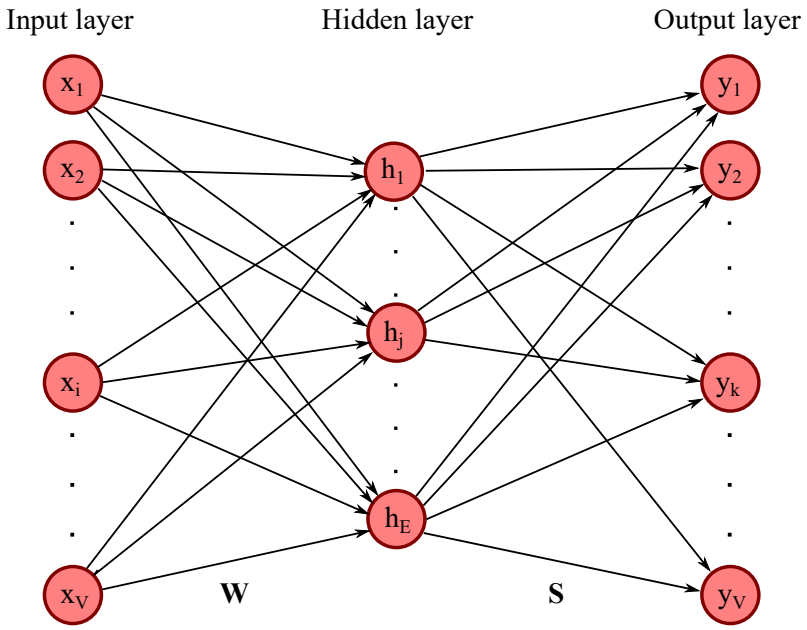


Figure 2.6: Structure of the Word2Vec network, where W and S are $V \times E$ and $E \times V$ matrices, respectively.

One of the distinctive features of GloVe is its use of matrix factorization to optimize the vector representations of words by minimizing a loss function that compares the dot product of two word vectors and the logarithm of their co-occurrence probability. GloVe's resulting word embeddings are applicable to various NLP tasks, such as text classification [43], information retrieval [44], and machine translation [45], [46].

GloVe embeddings are calculated by first letting X_i be the window size times the amount of times word v_i occurs in the training corpus. Furthermore, let $X_{i,j}$ denote the number of times that v_j occurs in a window of the given size around word v_i. By defining $p_{j|i} = X_{i,j}/X_i$, GloVe deems words v_i and v_j to be "close" if the ratio $p_{k|i}/p_{k|j}$ is close to 1 for most words v_k.

Similarly to Word2Vec, GloVe also uses target vectors, t_j, and context vectors, c_j. They are trained by fitting a parametric model, with the target- and context vectors as parameters:

$$t_i^{\mathrm{T}} c_k - t_j^{\mathrm{T}} c_k = \log \frac{p_{k|i}}{p_{k|j}} = \log \frac{X_{i,k}/X_i}{X_{j,k}/X_j}. \tag{2.6}$$

Equation (2.6) holds if $t_i^{\mathrm{T}} c_k = \log X_{i,k} - \log X_i$. Next, two biases b_j and $b_j^{(c)}$ are introduced for each j such that $\log X_i + \log X_k = b_i + b_k^{(c)} + b_k + b_i^{(c)}$. This works if $t_i^{\mathrm{T}} c_k + b_i + b_k^{(c)} = \log X_{i,k}$. Now the embeddings are calculated by picking target- and context vectors so as to minimize the double sum,

$$J = \sum_{i,k=1}^{V} f\left(X_{i,k}\right) \left(t_i^{\mathrm{T}} c_k + b_i + b_k^{(c)} - \log X_{i,k}\right)^2, \tag{2.7}$$

where they weighting function $f(\cdot)$ is introduced to mitigate the logarithm blowing up if $X_{i,k} = 0$, and has the following three properties:

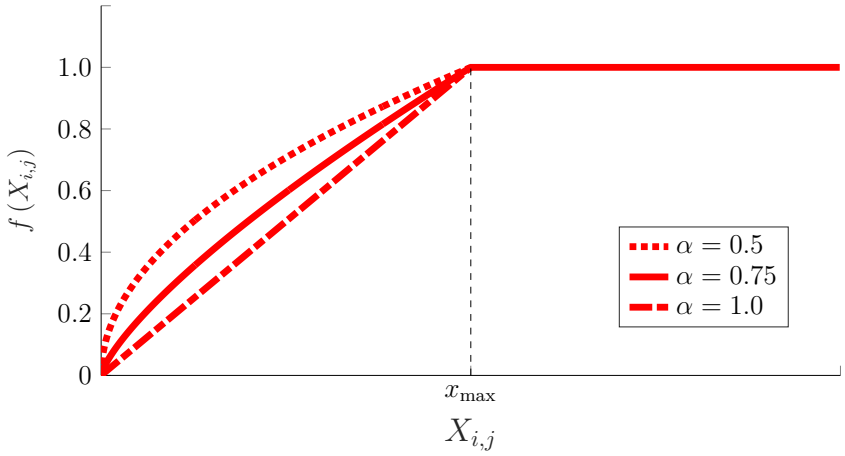


Figure 2.7: Weighting function of the GloVe word embedding model for various values of α, but with fixed $x_{\max}$.

- $f(0) = 0$ and it vanishes faster than $\log x$ as $x \to 0^+$.
- It is non-decreasing.
- $f(x) = 1$ when $x > x_{\max} > 0$ after some cutoff $x_{\max}$.

In their paper, Pennington et al. (2014) use a weighting function parameterized as,

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise,} \end{cases} \tag{2.8}$$

with values $x_{\max} = 100$ and $\alpha = 3/4$ [42]. An illustration of how Equation (2.8) behaves for different values of α can be found in Figure 2.7.

2.3.1.3 FastText

Developed by Facebook's AI Research team [47], FastText is a word embedding algorithm that extends the previously mentioned Word2Vec model. FastText represents words as bags of character N-grams. This approach enables the model to capture morphological details and effectively handle out-of-vocabulary words. This stands in contrast to Word2Vec and GloVe, which fail to provide vector representations for words outside their pre-defined dictionaries.

By representing words using character N-grams, fastText gains the ability to capture the meaning of shorter words and understand suffixes and prefixes. This model can be seen as a bag of words with a sliding window over each word, where the order of the N-grams within the window is not significant. Each word is represented by itself and all sub-N-grams for $K_1 \leq \mathrm{N} \leq K_2$[3]. Pre-processing of any word is made by enclosing the word in brackets (e.g., "`dogs`" becomes "`<dogs>`"). The full representation of *dogs* can be found in Table 2.1. The training process involves employing a skip-gram model that learns embeddings based on these character N-grams.

[3]Usually $K_1 = 3$ and $K_2 = 6$.

	sub-N-grams
N = 3	<do, dog, ogs, gs>
N = 4	<dog, dogs, ogs>
N = 5	<dogs, dogs>
N = 6	<dogs>

Table 2.1: Character sub-N-grams in the FastText representation of the word *dogs* with $3 \leq N \leq 6$.

2.4 BERT

Bidirectional Encoder Representations from Transformers (BERT), is a language model that was developed by Google in 2018 [48]. Its architectural design closely resembles that of the Transformer encoder block, illustrated in the left part of Figure 2.5. BERT has revolutionized the field of NLP as it achieves remarkable results on various benchmarking tests in the NLP domain [48]. The bidirectional aspect of BERT allows the model to consider both preceding and succeeding positions when encoding information.

The initial pre-training of the BERT language model involves training on a large unlabeled corpus, and subsequent fine-tuning enables it to be tailored for specific downstream NLP tasks with minimal architectural modifications. This fine-tuning process requires less data and training compared to the initial pre-training phase, making BERT highly efficient and adaptable to new tasks after the initial language modeling is performed.

BERT is pre-trained on two innovative language tasks: Masked Language Modeling (MLM) and Next Sequence Prediction (NSP). In MLM, a percentage of tokens in an input sequence are replaced with masking tokens, and the model is trained to predict the original tokens. During BERT's pre-training, approximately 15% of tokens were masked. NSP, on the other hand, involves predicting the relatedness of two sentences, facilitating a better understanding of sentence relationships in tasks like question answering. Furthermore, during the pre-training phase, BERT also learns positional embeddings, which enable the model to internalize the position of words within a sequence [48].

One of the key advantages of BERT is its ability to capture the context-dependent meaning of words in a sentence. This means that the BERT embeddings of two sentences that are similar in meaning will be closer together in the embedding space compared to embeddings of two sentences that are not similar in meaning. As a result, BERT embeddings have become an important tool for text comparison tasks [48].

2.4.1 Sentence-BERT

Sentence-BERT (SBERT) is an extension of the BERT model that uses a Transformer-based encoder to create fixed-length vector representations for sentences [49]. SBERT

employs siamese or triplet network architectures, described in further detail in Section 2.4.1.1, to acquire sentence embeddings that capture semantic similarity or relatedness between sentences.

SBERT produces a concise and fixed-length vector representation, known as a sentence embedding, for each input sentence. These embeddings effectively encode contextual information, encapsulating the semantic essence of the sentences. SBERT's versatility extends to various applications, including semantic similarity score computation and clustering of similar sentences [49], [50].

Thorough evaluations on diverse sentence semantics tasks have proven SBERT's superior performance when compared to previous methods of sentence embedding. Additionally, SBERT achieves this enhanced performance while maintaining computational efficiency, enabling its application across a wide range of tasks [49].

2.4.1.1 Siamese and Triplet Network Architecture

The siamese and triplet network architectures in NLP are neural network models designed to compare and measure the similarity between two input sequences [51]. They are commonly used for tasks such as text similarity detection, paraphrase identification, and sentiment analysis. The name "*siamese*" comes from the idea that the architecture consists of two identical sub-networks. Each arm processes one of the input sequences independently and produces a fixed-length representation of the input. These representations are then compared to determine the similarity between the two sequences [52].

The triplet network architecture is based on the concept of triplets, which consist of three input sequences: an anchor, a positive example, and a negative example. The goal of the model is to learn representations in such a way that the anchor and positive example are closer to each other in the representation space, while the anchor and negative example are farther apart [53], [54].

2.4.2 KeyBERT

KeyBERT is another extension of the BERT model. It is used as to extract the most relevant words from a sentences or document, called keywords. KeyBERT uses SentenceBERT for sentence embedding of the input and computes the cosine similarity between each word embedding and the embedding of the entire sentence. The highest scoring words are deemed as the most characteristic of the sentence as a whole. After calculating similarities KeyBERT returns the k most important words, determined by the hyperparameter k [55].

2.5 Autoregressive Language Models

Language models aim to capture the probability distribution of generated text [56]. The main objective is to compute the probability $P(X)$ for a given text $X = x_1, x_2, \ldots, x_m$, where every x is a word in the vocabulary, and leverage this probability model to generate text.

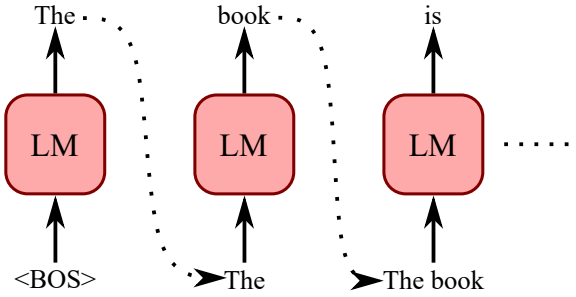


Figure 2.8: Illustration of autoregressive text generation from a language model, where the input to the model at time step t is the sequence of previously generated tokens.

The language model outputs a probability distribution over the vocabulary, representing the likelihood of each word in the vocabulary being the next word in the sentence. This distribution is obtained by applying the chain rule of probabilities where the likelihood of generating word sequence X can be expressed mathematically using Equation (2.9), where x denotes a word in the dictionary and $X_{<t} = x_1, \ldots, x_{t-1}$ represents the sequence of words generated up to the current time step, t.

$$\begin{aligned} P(X) =& P(x_1) \cdot P(x_2|x_1) \cdot P(x_3|x_1, x_2) \cdot \ldots \cdot P(x_m|X_{<m}) = \\ =& \prod_{t=1}^{m} P(x_t|X_{<t}) \end{aligned} \tag{2.9}$$

In this context, *autoregressive* refers to the fact that the model applies itself to its own outputs from previous steps, as is illustrated in Figure 2.8, allowing for sequential generation. The autoregressive language model, following the previously described left-to-right decomposition of text probability, generates the continuation token by token using a specific decoding strategy. The decoding algorithm determines the manner in which words are sampled from the probability distribution and several commonly employed decoding strategies are described in Section 2.6.

To train these language models, raw text data is used, and the training process involves maximizing the likelihood, which is equivalent to minimizing the cross-entropy loss

$$\mathcal{L}(X) = \sum_{t=1}^{m} -\log P(x_t|X_{<t}). \tag{2.10}$$

When training a model that depends on its own predictions a common approach is to use teacher forcing, where only the gold-standard reference is used as input, regardless of previous outputs. By doing so, the model learns to make its next prediction based on the correct input available in the training data, rather than using the potentially erroneous prediction from the previous time step. This approach prevents the model from being trained on inaccurate predictions. Figure 2.9 provides an example demonstrating the application of teacher forcing in practice. Note that the third input to the model is not based on the words the model has output so far, contrary to the process in Figure 2.8 [57].

