NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

The Large Language Model GreekDeBERTa

Apostolos K. Koukouvinis

Supervisors: **Manolis Koubarakis,** Professor
Despina - Athanasia Pantazi, PhD Candidate

ATHENS

Month Year

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Το Μεγάλο Γλωσσικό Μοντέλο GreekDeBERTa

Απόστολος Κ. Κουκουβίνης

Επιβλέποντες: **Μανόλης Κουμπαράκης,** Καθηγητής
Δέσποινα – Αθανασία Πανταζή, Υποψήφια Διδάκτωρ

ΑΘΗΝΑ

Μήνας Έτος

BSc THESIS

The Large Language Model GreekDeBERTa

Apostolos K. Koukouvinis
S.N.: 1115202000098

SUPERVISORS: **Manolis Koubarakis,** Professor
Despina - Athanasia Pantazi, PhD Candidate

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Το Μεγάλο Γλωσσικό Μοντέλο GreekDeBERTa

Απόστολος Κ. Κουκουβίνης
Α.Μ.: 1115202000098

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Μανόλης Κουμπαράκης,** Καθηγητής
Δέσποινα – Αθανασία Πανταζή, Υποψήφια Διδάκτωρ

ABSTRACT

Language Models for Natural Language Processing (NLP) have constituted a significant area of research and development for over two decades, with a notable trend towards increasing model complexity and size. The general expectation is that larger models with more trainable parameters will achieve superior performance, which has been largely validated by the breadth of research and multitude of models introduced over the years. This thesis introduces GreekDeBERTa, a specialized language model for modern Greek, designed primarily to enhance accuracy in NLP tasks. To explore the effects of model size and training tasks on performance, three distinct models were developed: a base model trained on the Masked Language Modeling (MLM) task, a base model trained on the Replacement Token Detection (RTD) task, and a smaller model also trained on the RTD task. These variations allow for a comprehensive evaluation of how different configurations impact the models' effectiveness in various NLP applications.

GreekDeBERTa follows the architecture of the original DeBERTa model, which is known for its Disentangled Attention mechanism and Enhanced Mask Decoder. The architecture is designed to improve the model's ability to understand context and semantics by separating content and position information. This architecture is especially advantageous in handling complex language structures, making it an ideal choice for developing a model tailored to the intricacies of the Greek language.

After developing and pre-training our model on a substantial dataset, we evaluated its performance on three widely used downstream NLP tasks: Part of Speech Tagging (PoS Tagging), Named Entity Recognition (NER), and Natural Language Inference (NLI). Given that GREEK-BERT utilized the F1 score to measure accuracy, we also adopted this metric to facilitate direct comparisons. Our results revealed that all three of our models surpassed GREEK-BERT across all tasks. Specifically, the $\text{GreekDeBERTa}_{\text{base}}$ MLM model achieved an F1 score of 98.4% in PoS tagging, outperforming GREEK-BERT's 98.1%. For the NER task, the same model reached an F1 score of 86.5%, exceeding GREEK-BERT's 85.7%. In the NLI task, the $\text{GreekDeBERTaV3}_{\text{base}}$ RTD variant led with an F1 score of 80.0%, compared to GREEK-BERT's 78.6%.

The development of these models marks a significant exploration of DeBERTa variants for the Greek language, setting new standards in NLP tasks. The DeBERTa models demonstrated superior capabilities in handling the complexities of the Greek language, surpassing the previously leading GREEK-BERT model. While the base variants of DeBERTa offer exceptional accuracy, they are resource-intensive. The $\text{GreekDeBERTaV3}_{\text{xsmall}}$ RTD model, though slightly behind in performance, provides a practical alternative with about 51% faster processing, making it suitable for applications where computational efficiency is paramount.

Overall, the introduction of GreekDeBERTa models represents an advancement in Greek NLP. This work not only improves accuracy across multiple tasks but also highlights the versatility of DeBERTa architecture in adapting to underrepresented languages like Greek, demonstrating significant potential for future applications.

SUBJECT AREA: Artificial Intelligence, Machine Learning, Natural Language Processing

KEYWORDS: DeBERTa, Named Entity Recognition, Natural Language Inference, Part of Speech Tagging, Disentangled Attention, Enhanced Mask Decoder, Masked Language Modeling, Replacement Token Detection

ΠΕΡΙΛΗΨΗ

Τα γλωσσικά μοντέλα για την επεξεργασία φυσικής γλώσσας (NLP) αποτελούν σημαντικό τομέα έρευνας και ανάπτυξης για περισσότερες από δύο δεκαετίες, με έντονη τάση προς την αύξηση της πολυπλοκότητας και του μεγέθους των μοντέλων. Η γενική προσδοκία είναι ότι τα μεγαλύτερα μοντέλα με περισσότερες παραμέτρους εκμάθησης θα επιτύχουν ανώτερη απόδοση, κάτι που έχει σε μεγάλο βαθμό επικυρωθεί από την έκταση της έρευνας και το πλήθος των μοντέλων που έχουν εισαχθεί με την πάροδο των χρόνων. Αυτή η πτυχιακή παρουσιάζει το GreekDeBERTa, ένα εξειδικευμένο γλωσσικό μοντέλο για τη σύγχρονη ελληνική γλώσσα, σχεδιασμένο κυρίως για να βελτιώσει την ακρίβεια στις εργασίες NLP. Για να εξερευνήσουμε τις επιπτώσεις του μεγέθους του μοντέλου και των εργασιών εκπαίδευσης στην απόδοση, αναπτύχθηκαν τρία διακριτά μοντέλα: ένα βασικό μοντέλο εκπαιδευμένο στην εργασία Masked Language Modeling (MLM), ένα βασικό μοντέλο εκπαιδευμένο στην εργασία Replacement Token Detection (RTD) και ένα μικρότερο μοντέλο επίσης εκπαιδευμένο στην εργασία RTD. Αυτές οι παραλλαγές επιτρέπουν μια ολοκληρωμένη αξιολόγηση του πώς διαφορετικές διαμορφώσεις επηρεάζουν την αποτελεσματικότητα των μοντέλων σε διάφορες εφαρμογές NLP.

Το GreekDeBERTa ακολουθεί την αρχιτεκτονική του αρχικού μοντέλου DeBERTa, το οποίο είναι γνωστό για τους μηχανισμούς Disentangled Attention και Enhanced Mask Decoder. Η αρχιτεκτονική έχει σχεδιαστεί για να βελτιώσει την ικανότητα του μοντέλου να κατανοεί τα συμφραζόμενα και τη σημασιολογία, διαχωρίζοντας τις πληροφορίες περιεχομένου και θέσης. Αυτή η αρχιτεκτονική είναι ιδιαίτερα αποδοτική στην αντιμετώπιση σύνθετων γλωσσικών δομών, καθιστώντας την ιδανική επιλογή για την ανάπτυξη ενός μοντέλου προσαρμοσμένου στις ιδιαιτερότητες της ελληνικής γλώσσας.

Αφού αναπτύξαμε και προεκπαιδεύσαμε το μοντέλο μας σε ένα εκτεταμένο σύνολο δεδομένων, αξιολογήσαμε την απόδοσή του σε τρεις ευρέως χρησιμοποιούμενες κατηγορίες εργασιών NLP: Επισήμανση των Μερών του Λόγου (PoS Tagging), Αναγνώριση Ονοματοποιημένων Οντοτήτων (NER) και Εξαγωγή Λογικού Συμπεράσματος (NLI). Δεδομένου ότι το GREEK-BERT χρησιμοποίησε τη μετρική F1 για τη μέτρηση της ακρίβειας, υιοθετήσαμε και εμείς αυτή τη μετρική για να διευκολύνουμε άμεσες συγκρίσεις. Τα αποτελέσματά μας έδειξαν ότι και τα τρία μοντέλα μας ξεπέρασαν το GREEK-BERTσε όλες τις εργασίες. Συγκεκριμένα, το μοντέλο $\text{GreekDeBERTa}_{\text{base}}$ MLM πέτυχε βαθμολογία F1 98,4% στην εργασία PoS, υπερβαίνοντας το 98,1% του GREEK-BERT. Στην εργασία NER, το ίδιο μοντέλο έφτασε σε βαθμολογία F1 86,5%, ξεπερνώντας το 85,7% του GREEK-BERT. Στην εργασία NLI, η παραλλαγή $\text{GreekDeBERTaV3}_{\text{base}}$ RTD είχε την καλύτερη απόδοση με βαθμολογία F1 80,0%, σε σύγκριση με το 78,6% του GREEK-BERT.

Η ανάπτυξη αυτών των μοντέλων σηματοδοτεί μια σημαντική εξερεύνηση των παραλλαγών DeBERTa για την ελληνική γλώσσα, καθορίζοντας νέα πρότυπα στις εργασίες NLP. Τα μοντέλα DeBERTa έδειξαν εξαιρετικές ικανότητες στην αντιμετώπιση των πολυπλοκοτήτων της ελληνικής γλώσσας, υπερβαίνοντας το προηγουμένως κυρίαρχο μοντέλο GREEK-BERT. Ενώ οι βασικές παραλλαγές του DeBERTa προσφέρουν εξαιρετική ακρίβεια, είναι απαιτητικές σε πόρους. Το μοντέλο $\text{GreekDeBERTaV3}_{\text{xsmall}}$, αν και ελαφρώς υστερεί στην απόδοση, παρέχει μια πρακτική εναλλακτική λύση με περίπου 51% ταχύτερη επεξεργασία, καθιστώντας το κατάλληλο για εφαρμογές όπου η αποδοτικότητα σε πόρους είναι κρίσιμης σημασίας.

Συνολικά, η εισαγωγή των μοντέλων GreekDeBERTa αντιπροσωπεύει μια πρόοδο στην

ελληνική NLP. Αυτό το έργο όχι μόνο βελτιώνει την ακρίβεια σε πολλαπλές εργασίες, αλλά και υπογραμμίζει την ευελιξία της αρχιτεκτονικής DeBERTa στην προσαρμογή σε υποεκπροσωπούμενες γλώσσες όπως τα ελληνικά, δείχνοντας σημαντικό δυναμικό για μελλοντικές εφαρμογές.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τεχνητή Νοημοσύνη, Μηχανική Μάθηση, Επεξεργασία Φυσικής Γλώσσας

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: DeBERTa, Αναγνώριση Ονοματοποιημένων Οντοτήτων, Εξαγωγή Λογικού Συμπεράσματος, Επισήμανση των Μερών του Λόγου, Disentangled Attention, Enhanced Mask Decoder, Masked Language Modeling, Replacement Token Detection

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my professor, Manolis Koubarakis, whose courses on Artificial Intelligence, Machine Learning, and Natural Language Processing introduced me to this fascinating field. His teachings and encouragement have been crucial in shaping my academic path and inspiring this thesis.

I would also like to extend my sincere thanks to my thesis supervisor, PhD candidate Despina-Athanasia Pantazi, for her constant support and guidance throughout the entire process. Her assistance, from introducing me to the subject of this thesis to patiently answering every question, has been invaluable. I am truly grateful for her advice and dedication.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The evolution of artificial intelligence (AI) and natural language processing (NLP) has been profoundly influenced by the advent of transformer-based models. These models, epitomized by architectures like BERT (Bidirectional Encoder Representations from Transformers), revolutionized NLP by employing a bidirectional training mechanism. This approach enables the model to consider the context from both directions, significantly enhancing the understanding of language. BERT's architecture includes multiple layers of self-attention, allowing the model to weigh the importance of different words in a sentence relative to one another, thus capturing various aspects of linguistic structure and meaning. This bidirectional nature, combined with the extensive pre-training on large corpora, has set new benchmarks in various NLP tasks such as question answering and language inference.

Building on the foundation laid by BERT, DeBERTa (Decoding-enhanced BERT with disentangled attention) introduced significant improvements. DeBERTa incorporates a disentangled attention mechanism, which separates the representation of word content and position, thus allowing the model to handle the complexities of language more effectively. Additionally, it features an enhanced mask decoder, which aids in better capturing the dependencies in the text. These innovations have enabled DeBERTa to surpass its predecessors in terms of accuracy.

This thesis explores the development and evaluation of GreekDeBERTa, a specialized variant of the DeBERTa model pre-trained in Greek language. The research focuses on adapting the DeBERTa architecture to better capture the nuances and complexities of modern Greek, which is an underrepresented language in the NLP field. Three different models are trained and compared: a base model trained on Masked Language Modeling (MLM), another base model trained on Replacement Token Detection (RTD), and a smaller model also trained on RTD. The thesis evaluates these models across several NLP tasks, including Part of Speech Tagging, Named Entity Recognition, and Natural Language Inference, comparing their performance with existing models like GREEK-BERT.

This thesis consists of the following chapters :

Chapter 2: Background
The second chapter delves into the theoretical foundations of artificial intelligence and machine learning. It covers various types of machine learning, including supervised, unsupervised, and reinforcement learning, and discusses essential NLP tasks. The chapter also introduces neural networks and deep learning, setting the stage for understanding advanced models like transformers.