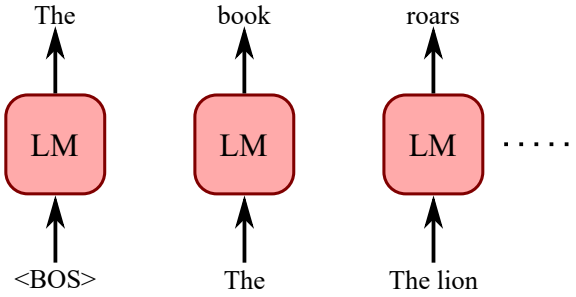


Figure 2.9: Example of teacher forcing where the gold-standard reference sentence used is "*The lion hunts...*".

To ensure effective generation, autoregressive language models are trained on massive datasets, with models like GPT–3 being trained on a staggering 400 billion tokens. This extensive training allows the model to capture rich linguistic patterns and generate high-quality and contextually relevant text [58].

2.5.1 Transfer Learning

The technique of transfer learning, or fine-tuning, involves utilizing a model that is pre-trained on a *source dataset* to solve a related problem by further training it on a *target dataset* [59]. The aim of transfer learning is to leverage the knowledge gained during pre-training, thereby reducing the amount of data required for training the model in the target domain. This results in reduced time and computational resources compared to training the model from scratch [60]. Currently, fine-tuning existing language models is considered a leading method in NLP and language modeling [61].

2.5.2 GPT–2

This section provides an overview of the GPT–2 model developed by OpenAI and the HuggingFace Transformers library utilized to perform fine-tuning.

2.5.2.1 Background and Description

Introduced in 2018, the Generative pre-trained Transformer (GPT) is a large language model developed by OpenAI [56]. It was designed as a semi-supervised model, utilizing both unsupervised pre-training and supervised fine-tuning stages. Unlike previous models that relied solely on supervised learning, GPT's semi-supervised approach allowed it to perform well on datasets that were not well-annotated and to train extremely large models more efficiently [56], [62]. GPT's architecture was based on a twelve-layer decoder-only transformer with twelve masked self-attention heads, and it used the Adam optimization algorithm [56], [63].

GPT was able to achieve robust transfer performance across diverse tasks due to its use of a transformer architecture, which provided a more structured memory than could be achieved through previous techniques. During transfer, GPT utilized task-specific input adaptations derived from traversal-style approaches that process

structured text input as a single contiguous sequence of tokens. Despite not being specifically tailored to individual tasks, GPT was able to outperform discriminatively trained models with task-oriented architectures on a variety of language processing tasks [56].

GPT–2 [64] was created as a scale-up of GPT, with its parameter count and dataset size increased by an entire order of magnitude. Much like the original GPT, GPT–2 was an unsupervised transformer model trained to generate text by predicting the next word in a sequence of tokens. GPT–2 was trained on a dataset of 8 million web pages and has 1.5 billion parameters.[4] It was evaluated on its performance on tasks in a zero-shot setting, meaning it was not specifically trained for these tasks [64].

The use of the Transformer architecture enabled GPT-series models to be trained on larger corpora than previous NLP models due to the ability to parallelize and self-supervize the training process. GPT–2 was trained on a new corpus, known as WebText, which was generated by scraping only pages linked to by Reddit posts that had received at least three upvotes prior to December 2017. The corpus was subsequently cleaned by parsing HTML documents into plain text, eliminating duplicate pages, and removing Wikipedia pages (since their presence in many other datasets could have induced overfitting) [64].

2.6 Decoding Algorithms

The autoregressive language model described in the previous section will output a probability distribution over the vocabulary $\mathcal{V}$, which represents the likelihood of each word in the vocabulary being the next word in the sentence. Figure 2.10 illustrates a toy example of such a distribution for a vocabulary consisting of only five words. The way in which words are sampled from this distribution is the decoding algorithm.

2.6.1 Standard Approaches to Decoding

This section introduces the standard strategies used when generating text from a neural LM, such as greedy decoding, beam search, nucleus- and top-k sampling, and temperature.

2.6.1.1 Maximization-Based Decoding

Greedy decoding implies selecting the highest-scoring word at each step of the generation process, as described in Algorithm 1 [65]. While this approach is computationally efficient, with a time complexity of $\mathcal{O}\left(|\mathcal{V}| \cdot T\right)$[5], and can work well for short sequences, it can lead to problems with longer ones. One issue with greedy search is that it only considers the highest conditional probability for each token in the vocabulary, which can result in suboptimal output sequences. This is because

[4] There also exist smaller versions with 117, 345, 762 million parameters, respectively.

[5] Where $|\mathcal{V}|$ is the size of the vocabulary and T is the maximum sequence length.

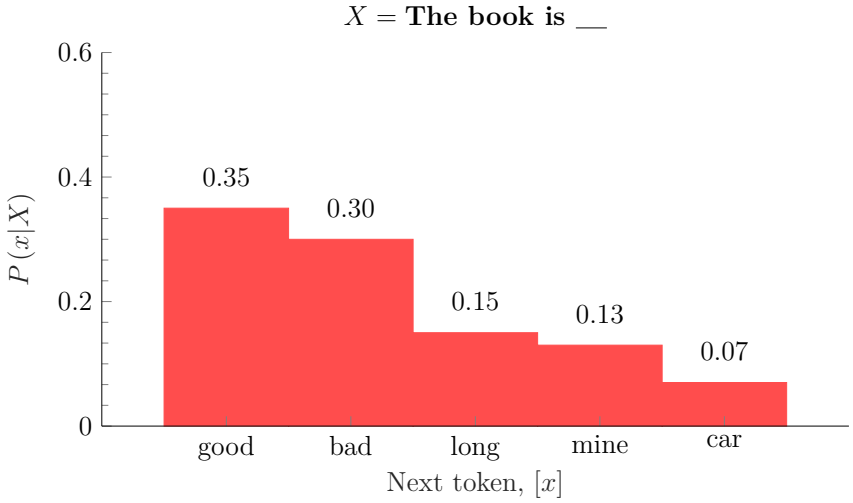


Figure 2.10: Probability distribution over a vocabulary consisting of five words given a starting prompt, X.

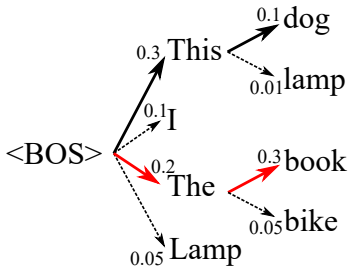


Figure 2.11: Example of a sequence generation where a greedy decoding strategy yields a suboptimal output. The "Beginning of Sequence"-token, `<BOS>`, tells the model to start generating text.

greedy search hides high probabilities that may be found in subsequent tokens. Figure 2.11 provides and illustration of this drawback where the algorithm generates the sequence "*This dog*" (marked in bold with a total score of $0.3 \times 0.1 = 0.03$), while the highest-scoring sequence in actuality is "*The book*" (marked in red with a total score of $0.2 \times 0.3 = 0.06$).

Algorithm 1 Pseudocode for greedy decoding

Let $X = x_1, \ldots, x_m$ be some initial token sequence
for ($i = m + 1, \ldots$ until stopping criterion is met) **do**
  $x_i \leftarrow \arg\max_x P(x|X)$
  append x_i to X
end for
return X

Beam search improves upon greedy decoding by considering multiple potential next words at each step and selects the top N candidates based on their likelihood, as described in Algorithm 2. The number of options considered is the number of

"beams" used in the search, i.e., the *beam width*. This allows for exploration of multiple levels of the output and assessment of the quality of all of these tokens combined [66]. Illustratively, a beam search with $N = 2$ would find the optimal solution to the example in Figure 2.11, where a greedy solution fell short.

However, beam search can lead to repetitive and uninformative text if it degrades into selecting the most probable option repeatedly [67]–[69]. Furthermore, it does not guarantee finding the output sequence with the highest score [70]. Increasing the beam size improves the quality of the output sequence, but at the cost of reduced decoder speed, since the computational complexity $\mathcal{O}(|\mathcal{V}| \cdot T \cdot N)$ increase linearly with the beam width, N. Additionally, there is a saturation point where further increase in beam size does not improve the quality of decoding anymore [71].

Algorithm 2 Pseudocode for beam search

Let N be the beam width
Let $X = x_1, \ldots, x_m$ be some initial token sequence
$B \leftarrow [X]$
for ($i = m + 1, \ldots$ until stopping criterion is met) **do**
    $C \leftarrow [\,]$
    for (each $b \in B$) **do**
        compute $P(x|b)$
        add $b + [x]$ to C for all x in the vocabulary, $\mathcal{V}$
    end for
    $B \leftarrow$ select N top-scoring candidates from C
end for
return top-scoring beam in B

2.6.1.2 Random Sampling

Sampling is a stochastic process where the next word/token is selected randomly based on the probabilities, as described in Algorithm 3. Deterministic methods such as greedy decoding and beam search have a problem of repetition and blandness, respectively, and random sampling offers a trade-off between coherence and diversity. However, this method can lead to incoherent outputs due to excessive randomness, as there is no guarantee that the words will fit together.

Algorithm 3 Pseudocode for random sampling

Let $X = x_1, \ldots, x_m$ be some initial token sequence
for ($i = m + 1, \ldots$ until stopping criterion is met) **do**
    $x_i \sim P(x|X)$
    append x_i to X
end for
return X

When sampling from a large vocabulary, the probability of each token becomes small, and the possibility of selecting a low-probability token is not negligible. If the

selected token is not suitable, the subsequent text generated may become nonsensical, which is why sampling from only a truncated subset of the vocabulary distribution is preferred.

Nucleus sampling was proposed by Holtzman et al. (2020) [67]. Instead of sampling from the entire vocabulary, this approach considers a subset called the top-p vocabulary, which is defined as the smallest set of tokens with cumulative probability mass exceeding a pre-determined threshold p. More formally, the truncation set, $\mathcal{V}^{(p)} \subseteq \mathcal{V}$ is the solution to the optimization problem in Equation (2.11). The probability distribution is then re-scaled with regards to this smaller set, from which the next word is sampled. E.g., nucleus sampling with $p = 0.6$ applied to the example distribution in Figure 2.10 would result in the truncation set $\mathcal{V}^{(p)} = \{\text{good}, \text{bad}\}$.

$$\begin{aligned} \min_{\mathcal{V}^{(p)}} \quad & \left|\mathcal{V}^{(p)}\right| \\ \text{s.t.} \quad & \sum_{x \in \mathcal{V}^{(p)}} P\left(x|X\right) \geq p, \end{aligned} \tag{2.11}$$

The size of the truncated vocabulary is determined dynamically based on the shape of the probability distribution at each time step. For high values of p, the top-p vocabulary consists of a small subset of the vocabulary that contains the vast majority of the probability mass [67]. Furthermore, applying nucleus sampling has a computational complexity of $\mathcal{O}\left(|\mathcal{V}| \cdot \log|\mathcal{V}|\right)$ for every word sampled, resulting in a total complexity of $\mathcal{O}\left(|\mathcal{V}| \cdot \log|\mathcal{V}| \cdot T\right)$

Top-k sampling is used to sample from a truncated set of only the k most probable words at each time step, with a similar computational complexity as nucleus sampling. The truncation set is defined as the top k highest-probability tokens in the distribution, or the solution $\mathcal{V}^{(k)} \subseteq \mathcal{V}$ to the maximization problem in Equation (2.12). E.g., top-k sampling with $k = 3$ applied to the example distribution in Figure 2.10 would result in the truncation set $\mathcal{V}^{(k)} = \{\text{good}, \text{bad}, \text{long}\}$.

$$\begin{aligned} \max_{\mathcal{V}^{(k)}} \quad & \sum_{x \in \mathcal{V}^{(k)}} P\left(x|X\right) \\ \text{s.t.} \quad & \left|\mathcal{V}^{(k)}\right| \leq k, \end{aligned} \tag{2.12}$$

By removing the tail of the probability distribution, top-k sampling can improve the quality of the generated text and make it less likely to go off-topic. However, the optimal value of k can vary between different time steps, and selecting an appropriate value of k can be challenging. This is because the distribution of words can change at each time step, which means that a value of k that works well in one step may not work as well in another [67].

Temperature plays a crucial role in sampling-based generation from LMs and provides a flexible mechanism to control the balance between exploration and exploitation in ATG [72]. To apply temperature, the logits are divided by a chosen

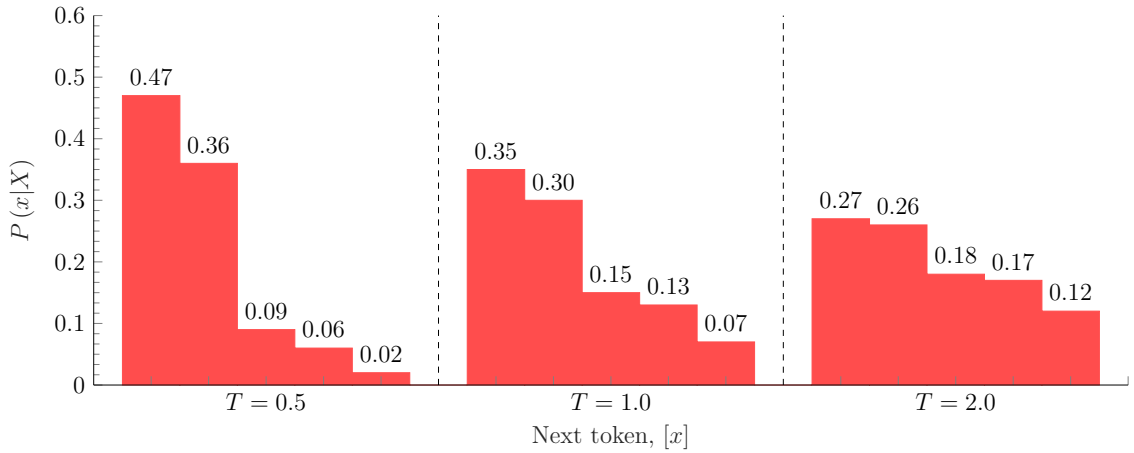


Figure 2.12: Illustration of how different values of temperature, T, effects the probability distribution from the example in Figure 2.10

temperature value, denoted as T, *before* either sampling directly or further truncating the vocabulary, e.g., by nucleus- or top-k sampling [73]. This rescaling of logits helps adjust the distribution of probabilities generated by the softmax function and its effect is illustrated in Figure 2.12.