Chapter 3: Related Work
This chapter reviews the literature on existing NLP models, focusing on BERT and its variants, including GREEK-BERT and DeBERTa. It outlines the key advancements and limitations of these models, providing a context for the development of GreekDeBERTa.

Chapter 4: GreekDeBERTa
The fourth chapter introduces the GreekDeBERTa model, describing its architecture and the methodologies employed for its training. It covers topics about the server resources, data cleaning, preprocessing, and the creation of a specialized vocabulary for modern Greek.

Chapter 5: Evaluation Tasks

This chapter evaluates the performance of GreekDeBERTa on several NLP tasks, including Part of Speech Tagging, Named Entity Recognition, and Natural Language Inference. It provides a comparative analysis with other models like GREEK-BERT, using F1 score to demonstrate the improvements.

Chapter 6: Conclusions
The final chapter provides and overview and evaluation of the whole thesis.

2. BACKGROUND

This chapter presents an overview of the key concepts and technologies that form the foundation of this research. It covers the evolution and applications of Artificial Intelligence (AI), Machine Learning (ML), Natural Language Processing (NLP), Deep Learning, Neural Networks, and Transformer architectures, which are crucial for understanding the topic discussed in this thesis.

2.1 Artificial Intelligence

Artificial Intelligence (AI) refers to simulation of human intelligence to machines, providing them problem-solving capabilities. The term AI encompasses numerous technologies, including machine learning (ML) and natural language processing (NLP). AI can be broadly classified into two categories : Wake AI and Strong AI [34]. Weak AI is trained and focuses to perform specific tasks, whereas Strong AI represents a theoretical form of AI in which a machine would possess intelligence equivalent to that of humans. This concept begins with Alan Turing's work, where by posing the question "Can Machines think?", he starts the exploration of the limits of machine intelligence, proposing the idea that machines may exhibit behavior indistinguishable from that of a human [78]. The initial formalization of AI field was done by John McCarthy, who first proposed the term 'artificial intelligence', setting the foundations for the evolution of the field in a more academic manner [44].

Over the years, the reputation and funding of AI experienced fluctuations, going through a number of active and inactive cycles, often described as Summers and Winters accordingly. Initially, from 1957 to 1974, AI flourished as computers were becoming faster and cheaper , with optimistic statements about being close to create machines "with the general intelligence of an average human being" [12], these were the years of the first AI Summer. However, after a nearly 20-year period of huge interest, the lack of computational resources combined with the false predictions and exaggerations by the experts [11], led to the first AI Winter - a period marked by reduced funding and interest in AI research, this period lasted roughly 6 years, between 1974 and 1980. Interest reignited in 1980's with the emerge of expert systems which mimicked the decision making process of a human expert : the program would deduce rules based on answers of experts in different fields. During the years 1980 to 1987, the funding raised again, resulting in the second Summer of AI. Unfortunately, most of the ambitious goals were not met and companies would not align with the specialized needs of expert systems, resulting in the harsh second Winter of AI. During this period, even the term AI was avoided and the investments in the field of AI were limited. However, this was the time when the backpropagation algorithm was revisited, starting a new era in Neural Networks (NN) and generally in AI[68]. Finally, the development of Support Vector Machines (SVMs) [15], the improvement on NN techniques and computational hardware combining with the raise of Internet that resulted in bigger amounts of data, led to the end of the second Winter of AI. This marked the beginning of a new era of regulated optimism in AI that we are living in.

2.2 Machine Learning

Machine learning (ML) is a branch of AI and computer science that focuses on the using data and algorithms to enable AI to imitate the way that humans learn, gradually improving its accuracy[36]. The main distinction between ML and traditional programming, is that ”rules” are not explicitly coded by humans : ML discover patterns in data automatically using a ML algorithm. In the end of the day, ML defines a mathematical representation of a problem : a model. ML algorithm must have a model characterized by parameters which are adjusted during learning process. During the training process the model will make some decisions, which are problem defined, based on the data given. Initially, those decisions will be random, or more accurately defined by the initial values of the parameters, the weights, of the model. While the training continues, weights change based on how accurate the decision was. To measure accuracy, ML algorithms use an error function that evaluates the decision of the model. Based on the evaluation, the algorithm will use a method to adjust the weights, this is called the optimization function. The training process could last from seconds to months, depending on the size of the model and how accurate we want the model to be in the training data. This is the process of fitting : Fitting refers to adjusting the values of the parameters in the model to improve accuracy. The goal of fitting is to create a model that performs well on unseen data, meaning that the model needs to generalize well in new data . Ultimately, generalization of a model to new data is what makes ML useful in computer science. Overfitting occurs when a model learns the training data too well, including its noise and outliers, resulting in poor performance on new data. Conversely, underfitting happens when a model is too simple to capture the underlying patterns in the data, leading to inadequate performance both on the training data and new data [5].

2.2.1 Types of Machine Learning

ML algorithms can be divided into five categories [30]:

- **Supervised machine learning :** In this type of ML, the model is trained on labeled data : there are the input variables (usually feature vectors) and the output targets (labels), the algorithm learns the mapping function from inputs to outputs by discovering hidden patterns. While this type of learning can achieve high levels of accuracy, it may be really difficult to acquire sufficient amount of labeled data. Supervised learning can be broadly divided into two main categories [58] :

    1. **Classification :** In classification tasks, the goal is to predict a discrete label or category for a given input, for example ’positive’ or ’negative’, ’spam’ or ’not-spam’. We could further divide the classifiers to Binary Classifiers and Multi-class Classifiers [75].
    2. **Regression :** In regression tasks, the objective is to predict a continuous value based on input data, for example given the features of a house to predict its value. We could further divide Regression algorithms into Linear and Non-linear Regression.

- **Unsupervised machine learning :** Unsupervised learning is a type of ML where the model is trained on data without labeled responses : the data is available only in the form of an input and there is no corresponding output variable. The primary goal is

to identify patterns, structures, or relationships within the data [20]. Unlabeled data is often easier to be acquired since there is no need for prior processing, however those algorithms are highly dependent on the quality of the input data : noisy data can significantly affect the model's accuracy. The most common technique in unsupervised learning is cluster analysis : grouping of data points into clusters based on their features relying on the natural groupings that are formed within a dataset.

- **Self-supervised machine learning :** self-supervised (SSL) is a type of ML model that learns from unlabeled data by generating its own labels, this is done by finding and exploiting relationships between the various input features. The goal of self-supervised learning is to minimize the need for labeled data by yielding 'pseudo-labels' from unlabeled data [40]. After creating the 'pseudo-labels', the model uses these pseudo-labels to learn patterns and features from the data. This phase is treated like supervised learning, where the model attempts to minimize the error between its predictions and the pseudo-labels. In essence, SSL leverages the structured learning process of supervised learning without needing external labeled data, thus combining the strengths of both supervised and unsupervised approaches.

- **Reinforcement learning :** reinforcement learning is a type of ML that trains an agent to make decisions in order to achieve the optimal result. It operates on the principles of trial and error, exploring various actions to discover the most rewarding strategies. The agent receives feedback in the form of rewards or penalties, which it uses to update its knowledge and improve future actions [9]. One of the challenges that arise in reinforcement learning, and not in other kinds of learning, is the trade-off between exploration and exploitation. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. The agent has to exploit what it already knows in order to obtain reward, but it also has to explore in order to make better action selections in the future [73]. Reinforcement learning is based on Markov Decision Process, a mathematical modeling of decision-making problems, characterized by states, actions, rewards, and transition probabilities. The agent learns by moving through states in each time step, taking actions based on a policy, and updating its knowledge with rewards received, thus improving its decision-making over time. Reinforcement learning algorithms are common in video game development and are frequently used to teach robots how to replicate human tasks.

- **Semi-supervised learning :** As the name suggests, this is an intermediate level between supervised and unsupervised learning techniques, by using both labeled and unlabeled data [41]. This method is used in situations that sufficient labeled data are hard to find but a large amount of unlabeled data is available. Semi-supervised learning uses the limited labeled data to infer patterns in unlabeled data improving its efficiency.

2.2.2 Datasets, Cleaning and Feature Engineering

ML models use training datasets to learn patterns, validation datasets to tune hyperparameters and prevent overfitting, and test datasets to evaluate their performance on unseen data. Usually, the validation set is a part of the dataset (10-30% of it) that is not used for training, but instead it is used to evaluate how well the model inferences with unseen data [6]. However, data found in the real world may be noisy, incomplete, or inconsistent, necessitating cleaning to remove errors and ensure that the data is appropriate to be

used. Key steps of data cleaning include modifying and removing incorrect and incomplete data fields, identifying and removing duplicate information and unrelated data, and correcting formatting, missing values, and spelling errors. Furthermore, a crucial step in enhancing the model's performance is feature engineering, which involves transforming raw data into meaningful features that better represent the underlying patterns. Feature engineering methods include techniques such as creating new features through mathematical transformations, combining existing features, and encoding categorical variables [17].

2.2.3 Evaluation and Optimization

ML aims to optimize the accuracy of the model. Accuracy needs somehow to be measured : the loss function, also known as the cost or objective function, quantifies the difference between the predicted outputs of a model and the actual target values. So ML models can then be trained by optimizing the value of the loss function [54]. Finding the suitable loss function to be used can be tricky and depends largely on the nature of the problem and the model being used. There is a variety of loss function families, common examples include Mean Squared Error (MSE) for regression tasks and Cross-Entropy Loss for classification tasks. Having found a way to represent the loss of the model in mathematical form, optimization algorithms are employed to adjust the model parameters to minimize the loss function, thereby improving the model's performance. The primary objective is to find the optimal set of parameters that lead to the lowest possible loss, indicating the highest model accuracy. Most of the deep model training is still based on the back propagation algorithm, which propagates the errors from the output layer backward and updates the variables layer by layer with the gradient descent based optimization algorithms [87]. This iterative method adjusts the parameters in the opposite direction of the gradient of the loss function with respect to the parameters.

2.3 Natural Language Processing

Natural Language Processing (NLP) is a field of AI that focuses on the interaction between computers and human (natural) languages, enabling computers to understand, interpret, and generate human language in a valuable way. It uses a variety of computational techniques in order to bridge the gap between human communication and computer understanding [67].

2.3.1 History

We could divide the history of NLP in 4 phases [49] :

1. **Phase 1: Late 1940s to Late 1960s** The first phase of NLP, spanning from the late 1940s to the late 1960s, was primarily focused on machine translation. Despite the enthusiasm and optimism of this period, researchers faced considerable challenges due to the limited power of early computational systems, with small storage and processing power. During the first phase of NLP, methods used included dictionary-based word-for-word translation and the development of autonomous sentence grammars and parsers to resolve syntactic and semantic ambiguities. Despite limited

computing resources, a lot of the foundational work was done through this period, establishing key techniques for syntactic analysis and lexicon building. However, the period ended with the infamous 1966 ALPAC Report, which criticized the lack of progress in Machine Translation and led to a significant reduction in funding for NLP research.

2. **Phase 2: Late 1960s to Late 1970s** The second phase of NLP, spanning from the late 1960s to the late 1970s, was influenced by the developments in AI, focusing on incorporating world knowledge and inference into NLP systems. This phase emphasized in semantic understanding and the construction of meaning representations, via methods like semantic networks, scripts, and case frames to represent and process knowledge. While the work done during this period was very optimistic, the results were usually domain-specific, due to the limitations in computational power, being really hard to create a general-purpose NLP model. The phase concluded as researchers recognized the difficulties in building robust, general-purpose NLP systems.

3. **Phase 3: Late 1970s to Late 1980s** The third phase of NLP, spanning from the late 1970s to the late 1980s, emphasized in grammatico-logical methods, focusing on formal grammatical theories and logic-based knowledge representation. This phase emphasized syntax-driven compositional interpretation into logical forms, supported by efficient parsing algorithms derived from context-free base grammar theories. It was during this period that the statistical NLP started, an approach that gave birth to the NLP as we know it today. This kind of algorithms were based on statistics and probability theory, leveraging large corpora of text to model language patterns and improve tasks such as speech recognition, parsing, and machine translation.

4. **Phase 4: Late 1980s Onward** The fourth phase, spanning from late 1980s to early 1990s, is marked by the shift to statistical language data processing. Researchers began to collect and analyze large amounts of text data to identify patterns and improve language understanding. This era saw the rise of techniques like probabilistic tagging and ML, which allowed systems to learn from data rather than relying solely on predefined rules. These advancements led to significant improvements in NLP applications, making them more accurate and adaptable for tasks such as speech recognition, machine translation, and information retrieval.