When T is set between 0 and 1, the distribution becomes skewed towards high-probability tokens, effectively reducing the mass in the tail of the distribution. This adjustment biases the model towards more confident predictions, resulting in a narrower range of sampled tokens. However, it is worth noting that analyses have highlighted a trade-off between generation quality and diversity when lowering the temperature [74], [75]. Conversely, higher temperature values, greater than 1, introduce more randomness into the sampling process. This increased randomness leads to a broader exploration of the probability distribution and encourages the model to consider a wider range of potential tokens. In extreme cases, when the temperature approaches infinity, the sampling becomes uniform, meaning that all tokens have an equal chance of being selected [67].

2.6.2 Domain-Relevant Decoding Algorithms

The decoding algorithms described above are not suitable for our project because they do not take into account the specific language learning context and may generate text that is not appropriate for students as it can be both uninformative and repetitive. Furthermore, they do not offer a way to guarantee that the prompt word occurs in the generated sentences, which is a key requirement for our model. This section goes more into depth on the specific methods we used to mitigate these drawbacks of the standard decoding algorithms.

2.6.2.1 Keyword2Text Decoding

Pascual et al. [76] present an approach to controlled language generation using large pre-trained language models (like GPT–2). They propose a decoding method called

Keyword2Text (K2T), that, similarly to temperature, involves adding a shift of the probability distribution over the vocabulary towards semantically similar words based on a given topic or keyword. The authors use cosine similarity of their respective GloVe embedding [42] to measure the semantic similarity between words. Using the following definition of the score function: $\text{score}(\cdot|X) = \log P(\cdot|X)$, the suggested method produces the following shift of the probability distribution to guide generation toward the semantic space of the given prompt word, w:

$$\begin{aligned}\text{score}'(x, w|\, X) = \text{score}(x|X) \\ + \lambda \cdot \max\{0,\, \cos(\gamma(x), \gamma(w))\},\end{aligned} \tag{2.13}$$

where $\gamma(\cdot)$ is the GloVe embedding of a word and λ is the strength of the probability modification. Only words with positive similarity to the prompt word are "boosted" so as to not negatively effect words that would otherwise be favourable according to the original score function.

To ensure the eventual generation of the prompt word, the authors propose an exponential growth of the λ-parameter throughout the generated sentence:

$$\lambda_t = \begin{cases} \lambda_0 \exp\left\{\frac{c \cdot t}{T}\right\} & \textbf{if } t < T \\ \infty & \textbf{otherwise}, \end{cases} \tag{2.14}$$

at step t. T is the maximum length of the generated string, while λ_0 and c are hyperparameters that control the initial value and growth of λ, respectively. This boosting of the probabilities for certain words only grows until the keyword has been generated, at which point λ is set to 0.

The authors demonstrate that this simple method can be used to impose hard constraints on language generation, and show that it performs well in practice, leading to diverse and fluent sentences while ensuring the appearance of given guide words.

2.6.2.2 Typical Sampling

Meister et al. (2022) developed a method to generate text with higher "informativeness" [77]. In the article, the authors propose a new approach to generating text using probabilistic language models. They argue that current models often underperform when generating text, and suggest that this may be due to a lack of consideration for the ways in which humans use language as a communication channel. The authors propose a method they call *typical sampling*, which involves sampling words from the set of words with information content close to the conditional entropy of the model, rather than always choosing words from the high–probability region of the distribution. Similar to nucleus- and top-k sampling, this is the solution to a minimization problem where the truncation set, $\mathcal{V}^{(\tau)} \subseteq \mathcal{V}$, optimizes the following:

$$\begin{aligned} \min_{\mathcal{V}^{(\tau)}} \quad & \sum_{x \in \mathcal{V}^{(\tau)}} |H(x|X) + \log P(x|X)| \\ \text{s.t.} \quad & \sum_{x \in \mathcal{V}^{(\tau)}} P(x|X) \geq \tau, \end{aligned} \tag{2.15}$$

where $H(\cdot)$ is the Shannon entropy[6], or the expected information content, of a random variable with support χ [78] and τ is a hyperparameter determining what probability mass to include in the truncation. This is done with a computational complexity of $\mathcal{O}(|\mathcal{V}| \cdot \log|\mathcal{V}| \cdot T)$, equivalent to both nucleus- and top-k sampling.

The authors demonstrate that this approach offers competitive performance in terms of quality while consistently reducing the number of repetitions, and suggest that it could be a promising approach for improving the performance of probabilistic language models in text generation tasks [77].

2.7 Evaluation Methods

In this section, the assessment techniques used in the project are presented.

2.7.1 Cosine Similarity

Cosine similarity is a useful measure for calculating the similarity between two non-zero vectors in an inner product space. It is calculated as the cosine of the angle between the two vectors, and the resulting score ranges from -1 to 1. If the score is 1, the vectors have the same orientation, while a score of 0 indicates that they are orthogonal. The magnitude of the vectors does not affect the cosine similarity score. This measure has various applications in natural language processing, including determining the similarity between two strings or measuring how similar two documents are based on the number of occurrences of each word in the document. A significant advantage of cosine similarity is its computational efficiency, particularly for sparse vectors, as only non-zero coordinates need to be considered. It is defined as:

$$\text{cosine similarity}(\mathbf{u}, \mathbf{v}) = \cos(\theta) = \frac{\mathbf{u} \cdot \mathbf{v}}{||u||\,||v||}, \tag{2.16}$$

where θ is the angle between vectors $\mathbf{u}$ and $\mathbf{v}$.

2.7.1.1 Cosine Similarity for SBERT Sentence Embeddings

It's common to combine the cosine similarity of vectors and the sentence embeddings from the BERT encoder to measure how similar the generated sentences are to the fine-tuning dataset and it is illustrated step-by-step in Figure 2.13. The advantage of using cosine similarity with BERT embeddings is that it provides a simple and effective way to measure the similarity between two sentences in the high-dimensional embedding space. Furthermore, the use of BERT sentence embeddings, introduced in Section 2.4, allows us to capture the contextual information of the sentences in a fixed-size embedding, which is particularly relevant for tasks that require an understanding of the meaning of the sentences.

[6] $H(x) = -\sum_{x \in \chi} p(x) \log p(x)$.

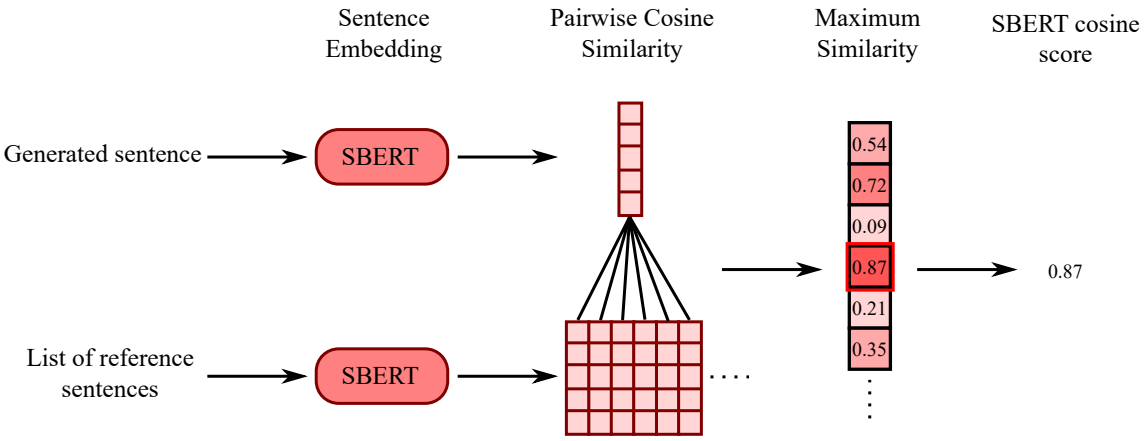


Figure 2.13: Illustration of the pipeline for the text comparison metric.

2.7.2 Perplexity

Perplexity is an evaluation metric for generative language models, calculated as a deterministic transformation of log-likelihood into an information-theoretic quantity. It is defined as:

$$\mathrm{PPL}\left(X\right) = 2^{-\frac{l(X)}{M}}, \tag{2.17}$$

where M is the total number of tokens in the held-out corpus and l is log-likelihood of word sequence $X = x_1, x_2, \ldots, x_m$,

$$l\left(X\right) = \sum_{t=1}^{M} \log P\left(x_t | X_{<t}\right). \tag{2.18}$$

Lower perplexities correspond to higher likelihoods, indicating better performance on this metric. Perfect language models achieve perplexity scores of 1, while the worst reasonable case scenario is a uniform, unigram model with a vocabulary size of V, which assigns equal probability $\frac{1}{V}$ to all words [79].

In practice, language models tend to produce perplexities in the range of 1 to V, with state-of-the-art models achieving increasingly lower perplexities on larger datasets [79]. For example, on the Penn Treebank dataset [80], a well-smoothed 5-gram model achieves a perplexity of 141 [81], while an LSTM language model can achieve a perplexity of approximately 80 [82], and enhancements to the LSTM architecture can bring the perplexity below 60 [83]. On larger datasets like the 1B Word Benchmark [84], perplexities of around 25 can be obtained by averaging together multiple LSTM language models [85].

2.7.3 MAUVE

In recent years, the development of open-ended text generation has led to the need for reliable evaluation metrics that can measure the similarity between machine-generated text and human-written language [86]. A proposed solution to this problem is MAUVE [87], a measure of comparison that evaluates the statistical gap between two text distributions: one from a text generation model and the other

from human-written text. MAUVE is a library built on PyTorch and implemented by HuggingFace Transformers.

The measure employs divergence frontiers and scales up to modern text generation models by computing information divergences in a quantized embedding space. The effectiveness of MAUVE is achieved by reducing the measurement to computing Kullback-Leibler divergences [88] in a quantized, low-dimensional space after embedding samples from each distribution with an external language model. The MAUVE measure consist of the area under the divergence curve, providing a summary of two type of errors: Type I is when the model put high mass of the probability distribution in an area which is unlikely to occur human natural language; Type II, instead, is when the model cannot generate text which is plausible in human content. MAUVE produces a number in the interval (0,1], where 1 indicates that machine-generated and human text are the same [87].

The implementation of MAUVE is user-friendly as it yields a scalar measure of the gap between neural text and human text. The measure can quantify differences in the quality of generated text based on the size of the model, the decoding algorithm, and the length of the generated text. In an extensive empirical study on three open-ended generation tasks, MAUVE was found to identify known properties of generated text, scale naturally with model size, and correlate with human judgements. Moreover, it was observed that MAUVE has fewer restrictions than existing distributional evaluation metrics [87].

3

Methods

In this chapter, an outline of the methods and setup used to achieve the objectives of the thesis will be presented. The dataset selection and processing procedures are described in Section 3.1. Section 3.2 outlines the model that was utilized along with its implementation process. Additionally, Section 3.3 provides a comprehensive explanation of the evaluation approach that was adopted for this research.

3.1 Dataset

This section will provide information about the data used in fine-tuning our model and the pre-processing pipeline. For further discussion pertaining to the dataset, see Section 5.1.1

3.1.1 Dataset Description

Tatoeba.org is a website that provides a large collection of example sentences in various languages [89]. The website allows users to download the data in the form of a TSV-file[1], which can then be used for building a task-specific dataset for fine-tuning a language model. Downloading the language data from there is relatively simple, it starts by accessing the Tatoeba website, then selecting the language and format you want the data to be in.

We chose Tatoeba as it is a good resource for building a task-specific dataset for fine-tuning a language model for four key reasons. Firstly, it provides a vast collection of sentences, which allows for building a large dataset for domain-adaptation. Secondly, the website's sentences are contributed by users, which provides a diverse set of sentences which lowers the risk of ideological biases within the dataset. Furthermore, the data provided by Tatoeba is free and open-source, which allows for easy access and usage. Lastly, the platform provides sentences in multiple languages, which makes it easier to extend the work done in this project to other language domains.

3.1.2 Pre-Processing of Data

The pipeline of the dataset pre-processing is illustrated in Figure 3.1.

[1]The dataset is available at: `https://downloads.tatoeba.org/exports/per_language/eng/`.