2.3.2 Neural Networks in NLP

In it first steps, NLP relied heavily on rule-based systems and statistical methods, which involved extensive manual labor and were limited in their ability to generalize across different languages and contexts. This changed with the developments in NN that enabled models to learn directly from data and discover patterns in the data itself, without the need of human guidance [29]. Since ML and NN algorithms cannot handle non-numeric input, there is a need to represent words and sentences using numbers. The two naive approaches would be to encode each word with a unique number (One-Hot Encoding) or counting the frequency of words in different text fragments (Bag-of-Words) [83]. Both methods result in high-dimensional, sparse (mostly zero) data and are unable to reuse information learned from previous appearances of a word. This changed with the introduction of word embeddings. Word embeddings are dense vector representations of words in a continuous vector space. These embeddings capture semantic similarities between words by placing similar

words closer together in this space. Models like Word2Vec [76] and GloVe [47] analyze the context in which words appear in large corpora of text and provide word representations that are dense and can store information about a word based on the context. NN have further revolutionized NLP through the development of more complex architectures that are going to be discussed in next sections.

2.3.3 NLP tasks

Through the development of NLP, several tasks have been proposed in order to enable machines to understand, interpret, and respond to human language. Some of those tasks include [39] [46] :

- **Speech Recognition (or Speech-to-Text):** the task of converting voice data to text data. Speech recognition works by converting spoken language into text using AI and ML algorithms to analyze sound patterns and linguistic structures. It is widely used in fields such as healthcare, automotive, and consumer technology. Challenges include accurately recognizing diverse accents, handling background noise, and differentiating between similar-sounding words. [42]
- **Parts of speech tagging:** Parts of Speech (POS) tagging involves assigning each word in a text to a particular grammatical category, such as noun, verb, adjective, etc., which helps in understanding the structure and meaning of sentences. This process is used in fields such as information retrieval, text-to-speech systems, and machine translation, but it faces challenges like handling ambiguous words and accurately tagging in different linguistic contexts [57].
- **Word Sense Disambiguation:** Word Sense Disambiguation (WSD) is the task of determining the correct meaning of a word in context, used for improving the accuracy of NLP applications such as machine translation and information retrieval. WSD faces difficulties due to polysemy, where words have multiple meanings, and the need for large datasets to train effective disambiguation models. [59]
- **Named entity recognition:** Named Entity Recognition (NER) identifies and classifies key entities in text, such as names of individuals, organizations, and locations, for better information extraction and text analysis. NER is used in various fields including healthcare, finance, and cybersecurity to improve data organization and decision-making processes. NER difficulties include handling ambiguous entities, nested entities, and varying entity representations across different contexts and languages. [37]
- **Sentiment analysis:** Sentiment analysis is a the process of determining the emotional tone of digital text, classifying it as positive, negative, or neutral. It is used in fields like customer service, brand monitoring, and market research to analyze large volumes of text data, such as social media comments and reviews. Challenges include accurately interpreting sarcasm, handling negations, and dealing with multipolarity within sentences [10].
- **Machine translation:** Machine translation is used to translate text from one language to another, maintaining the meaning and context of the original text. It is used for applications such as global communication, real-time customer service, and data analysis. Challenges include maintaining contextual accuracy and the need for large datasets for training effective models [8].

- **Natural language generation:** Text generation is used to create human-like text, useful in applications like content creation, customer service, and language translation. It leverages techniques such as NN and transformer-based models to produce coherent and contextually appropriate text. Challenges include maintaining quality, ensuring diversity, and addressing ethical concerns [43].

There are many other NLP tasks, but these are the most common and closer to the subject of this thesis.

2.4 Deep Learning

Deep learning is a subset of ML that focuses on algorithms inspired by the structure and function of the human brain, known as artificial neural networks (ANN). These networks consist of multiple layers, enabling the the model to capture complex patterns in large datasets. The deep learning process involves feeding data through these layers, where each layer extracts higher-level features from the raw input, progressively processing the information. This progression of computations through the network is called forward propagation [35]. The model then uses backpropagation, and adjusts the weights and biases of the network by calculating errors and propagating them backward through the layers to minimize these errors. This iterative process allows the model to learn from the data, gradually enhancing its accuracy [29]. The output layer of a deep learning model is the final layer where the network's predictions or classifications are made, based on the results of the previous layers. Deep learning is used in image and speech recognition, enabling technologies such as facial recognition systems and virtual assistants like Siri and Alexa. It is also applied in NLP tasks, such as language translation and sentiment analysis, and in autonomous driving systems to interpret sensor data and make real-time driving decisions [7]. The most common approach in deep learning is the use of NN, which are discussed in the next section.

2.5 Neural Networks

Neural networks (NN) are models that simulate the way the human brain processes information, using layers of nodes, or neurons. These networks are composed of an input layer, one or more hidden layers, and an output layer, where each node in a layer is connected to nodes in the subsequent layer, each with associated weights. By passing data through these layers and adjusting the connections based on errors, NN can learn to recognize patterns, classify data, and make decisions similar to the human brain. [32]. By being non linear, NN can capture complex patterns in data and with sufficient training and data, state of the art models can be created offering high accuracy in many tasks.

2.5.1 Modelization of brain

The human brain functions as a complex, highly efficient information-processing system, with billions of neurons interconnected by trillions of synapses. Neurons communicate through electrical impulses known as action potentials, which propagate along axons and are transmitted across synapses via chemical signals. This network enables the brain to receive and perceive information, make decisions, and respond to stimuli. NN model this

biological system by using artificial neurons and synapses to process information. These networks mimic brain functions by learning from data, adjusting synaptic weights, and producing outputs that simulate human decision-making processes. Although ANN are much simpler than the brain, they are inspired by its structure and mechanisms. The activation function in a neural network defines the output of a neuron based on its input, modeling the way biological neurons process signals. There are mainly two types of activation functions: the threshold function and the sigmoid function. The threshold function outputs a binary value based on whether the input exceeds a certain threshold. The sigmoid function provides a smooth transition between 0 and 1, allowing for more accurate outputs and enabling the model to capture complex, nonlinear relationships. This function is also differentiable, which is crucial for training NN using backpropagation [28].