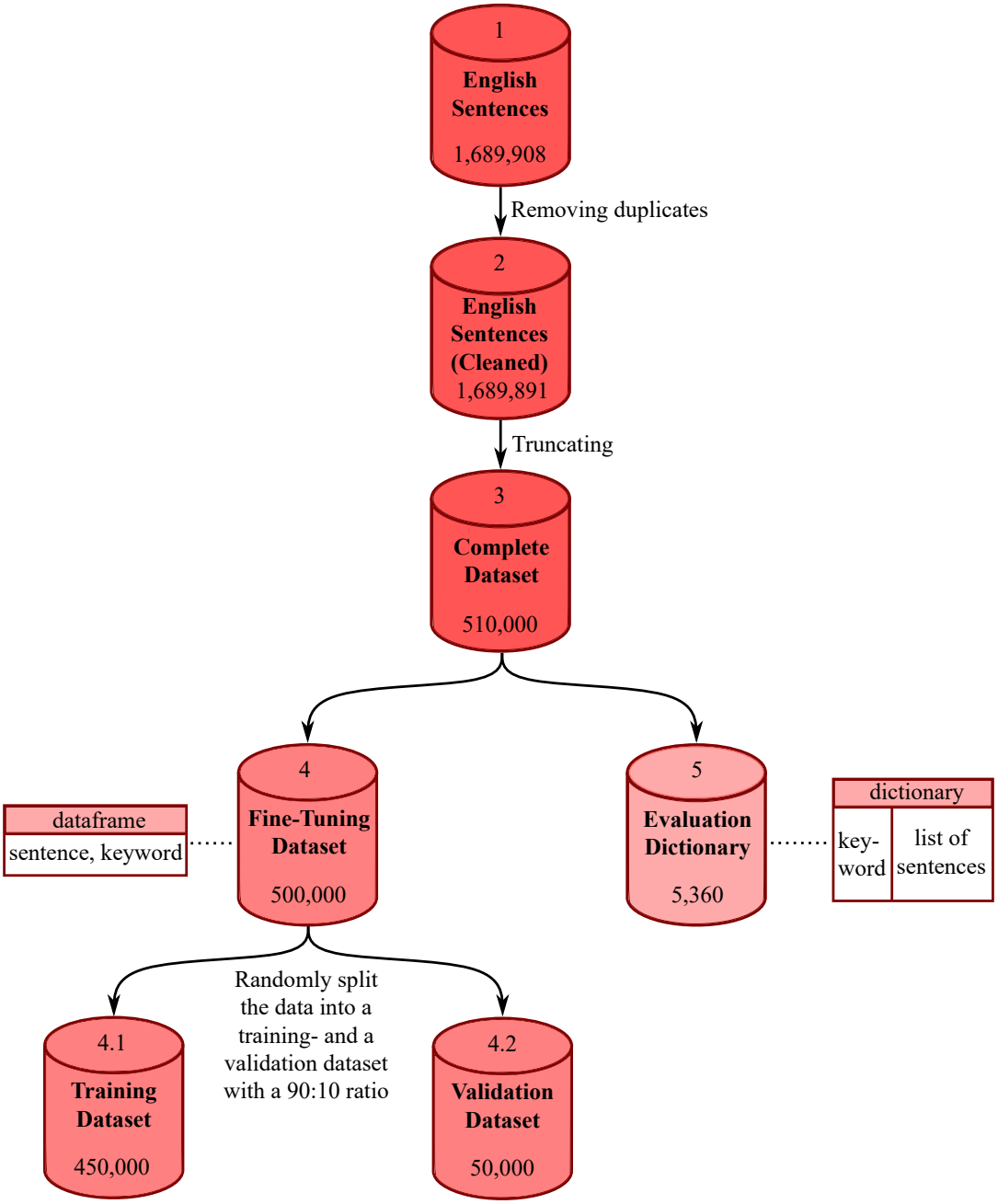


Figure 3.1: Schema of the pre-processing pipeline for our dataset.

The initial pre-processing included loading the aforementioned TSV-file into a Pandas DataFrame and removing empty strings and duplicate sentences. This process left 1,689,891 sentences after removing just 17 duplicates, and is represented by bin number 2 in Figure 3.1.

The vast amount of available sentences proved to make training impossible with the computing resources we had at our disposal. To circumvent this limitation and to avoid "`CUDA Out Of Memory`" errors we opted to truncate the dataset to only the first 510,000 sentences; 500,000 of which made up the *target dataset* for the fine-tuning of our model, found in bin number 4 of Figure 3.1.

From each sentence in the fine-tuning dataset, one word was selected as the designated keyword using `KeyBERT` [55] and Table 3.1 shows 10 examples of keyword-sentence pairs extracted by this method. 90% of these sentences were then randomly selected for the training dataset and the rest were used in the validation dataset, represented by bins 4.1 and 4.2, respectively. A more in depth description of how the fine-tuning was carried out can be found in Section 3.2.1.

Table 3.1: Examples of keywords extracted from sentences by KeyBERT [55].

Sentence	Keyword
I have to go to sleep.	sleep
I was in the mountains.	mountains
Is it a recent picture?	picture
I'll do my best not to disturb your studying.	disturb
For some reason I feel more alive at night.	night
It depends on the context.	context
Are you freaking kidding me?!	kidding
I don't want to be lame; I want to be cool!!	cool
I don't intend to be selfish.	selfish
Innocence is a beautiful thing.	innocence

To evaluate how well the fine-tuning let our model adapt to the domain of educational sentences we set aside 10,000 sentences and set up a hash table represented by bin 5. One key-value pair consists of a word and a list of all sentences where the keyword occurs in these 10,000 sentences. In total the dictionary consists of 5,360 words and for further explanation of how it was used in evaluating the fine-tuning, see Section 3.3.1.

Figure 3.2 shows how the sentence length distribution of the dataset is affected by the truncation process in step 2 of Figure 3.1. The blue bars represent the distribution of bin number 2 in Figure 3.1, while the red bars represent bin number 3. Initially, the average sentence length was $\mu = 7.50$ words per sentence with a standard deviation of $\sigma = 3.83$. After truncating the dataset, both the average sentence length and standard deviation decreases to 7.28 and 3.63, respectively. This is summarized in Table 3.2. Furthermore, ten extremely long[2] sentences were removed before fine-tuning, which can be found in Appendix A.1.

[2]In this case, "*extremely long*" means > 15 standard deviations away from the mean.

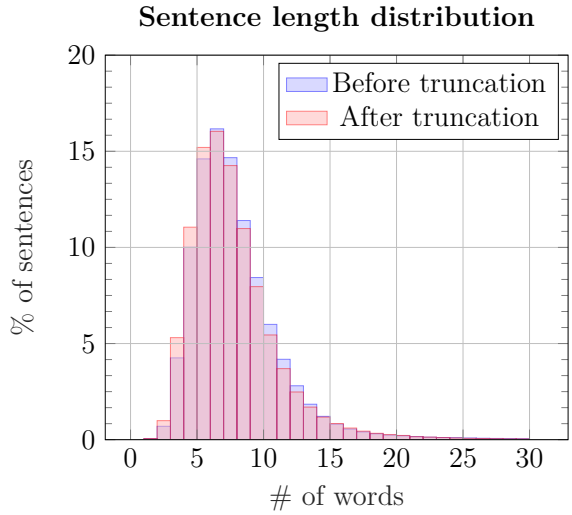


Figure 3.2: Histogram showing distribution of sentence length (in terms of number of words) of dataset before and after truncation.

Table 3.2: Average number of words per sentence (μ) and standard deviation (σ) of the dataset before and after truncation.

	μ	σ
Before truncation	7.50	3.83
After truncation	7.28	3.63

3.2 Model Implementation

For the implementation of the model in this research, the HuggingFace library's GPT–2 model was selected, specifically the `GPT2LMHeadModel`[3] variant, which is pretrained and designed for language modeling. The library also includes other implementations of GPT–2, such as `GPT2DoubleHeadsModel` and `GPT2ForTokenClassification`, which serve different purposes.

3.2.1 Fine-Tuning

To fine-tune the pretrained `GPT2LMHeadModel` on additional data, the Trainer class from the HuggingFace library was utilized. This class enabled the smooth fine-tuning of the model and was selected as the HuggingFace library is a SOTA library for NLP model implementations. The previously extracted keyword was given to the model as a prompt together with the sentence it was extracted from: "BOS w SEP <*sentence*> EOS", where BOS, SEP and EOS are tokens that indicate respectively, Beginning-Of-Sentence, SEParation and End-Of-Sentence, and then the training was carried out via teacher forcing, as described in Section 2.5. The model was fine-tuned on an NVIDIA Tesla V100-PCIE GPU (16GB) for three epochs.

[3] `https://huggingface.co/docs/transformers/v4.30.0/en/model_doc/gpt2#transformers.GPT2LMHeadModel`

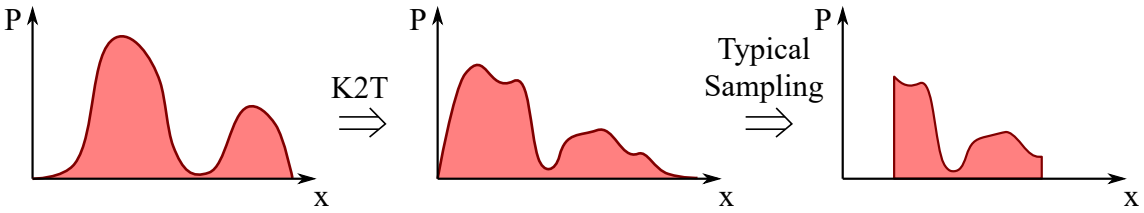


Figure 3.3: Illustration of the step-by-step effect our proposed decoding algorithm has on the probability distribution before sampling from the language model, where the x-axis are tokens in the vocabulary and the y-axis is the sampling probability distribution. The first step is a shift to the original distribution by the K2T method, and the second step represents the truncation of the vocabulary by the locally typical sampling.

The default settings in the HuggingFace trainer was used, but due to hardware related limitations, batch size during training was reduced to 10 (still using an evaluation batch size of 32). Parameters used during fine-tuning are presented in Table 3.3.

Table 3.3: Fine-tuning parameters for model.

Training batch size	10
Number of epochs	3
Starting learning rate	5e−5
Number of warmup steps	200
Weight decay	0.01

3.2.2 Decoding Algorithms

As described in Section 2.6, there are several shortcomings of the commonly used decoding algorithms. To mitigate these drawbacks a two-step approach was implemented to (1) guarantee that the keyword was generated every time and (2) increase the informational content of the generated sentences to better exemplify the keyword. An illustration of the combined effect of this method can be found in Figure 3.3.

In a similar manner to temperature being applied *before* the random sampling [67], the first step of the decoding process is a shift to the probability distribution output from the language model by the K2T algorithm described in Section 2.6.2.1. Implementation of this algorithm is very user-friendly and compatible with HuggingFace's library and PyTorch[4] since all source code is available with instructions on the author's GitHub repository[5]. Table 3.4 shows the hyper parameters values used from Equation (2.14), where T refers to the maximum sentence length, not temperature. These parameters were chosen after running some example with the evaluation dictionary to see what values are optimal for this task.

[4] `https://pytorch.org/`
[5] `https://github.com/dapascual/K2T`

After the word scores have been shifted in the first step, the vocabulary, from which sampling is performed, is truncated to mimic the informational density of the dataset, which the model learned during fine-tuning. This is illustrated as the second step in Figure 3.3 and is equally as intuitively implemented as the K2T probability shift. A `TypicalLogitsWarper`[6] implemented in the Transformer library directly by HuggingFace. This object accepts the probability mass hyperparamter, τ in Equation (2.15), as argument and works by setting the word scores of tokens excluded by the truncation to $-infinity$, effectively reducing their probability to be sampled to 0. When generating sentences, a parameter value of $\tau = 0.5$ was used, which is also included in Table 3.4.

Table 3.4: Hyperparameters used for K2T probability boosting of keyword and semantically similar words as well as locally typical sampling.

λ_0	0.8
c	0.25
T	30
τ	0.5

3.3 Evaluation

This section presents the pipeline used for evaluating the performance of the model throughout the distinct stages of the project.

3.3.1 Evaluating Fine-Tuning

As mentioned in Section 3.2.1, fine-tuning was carried out for 3 epochs, but to avoid overfitting, the model was evaluated every 1,500 steps on the validation dataset of bin 4.2 in Figure 3.1. A validation batch size of 32 was used and an early stopping was implemented where training would stop if validation loss did not improve for 3 evaluations on the validation set. At the end of training, the best performing model on the validation dataset was saved and used during the rest of the project. Parameters used to evaluate the fine-tuning are presented in Table 3.5 and loss data can be found in Section 4.1.

Table 3.5: Parameters used for evaluation during fine-tuning of model.

Evaluation batch size	32
Evaluation steps	1,500
Early stopping patience	3

[6] `https://huggingface.co/docs/transformers/internal/generation_utils#transformers.TypicalLogitsWarper`

3.3.2 Evaluating Generated Text

To evaluate the sentences generated by our model, a list of 500 keywords was constructed by using the top 1000 most frequently used English words, ranked based on the one billion word Corpus of Contemporary American English (COCA) [90][7]. From this ranking the words in places 501-1,000 were extracted, discarding the first 500 words reasoning that they were common enough to either be uninteresting; e.g., *a*, *the*, *of*, *and*; or common enough to be taught really early on in the language learning process and not really need exemplification; e.g., *house* or *they*.