2.5.2 Architecture of Neural Networks

A typical neural network has three types of layers: the input layer, hidden layers, and the output layer. The input layer receives the raw data, the hidden layers perform computations and extract features, and the output layer generates the final prediction or classification. [2]. Activation functions determine the output of a neuron by applying a mathematical transformation to the input signals. The input layer is the first layer of a neural network and is responsible for receiving the initial data. Each neuron in this layer represents a feature of the input data. For example, in image processing, each neuron might represent a pixel's intensity value. Hidden layers, may be absent in some very simple NN models, however in practice, to increase accuracy, multi-layered approaches are being followed : the neurons are arranged in layered fashion, in which the input and output layers are separated by a group of hidden layers [2]. Information moves only in one direction, from the input layer through any hidden layers and finally to the output layer. This is the simplest kind of artificial neural network using no cycles or different directions of information spreading [18], this type of NN are called Feed Forward NN. Activation functions play a crucial role in the functioning of NN. They introduce non-linearity into the network, allowing it to model complex data patterns. Some of them are presented in the image below : Activation functions take the weighted sum of inputs as their input and apply a mathematical transformation to produce an output that determines whether a neuron should be activated. This output can range from 0 to 1 for the sigmoid function, -1 to 1 for the tanh function, and from 0 to the input value for the ReLU function, allowing the network to model complex, non-linear relationships in the data. Typically, in classification tasks, a special function called the Softmax function is used to convert raw outputs into probabilities. It then chooses the highest probability to make the final classification.

2.5.3 Training a Neural Network

NN are theoretically capable of learning any mathematical function with sufficient training data, and some variants like recurrent NN are known to be Turing complete. Turing completeness refers to the fact that a neural network can simulate any learning algorithm, given sufficient training data [2]. In practice, the data needed to achieve this and the training time is extraordinarily large to do that. Training a neural network involves the process of optimizing the network's parameters so that it can accurately map inputs to the desired outputs. This process is iterative and it can be broken up into 4 steps :

1. **Forward Propagation :** we talk about it in the previous subsection, it is the the

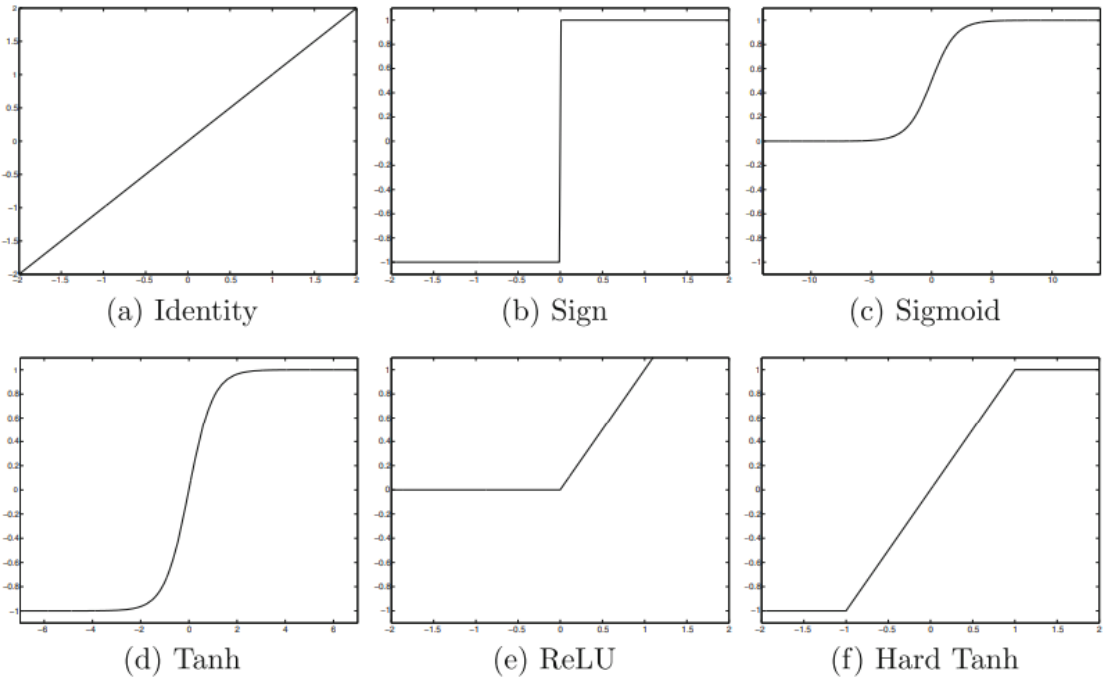


Figure 2.1: Various activation functions

process of passing the input data through the hidden layers of the network to obtain the output.

2. **Loss Computation :** The loss function, also known as the cost function, is a way to quantify the discrepancy between the network's prediction and the actual target. Some of the most common loss functions are Mean Squared Error and Cross Entropy Loss. It is important to note that the correct output must be known to compute this function, meaning the training data should be labeled. Alternatively, the neural network model can produce the 'correct output' using techniques like pseudo-labels.

3. **Backpropagation :** In multi-layered NN the loss is a complicated composition function of the weights in earlier layers [2]. During the backward phase, the goal is to learn the gradient of the loss function with respect to the different weights by using the chain rule of differential calculus. Then gradients are propagated backward from the output layer to the input layer, adjusting the weights and biases along the way. The further the distance between two layers, the smaller the impact the gradient has on them during backpropagation. This phenomenon, known as the vanishing gradient problem, occurs because gradients tend to diminish as they are propagated backward through the network, making it difficult for the earlier layers to learn effectively [4].

4. **Weight Updates :** Once the gradients are computed, the network's weights are updated to minimize the loss. This is typically done using an optimization algorithm such as gradient descent [60]. Key component in this step is the Learning Rate : it dictates the magnitude of the steps the model takes during gradient descent [16]. A large learning rate can lead to faster convergence but risks overshooting the optimal solution, whereas a small learning rate ensures more precise convergence but can result in significantly slower training. Some of the most common gradient descent algorithms are Stochastic Gradient Descent, Mini-batch Gradient Descent,

and Adam, while there are some non-gradient descent algorithms like Genetic Algorithms and Simulated Annealing, which explore different optimization strategies and can be useful in scenarios where gradient descent might struggle.

In conventional NN, the primary components that undergo change during training are the weights and biases of the neurons, the other components remain the same.