For every word in this list three sentences were generated, giving a total of 1,500 sentences and on these sentences a suite of five metrics were calculated. Apart from cosine similarity, perplexity and MAUVE, which were introduced in Section 2.7, the proportion of sentences in which the keyword was generated and the proportion of sentences that were identical to an example in the dataset were calculated. The percentage of generated sentences containing the prompt word is important to measure since it is one of the key requirements of our model. Furthermore, it is also important to keep track of to what degree the model generates sentences that are already in the dataset because, if we generate sentences that are identical to some examples in the dataset, why not just use the existing sentences from Tatoeba to practice using the keyword?

The evaluation pipeline described above was computed three times during the project. First, it was evaluated on sentences generated after the fine-tuning to the target dataset and for generation a temperature of 0.5 was used with a hybrid top-k- and nucleus sampling scheme with $k = 75$ and $p = 0.75$. Second, to evaluate the effect of adding the K2T probability shift, the temperature was set to 1 and K2T was implemented with the hyperparameters found in Table 3.4. This intermediary model used the same hybrid sampling scheme as the previous one described above. Lastly, the full implementation of `SpeakEasy` was evaluated with the hyperparameter set found in Table 3.4.

There is a significant possibility that automatically applied metrics are not able to capture and reflect the specific aspects of what makes a sentence a good example to practice on for a language learner. To evaluate the evolution of the "usefulness" of the generated sentences, human evaluation was used.

3.3.2.1 Human Evaluation

One big factor to take in to account when evaluating the performance of a model in the specific domain of language learning is how useful the generated sentences are when exemplifying the proper use of a word for someone learning a new language. This is a rather intangible property of a text and something which is not easily reflected via an automatically calculated metric.

To perform this evaluation, a questionnaire was used containing 100 questions which prompted respondents to rank four sentences containing the same keyword on how

[7]Available at: `https://www.wordfrequency.info/samples.asp`

useful they were to language learners. Examples of questions are presented in Figure 3.4. The keywords were the first 100 words in the list used for the automatic evaluation described above, the 501st to the 600th most frequently used English words. One of the four sentences was sampled randomly from the human written sentences in the Tatoeba dataset containing the keyword, while the remaining three were generated one from each model evaluated in the automatic evaluation, respectively. One thing to keep in mind is that the evaluators are non-professional linguists but rather friends and students, so the results will not be viewed from a pedagogical perspective.

The four example sentences were presented in a random order, and respondents were instructed to rank them from 1 to 4, where 4 represented the worst and 1 the best sentence. Furthermore, the instructions for the survey included the following list of criteria to have in mind when ranking the examples, in a *descending* order of importance:

1. **Grammatical correctness and sense-making:** The most crucial aspect is that the sentence is grammatically correct and makes sense. If a sentence fails in this regard, please rank it last.
2. **Exact inclusion of the keyword:** Check if the keyword is included in the sentence exactly as it is given in the question. If it is not, it should be ranked lower.
3. **Exemplification of the prompt word:** If both the grammar and keyword criteria are fulfilled, distinguish between sentences based on how well they exemplify the given prompt word. Consider whether the sentence effectively conveys the meaning of the prompt word.
4. **Usefulness to language learners:** Finally, evaluate how useful each sentence would be for a language learner to train on. Consider whether the sentence provides meaningful context and aids in language comprehension and learning.

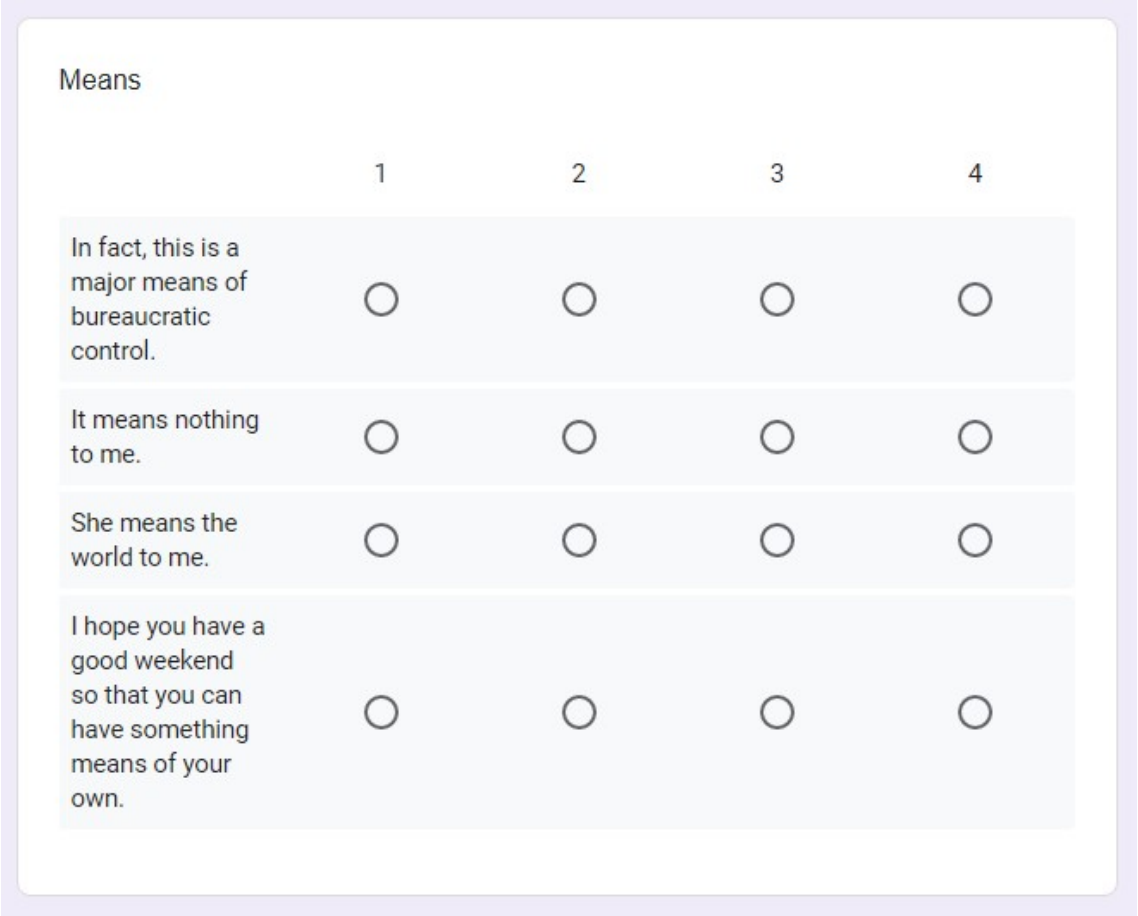


Figure 3.4: Example of a question in the human evaluation questionnaire.

4

Results and Analysis

In this chapter, the results of the experiments conducted during this project are presented and analyzed. First, Section 4.1 presents the results of the fine-tuning of the GPT–2 model. Then, examples of sentences generated by `SpeakEasy` are presented in Section 4.2. Sections 4.3 provides a presentation and analysis of the results from the automatic- and human evaluation.

4.1 Fine-Tuning

Fine-tuning of the model lasted during approximately eight hours and all three epochs elapsed without early stopping. Average (over batch) loss values on both the training- and validation data are presented in Figure 4.1, where the loss function is the cross entropy of Equation (2.10). After an initial transient during the beginning of each epoch the model quickly converges, and small improvements are made between epochs.

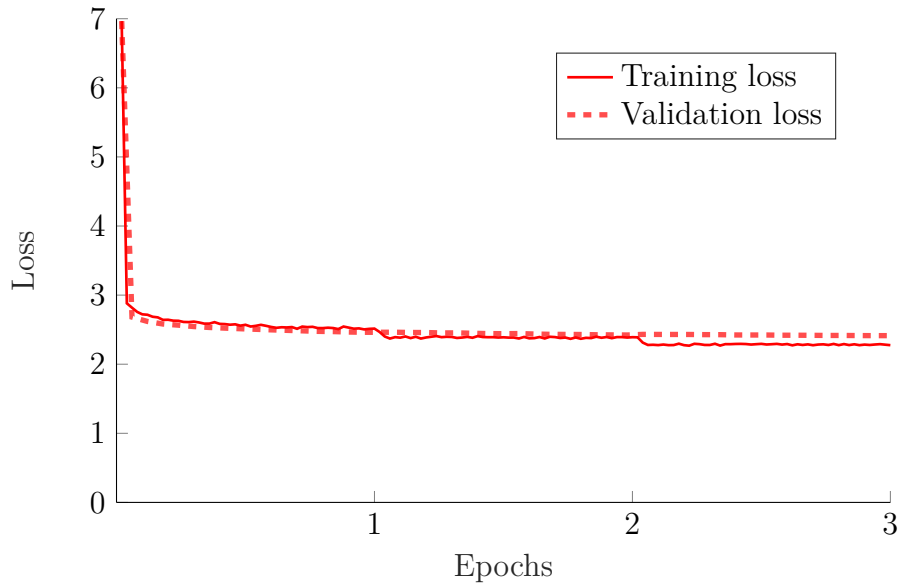


Figure 4.1: Plot of training- and validation loss during fine-tuning of the GPT–2 model for three epochs.

4.2 Example SpeakEasy Outputs

The table 4.1 presents some example of sentences generated by SpeakEasy given the keyword as input to the model. Generating an example sentence from a given keyword with SpeakEasy takes on average 0.45 ± 0.19 seconds, measured over 1000 sentence generations. This time includes tokenizing the prompt word, generating the sentence token-by-token and decoding the generated sentence from the word tokens. Although we have not found any studies, nor performed one of our own, investigating the average time for a human to write example sentences, we are convinced a human would not be faster than SpeakEasy. This is especially true when generating a large amount of sentences in short succession, since a human most probably would experience fatigue and slow down with time, but a computer would not.

Table 4.1: Examples of sentences generated by SpeakEasy.

Keyword	Sentence generated
ways	There are many ways to express this.
voice	His voice began to be heard over the whole room.
ready	We'll be ready in five minutes.
strong	Tom is a strong person.
society	As a society, we can provide food and clothes to the poor.
single	She has a single mother.
results	We could have found better results.
student	She is a student at this school.
hair	He shaved his hair.
medical	A medical emergency is an urgent and necessary one.

4.3 Evaluation Results

This section presents the results of the human and automatic evaluation. Since text generation by sampling is a stochastic process, all numerical results are presented as the mean score over a sample and a confidence interval of two standard deviations, corresponding to a confidence level (Cl) of 95% [91]. The standard deviation, σ, of a random sample of size N is calculated as

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}, \tag{4.1}$$

where μ is the sample mean.

4.3.1 Automatic Evaluation

The results of calculating the MAUVE scores, perplexities and cosine similarities of the three models, as described in Section 3.3.2, are presented in Table 4.2, while Table 4.3 shows how frequent each model samples from the dataset and successfully generates the keyword. Values in bold text are the best results for that given metric

Table 4.2: Automatic quality metrics for the different model stages throughout the project, as described in Sections 2.7 and 3.3.2.

	MAUVE (↑)	PPL	Cosine sim. (↑)
Tatoeba	—	215.5 ± 496.6	—
Only fine-tuning	0.621 ± 0.618	170.3 ± 385.1	0.572 ± 0.124
Fine-tuning + K2T	0.745 ± 0.678	188.8 ± 383.1	0.575 ± 0.116
SpeakEasy	$\mathbf{0.755 \pm 0.592}$	$\mathbf{234.8 \pm 510.9}$	$\mathbf{0.577 \pm 0.118}$

Table 4.3: Percentages of sampled sentences and successfully generated keywords from the examples generated by the autoregressive Transformer language models.

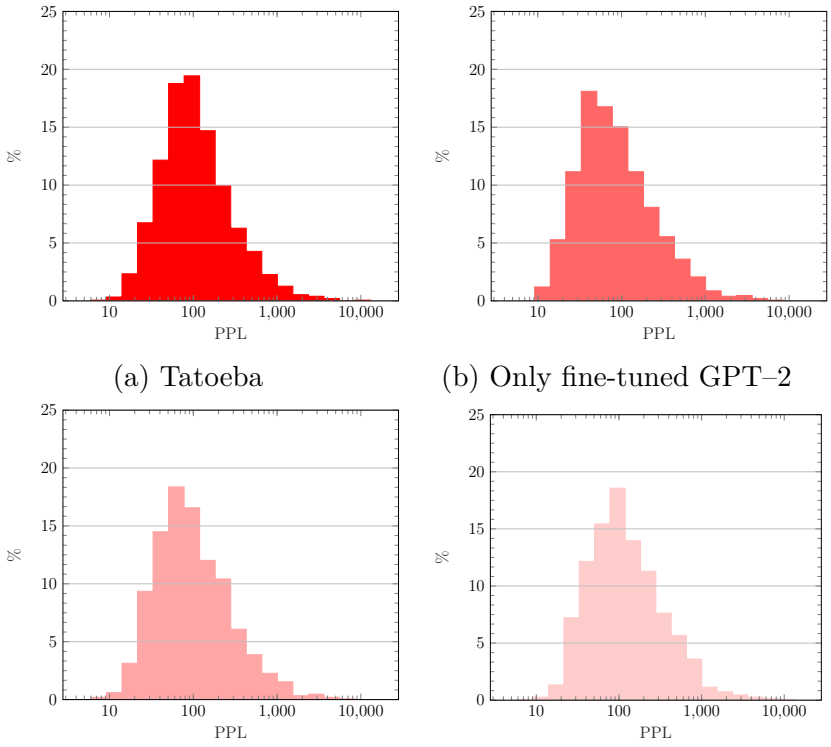
	% sampled (↓)	% keyword (↑)
Only fine-tuning	9.66%	84%
Fine-tuning + K2T	3.83%	**100**%
SpeakEasy	**1.83**%	**100**%

among the models and the direction of the arrow indicates if a high or low value is preferred. For perplexity (PPL), the preferred value is the one closest to that of the reference example sentences from Tatoeba. Furthermore, perplexities are evaluated under an independent, i.e., not fine-tuned, LM; In this case, DistilGPT–2 [92] was used. For MAUVE and cosine similarity there are no reference values, since they are calculated as a relationship between the generated sentences and the Tatoeba sentences.