2.5.4 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are designed for sequential data like text sentences and time-series and the connections between their units form a directed cycle, creating an internal state that allows them to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs, making them capable of learning dependencies over time. They achieve this by maintaining a hidden state that captures information about previous elements in the sequence, allowing the network to make informed predictions about future elements [69]. RNNs are implemented to handle sequential data by maintaining a memory of previous inputs, which influences their current output. This is done by using loops within their architecture, where the output from one time step is fed as input to the next. RNNs are trained using Backpropagation Through Time (BPTT), an extension of backpropagation suited for sequence data. [31] RNNs were proposed to address the limitations of traditional NN in

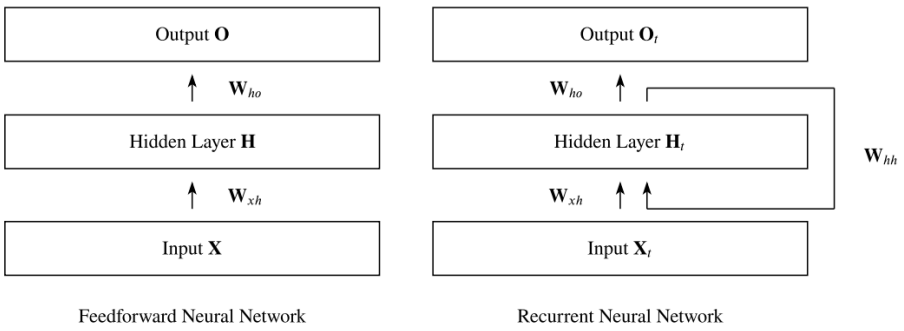


Figure 2.2: Visualization of differences between Feedfoward NNs and Recurrent NNs

handling sequential data, where the order and context of inputs are crucial. However, despite their promising capabilities, RNNs face significant challenges. One of the primary limitations is the difficulty in learning long-term dependencies due to the vanishing gradient problem, where gradients of the loss function decrease exponentially during backpropagation through time, making it hard for the network to learn and retain information over long sequences [55]. To deal with these issues, variants like Long Short-Term Memory (LSTM) networks were developed which are discussed in the next subsection.

2.5.5 Long Short-Term Memory Neural Networks

Long Short-Term Memory (LSTM) networks are a type of RNN specifically designed to address the limitations of standard RNNs in learning long-term dependencies. Traditional

RNNs struggle with the vanishing gradient problem, where gradients diminish exponentially during backpropagation through time, making it difficult to learn and remember long-range dependencies in sequential data. LSTMs overcome this by incorporating a memory cell and three gating mechanisms (input gate, output gate, and forget gate) that regulate the flow of information, enabling the network to maintain and update its cell state effectively over long sequences [70].
The core of an LSTM is its memory cell, which retains information over arbitrary time intervals. The input gate controls the extent to which new information flows into the cell, the forget gate determines what portion of the past cell state should be discarded, and the output gate regulates the output from the cell. This architecture allows LSTMs to selectively remember or forget information, effectively managing the issue of vanishing gradients. Consequently, LSTMs are highly effective for tasks involving long-term dependencies, such as language modeling, machine translation, and time-series forecasting [27].
The implementation of LSTM networks involves constructing a network with a series of LSTM cells, each containing input, output, and forget gates that control the flow of information. During training, the cell states are updated at each time step based on the current input and the previous hidden state. The forget gate decides which information from the previous cell state to discard, the input gate determines what new information to store, and the output gate controls the output to the next LSTM cell. This architecture allows LSTMs to maintain and update their internal states, enabling them to learn complex temporal patterns over long sequences. The network is typically trained using backpropagation through time (BPTT), which adjusts the weights of the gates to minimize prediction errors over the training data. Despite their complexity, frameworks like TensorFlow and PyTorch provide efficient implementations of LSTMs, making them accessible for practical applications in various domains [65][74].
Despite their advantages, LSTMs have certain limitations. They are computationally intensive due to their complex architecture and require significant computational resources for training and inference. Additionally, LSTMs involve many hyperparameters (e.g., number of layers, number of units per layer, learning rates) that need careful tuning to achieve optimal performance. This tuning process can be time-consuming and requires substantial expertise. Moreover, while LSTMs mitigate the vanishing gradient problem, they can still face challenges with very long sequences and may not always capture extremely long-term dependencies as effectively as more advanced models like the Transformer architecture [72].

2.6 Transformers

The Transformer model, introduced in the seminal 2017 paper ”Attention is All You Need” [81], revolutionized the field of deep learning by eliminating the need for recurrence in NN. Unlike RNNs and LSTMs, which process data sequentially, Transformers utilize a self-attention mechanism to process input sequences in parallel, significantly enhancing computational efficiency and enabling the handling of long-range dependencies in data. This architecture has become foundational in NLP and has been applied to various tasks including machine translation, text generation, and more [33].

Transformers utilize two primary techniques: positional encoding and self-attention. Positional encoding assigns unique positional information to tokens, allowing the model to understand the order of the sequence. Self-attention enables the model to weigh the

importance of each token in relation to others in the sequence, capturing contextual relationships efficiently. These innovations allow Transformers to excel in tasks requiring the understanding of complex patterns and long-range dependencies, making them a powerful tool in modern AI applications such as OpenAI's GPT and Google's BERT models.

2.6.1 Attention

The attention mechanism was introduced in the paper "Neural Machine Translation by Jointly Learning to Align and Translate" [14], which fundamentally changed how NN handle sequential data. This method allows the model to focus on relevant parts of the input sequence dynamically, enhancing its ability to capture dependencies by aligning and translating simultaneously. This innovation laid the groundwork for subsequent developments in neural network architectures, particularly in improving the performance and efficiency of models used in tasks such as machine translation and text generation. Some of the important concepts introduced in the paper "Attention is All You Need" are :

- **Query - Key - Value :** Before continuing, we need to make clear what are these components and how they are used [80].

  The **query** vector represents the element of interest or the context from the current position in the input sequence or the previous layer's output. It is used to determine the similarity or relevance between this context and other elements in the input sequence, specifically the key vectors. For instance, when translating "apple" from English to French, "apple" is the query, the keys are representations of all words in the English sentence, and the values are their corresponding translations in French.

  The **key** vector, like the query vector, is a projection of the input data and is associated with each element in the input sequence. Key vectors determine the relevance of each input element to the query, typically calculated using a dot product or another similarity measure. For example, in translating "apple," key vectors could be representations of words like ["cat," "apple," "tree," "juice"].

  The **value** vector, like the key vector, is a projection of the input data and is associated with each element in the input sequence. It stores the information used to update the query's representation, weighted by attention scores derived from query-key interactions. Higher attention scores increase the importance of corresponding value vectors in the final output. For example, in translating "apple," value vectors might be ["chat," "pomme," "arbre," "jus"], representing the corresponding French translations.

- **Scaled dot-product Attention :** attention mechanism that computes attention weights using a scaled dot product to measure the similarity between vectors. The scaling prevents attention weights from becoming excessively large, and the output is a weighted sum of the values, determined by the calculated attention weights. The final output is computed by the following formula :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

  with d_k the keys dimension

- **Multi-head attention :** attention mechanism that employs multiple scaled dot-product attention heads operating in parallel, each with distinct query, key, and value matrices.