The results presented in Tables 4.2 and 4.3 show that SpeakEasy is the best performing model for all metrics evaluated, although the differences are small and within the confidence intervals for both the MAUVE score and the cosine similarity. When measuring the percentage of sentences where the keyword was successfully generated in its exact form, both SpeakEasy and Fine-tuning + K2T scored 100%. This is to be expected since the probability shift applied by K2T increases exponentially until the keyword is generated, essentially forcing the model to include the keyword in every generated example sentence. Furthermore, SpeakEasy also generated the least amount of sentences that were already in the dataset.

Upon further inspection of the data in Table 4.2, the perplexity scores stand out in that they have a standard deviation significantly larger than their mean value, for every model and the dataset alike. Considering that perplexity values are non-negative, this hints at a small number of outliers with perplexities much greater than the mean. Figure 4.2 shows the distribution of the perplexity values for all models examined and the reference dataset and an inspection of these histograms indicates that this is the case. The minimum, median and maximum values for perplexities reported in Table 4.4 further supports this conclusion.

Overall, the automatically applied metrics indicate a slight increase in performance for every addition made to the model after fine-tuning. However, the differences are small and with the given confidence intervals, no definite conclusions can be



(a) Tatoeba (b) Only fine-tuned GPT–2

(c) Fine-tuning + K2T (d) SpeakEasy

Figure 4.2: Distribution of perplexity scores for all four example sentence sources, where the count in each bin is normalized to a percentage of the total number of sentences.

Table 4.4: Minimium, median and maximum perplexity scores for the Tatoeba dataset, fine-tuned GPT–2, K2T-controlled generation and SpeakEasy.

	Min	Median	Max
Tatoeba	9.159	94.508	12,500.550
Only fine-tuning	9.640	72.796	7,890.354
Fine-tuning + K2T	7.841	86.228	6,487.439
SpeakEasy	6.380	107.656	9,791.417

drawn from this data alone without analysing the results from the human evaluation down below. Further discussion of the reliability of these results can be found in Section 5.1.4.

4.3.2 Human Evaluation

All results in this section are based on the questionnaire introduced in Section 3.3.2.1, which got 8 respondents and a total of 792 data points.

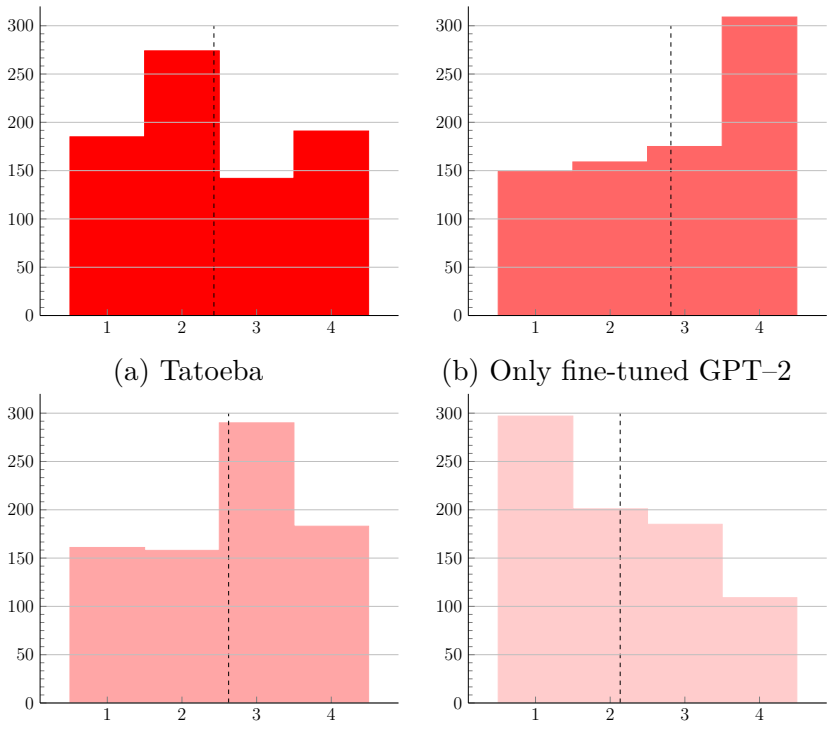
Figure 4.3 shows histograms over the amount of times each model was ranked at each position, where 1 indicates the best sentence and 4 represents the worst. This data shows that the model which was only fine-tuned tends to be ranked last of the four alternatives, while the fine-tuned model with controlled K2T generation tends to be ranked as the second to worst alternative. The sentences generated by our model, SpeakEasy, were most commonly ranked as the best, while the ranking of the example sentences from Tatoeba were more evenly distributed between the ranks, with a slight trend towards ranks 2 and 4. These results indicate that the model constructed in this project, SpeakEasy, performs better than all other example sentence sources used in the survey, even the sentences from the dataset itself.

In Table 4.5, mean ranks afforded to the three types of sentences are presented, along with 95% confidence intervals. However, the differences are small, especially considering the large confidence interval in all four cases, and further analysis is needed to support this claim.

Table 4.5: Mean rank afforded to each type of example sentence during human evaluation.

	Mean rank (↓)
Tatoeba	2.43 ± 2.18
Only fine-tuning	2.81 ± 2.29
Fine-tuning + K2T	2.63 ± 2.10
SpeakEasy	$\mathbf{2.13 \pm 2.13}$

When examining Figure 4.4, which shows the distribution between the sentence sources for each individual rank, it is evident that most 1's tend to be given out to sentences generated by SpeakEasy, most 2's are given to human-written sentences from Tatoea, most 3's are afforded to sentences generated by a fine-tuned model with K2T generation, and the sentences source which gets ranked 4^{th} most often is the GPT–2 model that was only fine-tuned without any additional domain-specific decoding algorithms. This is further supported by looking at the average scores per question each model got, represented as a box plot in Figure 4.5. Here one clearly sees that the ranking proposed above holds, not only for median values, but for minimum values, 25^{th} percentiles, 75^{th} percentiles and maximum values as well.



(a) Tatoeba (b) Only fine-tuned GPT–2

(c) Fine-tuning + K2T (d) SpeakEasy

Figure 4.3: Histograms for the individual sentence sources from the evaluation survey. Sample means are marked by dashed black lines.

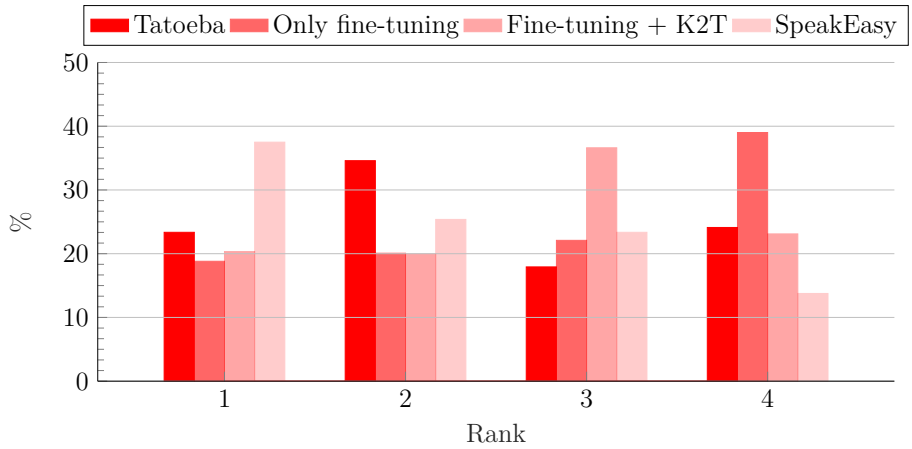


Figure 4.4: Bar graph indicating how the different ranks were distributed between the example sentence sources.

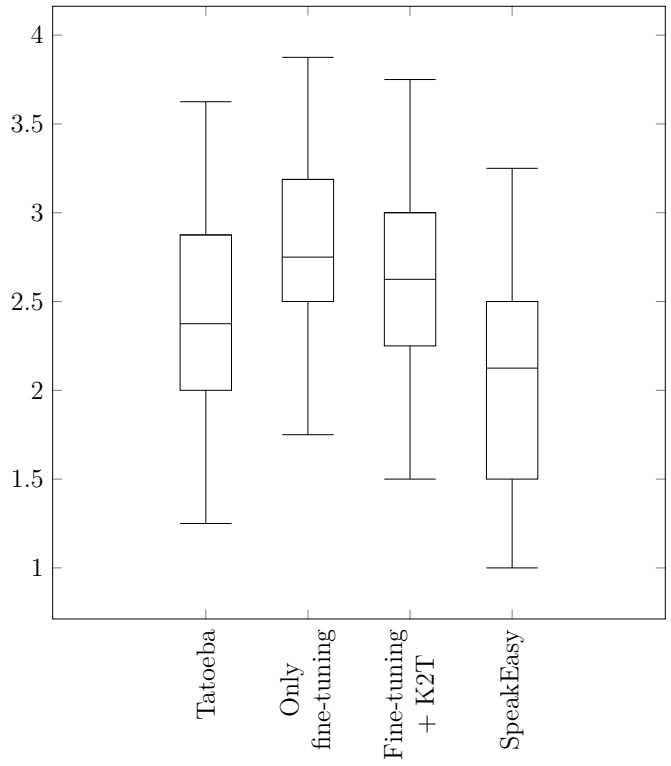


Figure 4.5: Box plot showing the spread of the average rank per question over all respondents for every example source.

4.4 Summary

To summarize the key findings, the SpeakEasy model performs well enough to generate sentences with a higher perceived quality than the human-written Tatoeba example sentences. Additionally, both automatic and human evaluation data suggests that both the domain-specific decoding methods, K2T and typical sampling, provide an improvement of the quality of the generated sentences. All sentences generated by SpeakEasy contained the keyword in the exact form it is given in the prompt and only 1.83% of sentences were identical to an entry in the datset. Furthermore, sentences were generated with what we confidently claim to be super-human speed, but no formal study comparing the writing speed of humans and SpeakEasy has been carried out. A discussion regarding potential error factor of these results is provided in Section 5.1.4.

5

Discussion

This chapter presents a discussion regarding several key aspects of our project. Section 5.1 provides a reflection regarding potential issues with the methods used and the results obtained, while ethical considerations are handled in Section 5.2. Finally, factors limiting the project are discussed in Section 5.3.

5.1 Potential Errors in Methods and Results

In this section, we will examine various factors that may have influenced the proposed model and the outcomes of the project.

5.1.1 Dataset

It is important to address potential shortcomings and sources of errors within the dataset used for fine-tuning and evaluation. While Tatoeba is widely used and provides a valuable resource for example sentences, certain considerations need to be taken into account.

One possible concern is the reliability and consistency of the dataset itself. It is possible that the dataset may contain sentences in languages other than English, due to user errors when submitting sentences to the website. This introduces the risk of encountering non-English sentences within the dataset, which does not align with the intended focus of our project. However, we never encountered any instances of this when reading sentences from the dataset, the vast amount of data makes it impossible to manually check every sentence for a correct language labelling. A potential way to counteract these kinds of error could have been to put every sentence in the dataset through a language detection model[1] and removing non-English entries.

Additionally, Tatoeba gathers sentences from various contributors, and the dataset includes sentences from multiple sources. It is crucial to acknowledge that the credibility and quality of these sources can vary. Some sentences may be uploaded by less reliable or biased users, which introduce unintended biases and errors into the dataset. In turn, this can affect the generated example sentences and potentially

[1]For example this model based on the RoBERTa model [93]: `https://huggingface.co/papluca/xlm-roberta-base-language-detection`.

result in erroneous sentences, like the ones removed and presented in Appendix A.1, or even outright offensive language. This particular problem is further discussed in Section 5.2. For this reason we cannot say that all the sentences are going to be pedagogically useful.

During pre-processing of the data, duplicate sentences were removed to avoid the artificial skewing of the training data toward the semantic space of these particular sentences. However, if there was not an exact match between the sentences, automatic removal might not have occurred. A more elaborate and exhaustive search for duplicates perhaps could have found additional pairs of equivalent sentences and removed these types of biases.

5.1.2 Model Implementation

Although efforts have been made to develop a functional and effective model, limitations in terms of time and resources have constrained the exploration of various parameters that could potentially enhance the model's performance.

One notable disadvantage is the limited exploration of different parameters during the transfer learning process. Due to time constraints, only a very restricted set of parameters was investigated, hindering our ability to fully assess their impact on the model's effectiveness. Parameters such as batch size, number of epochs, weight decay, and learning rate could have been further explored to determine their optimal values and ascertain their potential benefits to the model's performance.

For instance, altering the batch size could have yielded valuable insights. Increasing the batch size has the potential to expedite the training process, allowing for more iterations and providing additional time for refining the model between each training cycle. Exploring the effects of varying batch sizes could have shed light on the trade-off between computational efficiency and model performance.

Furthermore, the number of epochs, weight decay, and learning rate are crucial hyperparameters that could significantly impact the model's convergence and generalization capabilities. By thoroughly investigating the effects of different values for these parameters, we could have gained a deeper understanding of their influence on the model's performance and made informed decisions to optimize the learning process.

Considering the limitations encountered in parameter exploration, it is essential to acknowledge the potential for further improvements in the model's implementation. Allocating more resources and dedicating additional time to experiment with a broader range of parameters could enhance the model's performance and uncover potential optimizations.

Lastly, teacher forcing, introduced in Section 2.5, may not be the optimal method to employ when fine-tuning a language model for the specific language generation task at hand. This is primarily because there is no single "gold standard" sentence that can serve as a definitive reference for every keyword. Instead, there exists a vast range of equally valid and appropriate example sentences. Relying solely on teacher

forcing, which enforces a predetermined ground truth input sequence, may restrict the model's ability to explore the diverse and nuanced possibilities offered by the English language. Incorporating alternative approaches that allow for more flexible and diverse sentence generation, such as reinforcement learning or a combination of techniques, could potentially yield more desirable results in this scenario. Indeed, popular LMs, such as chatGPT, use reinforcement with human feedback to improve the relevance of the text [23].

5.1.3 Evaluation

Ensuring the reliability and accuracy of the evaluation results is of importance to obtain meaningful insights into the model's performance. However, it is important to acknowledge certain factors that may introduce variability and potential errors into the evaluation process.

Assessing the performance of a language model presents a significant challenge, necessitating the use of diverse performance metrics and evaluation approaches. Relying solely on automatically calculated metrics can be insufficient, which led us to adopt a strategy involving both automatic metrics and a human analysis of the generated sentences. To conduct this analysis, human evaluators were enlisted to rank example sentences from various sources in terms of their quality.

Human evaluation, although valuable, can be subject to certain limitations and sources of unreliability. One such challenge arises when respondents are faced with the task of determining the usefulness of the generated examples for language learners specifically, leading to potential ambiguities in their assessments. Additionally, the presence of words with multiple meanings, such as "*current*" and "*can*", may pose challenges for evaluators, as it becomes difficult to discern the intended meaning associated with a particular keyword. This inherent ambiguity can introduce inconsistencies in the evaluation process and impact the reliability of the results.

Furthermore, human errors can also contribute to the unreliability of the evaluation. Respondents may inadvertently misunderstand the evaluation instructions, leading to unintended biases or inaccuracies in their rankings. Additionally, the possibility of accidental "misclicks" or selection of incorrect inputs by respondents cannot be entirely ruled out, potentially affecting the accuracy of the evaluation results.

Several measures can be taken to address these concerns and improve the evaluation process. Clear and unambiguous instructions should be provided to the evaluators, ensuring a common understanding of the evaluation criteria and objectives. It may also be beneficial to provide additional guidance or examples to aid evaluators in making informed and consistent assessments, especially in cases where words with multiple meanings are involved. Moreover, incorporating domain experts, such as English teachers, instead of non-professionals as evaluators in the project would bring valuable expertise and insights to the evaluation process. Their specialized knowledge and experience in language learning and teaching would most likely provide more accurate and informed assessments of the generated example sentences.

5.1.4 Results

While the mean values for all the measurements indicate promising outcomes, it is important to acknowledge the impact of large standard deviations on the confidence intervals. The wide range of values within these intervals implies a significant level of variability and uncertainty in the results. This variability can be attributed to various factors, including a small number of very large outliers in the case of perplexities and the limited number of respondents in the survey. Increasing the number of participants in the survey, would have yielded a more robust and statistically significant result.

5.2 Ethical Considerations of the Project

Throughout this project, various ethical and sustainability considerations come into play, particularly regarding the automatic generation of natural language and its impact on society.

Firstly, there is the ecological sustainability aspect associated with training the LM. The training process consumes significant computational and electrical resources, which have environmental implications. However, it should be noted that in this study, fine-tuning an existing model was chosen over training a model from scratch, which helped save the resources that would have been utilized in the pre-training phase [94].

Secondly, the social sustainability of the model is of utmost importance. It is crucial to address the potential risks of generating offensive language or propagating discriminatory biases learned from the dataset. Special care must be taken to ensure that SpeakEasy adheres to ethical guidelines and avoids generating content that may be harmful, offensive, or discriminatory. These concerns weren't taken into consideration because we focused on building a model according to our aim, generating pedagogically useful sentences. Therefore, we did not aim to build a model that is ready to be publicly deployed. Measures such as pre-processing the dataset to filter out inappropriate or biased content should be implemented to mitigate these risks. Furthermore, monitoring and evaluating the model's output is necessary to identify and rectify any unintended offensive language that may emerge during the language generation process. For this purpose, an automatic flame detector for offensive or abusive language could be used, as proposed by Razavi et al. (2010) [95].

5.3 Limitations

This study faced certain limitations. The time constraints of the project imposed limitations on the amount of times we were able to fine-tune the language model and optimize the hyperparameters of the training process. These time constraints also inhibited an in-depth investigation of the potential issues discussed in Sections 5.1 and 5.2. Furthermore, computational limitations put an upper limit on the batch size during training, negatively impacting training time, and effectively further reducing

the ability to perform an extensive hyperparamater optimization.

6

Conclusion

6.1 Project summary

This project had the goal of developing a model which generates high-quality sentences exemplifying a single English word to aid language learners by providing them with sentences to practise on. The aim was to provide a model that takes a prompt consisting of the specific keyword that the user wants to learn, and generates a grammatically correct sentence which helps the user understand and learn the word's usage effectively; SpeakEasy does just that.

The results obtained align with our objective, where both automatic and human evaluation shows a visible improvement for our model in generating the sentences at every stage throughout the project. The perplexity shows that the sentences generated from the fully implemented SpeakEasy are closer to the human-written examples than those from a fine-tuned GPT–2 model, with or without controlled K2T [76] generation implemented. MAUVE scores and cosine similarity also indicate that SpeakEasy is closer to the human sentences than the other models evaluated. Furthermore, also the human evaluation supports the conclusion that SpeakEasy's sentences achieve a higher quality then the others, even achieveing a lower[1] average rank than the human-written control sentences from the dataset included in the evaluation survey.

However, we are aware of some potential limitations and issues, which are discussed further in Chapter 5. The key limitations were time-related, challenging our ability to adequately investigate the hyperparameters' role in the result of fine-tuning. The two main potential error sources were related to inadequacies in the dataset and the multi-faceted issue of evaluating automatically generated texts without a ground truth reference.

In conclusion, the results achieved in this project are sufficient to answer the research questions posed in Section 1.3. An NLP-based system can be trained to generate contextually appropriate English sentences for language learners, even achieving a perceived higher quality that the human-written examples from Tatoeba according to the human evaluators. This is achieved by using a decoding algorithm combining K2T controlled generation, to ensure the successful generation of the keyword, with

[1] A rank of 1 indicates the **best** sentence, while 4 represents the **worst** from the examples in the survey.

locally typical sampling [77], to increase the informational content of the sentences. Evaluation is performed using a combination of five key metrics described in Section 3.3.2 with a ranking-based human evaluation questionnaire where respondents rank four sentences at a time based on four criteria outlined in Section 4.3.2. This is, to the best of our knowledge, the first time a generative LM has been applied to the problem of generating suitable example sentences specifically in a language learning context, as previous methods have been retrieval or ranking based [2]–[4].

6.2 Future Work

There are a lot of potential future work for this project. Firstly, adapting the complexity of the generated sentences according to the user's language proficiency level could be an avenue for future research. A preliminary idea for achieving this could be to make use of the specific vocabularies associated with the different CEFR levels of language proficiency [96].

Secondly, another extension of this work could be to develop a similar model which accepts multiple words as input, like phrasal verbs as "*get up*" or "*get back*". In fact, phrasal verbs constitute a significant aspect of the English language, because they are numerous and sometimes they completely alter the meaning of the words, when taken in isolation. Related to this, there could be also the option to express the desired usage of a word, like "*fall, verb*" or "*fall, season*", to avoid confusing homographs and have the sentence exemplify the exact version of the word that the user wants to practice on using.

Furthermore, in future iterations, it is recommended to conduct comprehensive parameter sweeps, systematically varying the parameters of interest to identify the optimal configurations. This iterative approach would allow for a more thorough exploration of the parameter space and provide valuable insights into how different settings impact the model's behaviour and performance.

Bibliography

[1] Duolingo, Inc., *2022 Duolingo Language Report*, 2022. [Online]. Available: https://blog.duolingo.com/2022-duolingo-language-report/.

[2] I. Pilán, E. Volodina, and L. Borin, "Candidate sentence selection for language learning exercises: from a comprehensive framework to an empirical evaluation," *TAL*, vol. 3, pp. 1–25, 2016. [Online]. Available: https://arxiv.org/pdf/1706.03530.pdf.

[3] A. Kilgarriff, M. Husák, K. McAdam, M. Rundell, and P. Rychlý, "GDEX: Automatically finding good dictionary examples in a corpus," in *Proceedings of EURALEX*, 2008, pp. 425–432. [Online]. Available: https://www.euralex.org/elx_proceedings/Euralex2008/095_Euralex_2008_Adam%5C%20Kilgarriff_Milos%5C%20Husak_Katy%5C%20McAdam_Michael%5C%20Rundell_Pavel%5C%20Rychly_GDEX_Automatically%5C%20Finding%5C%20Good%5C%20Di.pdf.

[4] I. Pilán, E. Volodina, and R. Johansson, "Automatic Selection of Suitable Sentences for Language Learning Exercises," in *20 Years of EUROCALL: Learning from the Past, Looking to the Future*, Évora, Portugal, 2013, pp. 218–225, ISBN: 978-1-908416-13-1. [Online]. Available: http://research-publishing.net/publication/chapters/978-1-908416-13-1/Pilan_Volodina_et_al_164.pdf.

[5] E. Sumita, F. Sugaya, and S. Yamamoto, "Measuring Non-native Speakers' Proficiency of English by Using a Test with Automatically-Generated Fill-in-the-Blank Questions," in *Proceedings of the Second Workshop on Building Educational Applications Using NLP*, Ann Arbor, Michigan: Association for Computational Linguistics, Jun. 2005, pp. 61–68. [Online]. Available: https://aclanthology.org/W05-0210.

[6] M. Wojatzki, O. Melamud, and T. Zesch, "Bundled Gap Filling: A New Paradigm for Unambiguous Cloze Exercises," in *Proceedings of the 11th Workshop on Innovative Use of NLP for Building Educational Applications*, San Diego, CA: Association for Computational Linguistics, Jun. 2016, pp. 172–181. DOI: 10.18653/v1/W16-0519. [Online]. Available: https://aclanthology.org/W16-0519.

[7] E. Kumar, *Natural language processing*. IK International Pvt Ltd, 2013.

[8] J. Allen, *Natural language understanding*. Benjamin-Cummings Publishing Co., Inc., 1995.

[9] D. D. McDonald, "Natural language generation.," *Handbook of natural language processing*, vol. 2, pp. 121–144, 2010.

[10] T. Poibeau, *Machine translation*. MIT Press, 2017.
[11] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," *Information*, vol. 10, no. 4, p. 150, 2019.
[12] A. Andrenucci and E. Sneiders, "Automated question answering: Review of the main approaches," in *Third International Conference on Information Technology and Applications (ICITA'05)*, IEEE, vol. 1, 2005, pp. 514–519.
[13] T. Nasukawa and J. Yi, "Sentiment analysis: Capturing favorability using natural language processing," in *Proceedings of the 2nd international conference on Knowledge capture*, 2003, pp. 70–77.
[14] L. Deng and Y. Liu, *Deep learning in natural language processing*. Springer, 2018.
[15] H. Li, "Deep learning for natural language processing: advantages and challenges," *National Science Review*, vol. 5, no. 1, pp. 24–26, 2018.
[16] J. Mahapatra, S. K. Naskar, and S. Bandyopadhyay, "Statistical natural language generation from tabular non-textual data," in *Proceedings of the 9th International Natural Language Generation conference*, 2016, pp. 143–152.
[17] S. Pan and J. Shaw, "SEGUE: A hybrid case-based surface natural language generator," in *Natural Language Generation: Third International Conference, INLG 2004, Brockenhurst, UK, July 14-16, 2004. Proceedings*, Springer, 2004, pp. 130–140.
[18] E. Reiter, S. Sripada, J. Hunter, J. Yu, and I. Davy, "Choosing words in computer-generated weather forecasts," *Artificial Intelligence*, vol. 167, no. 1-2, pp. 137–169, 2005.
[19] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Cernock, "Empirical Evaluation and Combination of Advanced Language Modeling Techniques.," in *Interspeech*, 2011, pp. 605–608.
[20] T. Wolf, L. Debut, V. Sanh, *et al.*, "Transformers: State–of–the–art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.
[21] K. S. Kalyan, A. Rajasekharan, and S. Sangeetha, "AMMUS: A survey of transformer-based pretrained models in natural language processing," *arXiv preprint arXiv:2108.05542*, 2021.
[22] L. Ouyang, J. Wu, X. Jiang, *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 730–27 744, 2022.
[23] OpenAI, *GPT-4 Technical Report*, 2023. arXiv: `2303.08774 [cs.CL]`.
[24] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.
[25] B. Mehlig, *Machine Learning with Neural Networks*. Cambridge University Press, 2021.
[26] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[27] H. J. Kelley, "Gradient theory of optimal flight paths," *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
[29] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *CoRR, abs/1211.5063*, vol. 2, no. 417, p. 1, 2012.
[30] S. Hochreiter and J. Schmidhuber, "Long short–term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[31] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
[32] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.
[33] T. Wolf, L. Debut, V. Sanh, *et al.*, "Huggingface's transformers: State–of–the–art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.
[34] G. Grefenstette, "Tokenization," *Syntactic Wordclass Tagging*, pp. 117–133, 1999.
[35] J. Jiang and C. Zhai, "An empirical study of tokenization strategies for biomedical information retrieval," *Information Retrieval*, vol. 10, pp. 341–363, 2007.
[36] T. Kudo and J. Richardson, "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," *arXiv preprint arXiv:1808.06226*, 2018.
[37] Y. Wu, M. Schuster, Z. Chen, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.
[38] M. Schuster and K. Nakajima, "Japanese and korean voice search," in *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, IEEE, 2012, pp. 5149–5152.
[39] M. K. Dahouda and I. Joe, "A deep-learned embedding technique for categorical features encoding," *IEEE Access*, vol. 9, pp. 114 381–114 391, 2021.
[40] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.
[41] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.
[42] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.
[43] W. K. Sari, D. P. Rini, and R. F. Malik, "Text Classification Using Long Short-Term Memory with GloVe," *Jurnal Ilmiah Teknik Elektro Komputer dan Informatika (JITEKI)*, vol. 5, no. 1, pp. 85–100, 2019.
[44] L. Galke, A. Saleh, and A. Scherp, "Word embeddings for practical information retrieval," in *Informatik 2017*, Gesellschaft für Informatik, 2017, pp. 2155–2167.
[45] J. E. Font and M. R. Costa-Jussa, "Equalizing gender biases in neural machine translation with word embeddings techniques," *arXiv preprint arXiv:1901.03116*, 2019.
[46] N. S. Sushma and S. K. Sharma, "Effect of GloVe, Word2Vec and FastText Embedding on English and Hindi Neural Machine Translation Systems," in

Proceedings of Data Analytics and Management: ICDAM 2022, Springer, 2023, pp. 433–447.

[47] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the association for computational linguistics*, vol. 5, pp. 135–146, 2017.

[48] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[49] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese BERT-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[50] K. Tsumuraya, M. Amano, M. Uehara, and Y. Adachi, "Topic-Based Clustering of Japanese Sentences Using Sentence-BERT," in *2022 Tenth International Symposium on Computing and Networking Workshops (CANDARW)*, IEEE, 2022, pp. 255–260.

[51] D. Chicco, "Siamese neural networks: An overview," *Artificial neural networks*, pp. 73–94, 2021.

[52] S. K. Roy, M. Harandi, R. Nock, and R. Hartley, "Siamese networks: The tale of two manifolds," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3046–3055.

[53] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in *Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*, Springer, 2015, pp. 84–92.

[54] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[55] M. Grootendorst, "KeyBERT: Minimal keyword extraction with BERT," *Zenodo*, 2020.

[56] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[57] A. M. Lamb, A. G. ALIAS PARTH GOYAL, Y. Zhang, S. Zhang, A. C. Courville, and Y. Bengio, "Professor forcing: A new algorithm for training recurrent networks," *Advances in neural information processing systems*, vol. 29, 2016.

[58] T. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[59] F. Zhuang, Z. Qi, K. Duan, *et al.*, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[60] J. Quinn, J. McEachen, M. Fullan, M. Gardner, and M. Drummy, *Dive into deep learning: Tools for engagement*. Corwin Press, 2019, p. 551.

[61] J. Dodge, G. Ilharco, R. Schwartz, A. Farhadi, H. Hajishirzi, and N. Smith, "Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping," *arXiv preprint arXiv:2002.06305*, 2020.

[62] Y. Tsvetkov, *Opportunities and Challenges in Working with Low-Resource Languages*, 2017. [Online]. Available: `https://www.cs.cmu.edu/~ytsvetko/jsalt-part1.pdf`.

[63] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[64] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[65] J. Gu, K. Cho, and V. O. Li, "Trainable greedy decoding for neural machine translation," *arXiv preprint arXiv:1702.02429*, 2017.

[66] C. Meister, T. Vieira, and R. Cotterell, "If beam search is the answer, what was the question?" *arXiv preprint arXiv:2010.02650*, 2020.

[67] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, "The curious case of neural text degeneration," *arXiv preprint arXiv:1904.09751*, 2019.

[68] Z. Fu, W. Lam, A. M.-C. So, and B. Shi, "A theoretical analysis of the repetition problem in text generation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 12 848–12 856.

[69] A. Fan, M. Lewis, and Y. Dauphin, "Hierarchical neural story generation," *arXiv preprint arXiv:1805.04833*, 2018.

[70] C. Meister, T. Vieira, and R. Cotterell, "Best-first beam search," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 795–809, 2020.

[71] E. Cohen and C. Beck, "Empirical analysis of beam search performance degradation in neural sequence models," in *International Conference on Machine Learning*, PMLR, 2019, pp. 1290–1299.

[72] J. Ficler and Y. Goldberg, "Controlling linguistic style aspects in neural language generation," *arXiv preprint arXiv:1707.02633*, 2017.

[73] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for Boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[74] T. B. Hashimoto, H. Zhang, and P. Liang, "Unifying human and statistical evaluation for natural language generation," *arXiv preprint arXiv:1904.02792*, 2019.

[75] M. Caccia, L. Caccia, W. Fedus, H. Larochelle, J. Pineau, and L. Charlin, "Language GANs falling short," *arXiv preprint arXiv:1811.02549*, 2018.

[76] D. Pascual, B. Egressy, C. Meister, R. Cotterell, and R. Wattenhofer, "A plug-and-play method for controlled text generation," *arXiv preprint arXiv:2109.09707*, 2021.

[77] C. Meister, T. Pimentel, G. Wiher, and R. Cotterell, "Locally typical sampling," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 102–121, 2023.

[78] A. Lesne, "Shannon entropy: a rigorous notion at the crossroads between probability, information theory, dynamical systems and statistical physics," *Mathematical Structures in Computer Science*, vol. 24, no. 3, e240311, 2014.

[79] J. Eisenstein, "Natural language processing," *Jacob Eisenstein*, 2018.

[80] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn Treebank," 1993.

[81] T. Mikolov and G. Zweig, "Context dependent recurrent neural network language model," in *2012 IEEE Spoken Language Technology Workshop (SLT)*, IEEE, 2012, pp. 234–239.

[82] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[83] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing LSTM language models," *arXiv preprint arXiv:1708.02182*, 2017.

[84] C. Chelba, T. Mikolov, M. Schuster, *et al.*, "One billion word benchmark for measuring progress in statistical language modeling," *arXiv preprint arXiv:1312.3005*, 2013.

[85] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, "Exploring the limits of language modeling," *arXiv preprint arXiv:1602.02410*, 2016.

[86] R. Zellers, A. Holtzman, H. Rashkin, *et al.*, "Defending against neural fake news," *Advances in neural information processing systems*, vol. 32, 2019.

[87] K. Pillutla, S. Swayamdipta, R. Zellers, *et al.*, "MAUVE: Measuring the gap between neural text and human text using divergence frontiers," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4816–4828, 2021.

[88] J. M. Joyce, "Kullback-leibler divergence," in *International encyclopedia of statistical science*, Springer, 2011, pp. 720–722.

[89] A. Tatoeba, *Tatoeba*, 2006. [Online]. Available: `https://tatoeba.org/`.

[90] M. Davies, "The Corpus of Contemporary American English as the first reliable monitor corpus of English," *Literary and linguistic computing*, vol. 25, no. 4, pp. 447–464, 2010.

[91] J. A. Rice, *Mathematical statistics and data analysis*. Cengage Learning, 2006.

[92] HugginFace, *DistilGPT–2*, 2019. [Online]. Available: `https://huggingface.co/distilgpt2/`.

[93] Y. Liu, M. Ott, N. Goyal, *et al.*, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[94] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," *arXiv preprint arXiv:1906.02243*, 2019.

[95] A. H. Razavi, D. Inkpen, S. Uritsky, and S. Matwin, "Offensive language detection using multi-level classification," in *Advances in Artificial Intelligence: 23rd Canadian Conference on Artificial Intelligence, Canadian AI 2010, Ottawa, Canada, May 31–June 2, 2010. Proceedings 23*, Springer, 2010, pp. 16–27.

[96] Council of Europe, *Common European Framework of Reference for Languages (CEFR)*, 2023. [Online]. Available: `https://www.coe.int/en/web/common-european-framework-reference-languages/level-descriptions`.

A

Appendix 1

A.1 Removed Sentences

Table A.1 shows the ten sentences removed due to excessive length.

Table A.1: Sentences manually removed from dataset.

Beginning of Table	
Removed sentence	# of words
Only the assumption that the reader - I better say: the prospective reader, because for the moment there is not the slightest prospect, that my writing could see the lights of publicity, - unless it miraculously left our endangered fortress Europe and brought a hint of the secrets of our loneliness to those outside; - I beg to be allowed to begin anew: only because I anticipate the wish to be told casually about the who and what of the writer, I send some few notes on my own individuum out before these openings, - of course not without the awareness that exactly by doing so I might provoke doubts in the reader, that he is in the right hands, which is to say: if I, from all my being, am the right man for a task to which maybe the heart pulls me more than any qualifying relation in character.	151
Therefore, putting on one side imaginary things concerning a prince, and discussing those which are real, I say that all men when they are spoken of, and chiefly princes for being more highly placed, are remarkable for some of those qualities which bring them either blame or praise; and thus it is that one is reputed liberal, another miserly, using a Tuscan term (because an avaricious person in our language is still he who desires to possess by robbery, whilst we call one miserly who deprives himself too much of the use of his own); one is reputed generous, one rapacious; one cruel, one compassionate; one faithless, another faithful; one effeminate and cowardly, another bold and brave; one affable, another haughty; one lascivious, another chaste; one sincere, another cunning; one hard, another easy; one grave, another frivolous; one religious, another unbelieving, and the like.	144

Continuation of Table A.1	
Such abundance I accept apparent in this country, such top moral values, humans of such caliber, that I do not anticipate we would anytime beat this country, unless we breach the actual courage of this nation, which is her airy and cultural heritage, and, therefore, I adduce that we alter her old and age-old apprenticeship system, her culture, for if the Indians anticipate that all that is adopted and English is acceptable and greater than their own, they will lose their self-esteem, their built-in self-culture and they will become what we ambition them, an absolutely bedevilled nation.	97
Same time tomorrow, this time yesterday, same time next week, this time last week, same time next month, this time last month, same time next year, this time last year, never the same, this time tomorrow, same time yesterday, this time next week, same time last week, this time next month, same time last month, this time next year, same time last year, same time tomorrow, this time yesterday, same time next week, this time last week, same time next month, this time last month, same time next year, this time last year, never the same.	96
A good translator is one who, akin to a worm, can squirm with delight around in the author's brain mass, and leave through the tongue to have its taste impressions confirmed, and then leave well-formed traces on paper line after line, sheet after sheet, until the last page is done and he once again returns to his former self, and as a member of the smallest and least respected low-salary group, signs up yet again to exert his effort for God's representative on earth, the publishers.	87
What an unfailing barrier against vice, immorality and bad habits are those tastes which lead us to embellish a home, to which at all times and in all places we turn with delight, as being the object and the scene of our fondest cares, labours and enjoyments; whose humble roof, whose shady porch, whose verdant lawn and smiling flowers all breathe forth to us, in true, earnest tones, a domestic feeling that at once purifies the heart and binds us more closely to our fellow beings.	86
Another thing which makes preferable moving by foot or by bicycle is the use of a car: nowadays the level of air pollution is very high because of the high number of cars, and, to reduce that level, the municipality has decided to forbid the road traffic on certain days during a certain period; furthermore, there are always traffic jams on Reggio Emilia's roads, so you'll waste lots of time while stuck in a traffic jam.	76
As I stood upon the bluff before my cottage on that clear cold night in the early part of March, 1886, the noble Hudson flowing like the grey and silent spectre of a dead river below me, I felt again the strange, compelling influence of the mighty god of war, my beloved Mars, which for ten long and lonesome years I had implored with outstretched arms to carry me back to my lost love.	74

Continuation of Table A.1	
An Englishman, a Scotsman, an Irishman, a Welshman, a Gurkha, a Latvian, a Turk, an Aussie, a German, an American, an Egyptian, a Japanese, a Mexican, a Spaniard, a Russian, a Pole, a Lithuanian, a Jordanian, a Kiwi, a Swede, a Finn, an Israeli, a Romanian, a Bulgarian, a Serb, a Swiss, a Greek, a Singaporean, an Italian, a Norwegian, an Argentinian, a Libyan and a South African went to a night club.	73
A boy having sold a cow at the fair at Hereford, was way-laid by a highwayman, who at a convenient place demanded the money; on this the boy took to his heels and ran away but being overtaken by the highwayman, who dismounted, he pulled the money out of his pocket and strewed it about, and while the highwayman was picking it up, the boy jumped upon the horse and rode home.	72
End of Table	