This setup enables the model to attend to various parts of the input simultaneously, enhancing its ability to learn complex relationships within the data. The individual outputs from each head are concatenated to form a richer representation of the input. For instance, in translating ”I love my dog” from English to French, different heads might focus on different word pairs, producing a more nuanced and accurate translation.

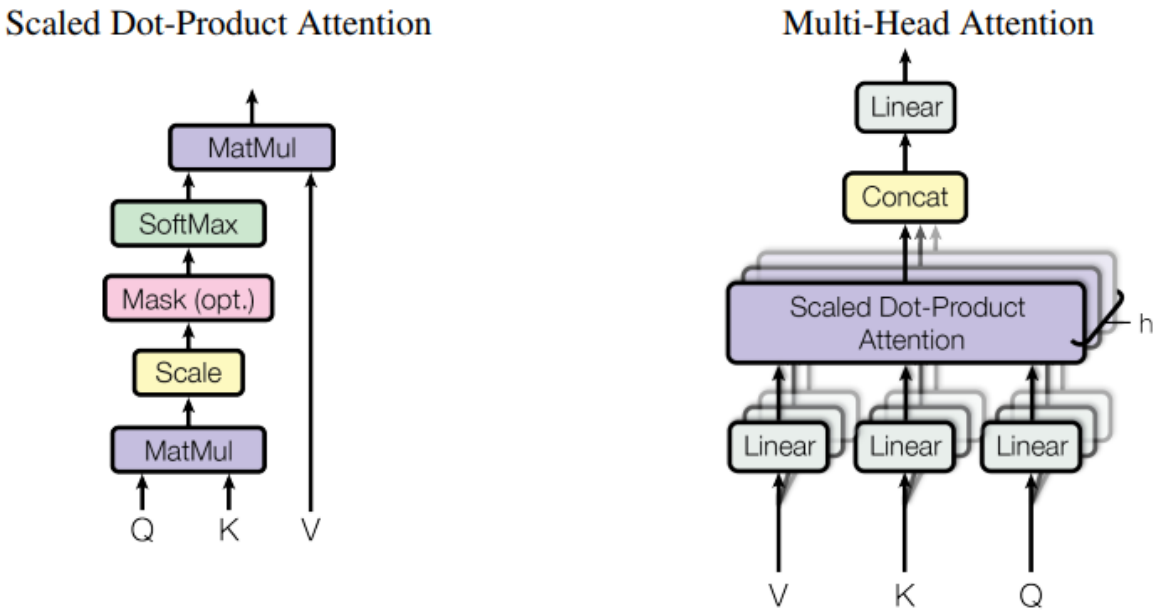


Figure 2.3: Attention mechanism

- **Self-attention :** attention mechanism allowing a neural network to focus on different parts of its own input, crucial for handling long and complex texts in NLP tasks. It uses the scaled dot-product attention mechanism where the query, key, and value matrices are identical, enabling the model to attend to different parts of the same input sequence.

2.6.2 Transformer Architecture

Transformer utilize an encoder-decoder structure. The encoder maps an input sequence of symbols into continuous representations, which the decoder then uses to generate an output sequence of symbols, one element at a time. This process is auto-regressive, meaning the model uses previously generated symbols as additional input for generating the next symbol. The Transformer model follows this architecture but employs stacked self-attention and fully connected layers for both the encoder and decoder.

The Transformer’s encoder consists of a stack of six identical layers, each comprising a multi-head self-attention mechanism followed by a position-wise fully connected feed-forward network. Residual connections and layer normalization are applied around each sub-layer, maintaining an output dimension of 512. Similarly, the decoder includes six identical layers, with an additional sub-layer performing multi-head attention over the encoder’s output. The decoder’s self-attention mechanism is modified to prevent future positions from being attended to, ensuring that predictions for each position depend only on the known outputs of preceding positions [21].

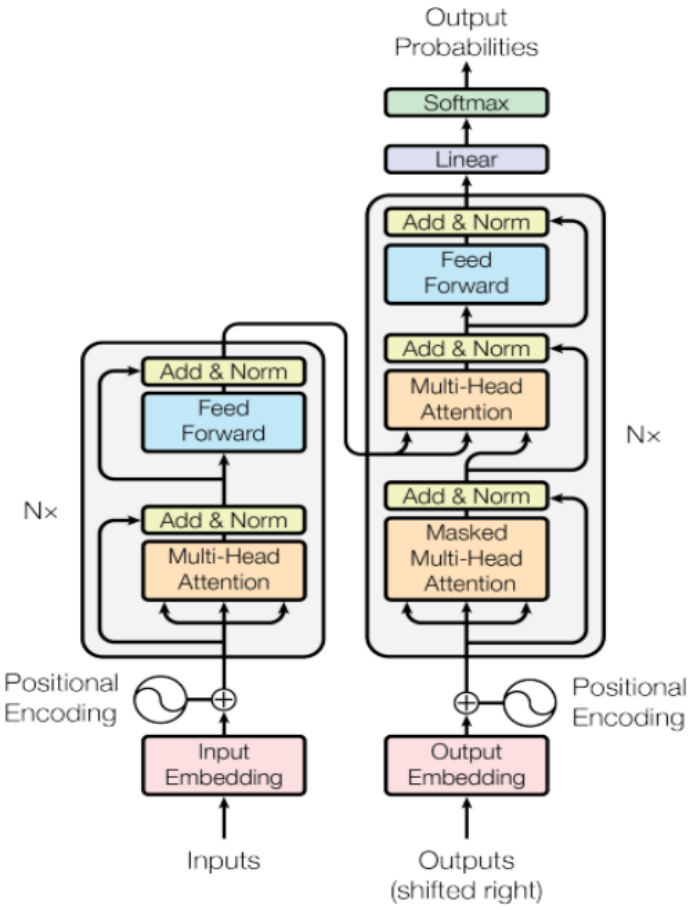


Figure 2.4: Transformer architecture

In addition to attention sub-layers, each layer in the Transformer's encoder and decoder includes a fully connected feed-forward network applied identically to each position. This network consists of two linear transformations with a ReLU activation in between. Since the Transformer model, unlike RNN, lacks recurrence and convolution, it must incorporate information about the sequence order to process tokens effectively. This is achieved by adding "positional encodings" to the input embeddings in both the encoder and decoder stacks. These positional encodings enable the model to utilize sequence order by summing the encodings with the embeddings. The positional encodings are derived from sine and cosine functions of different frequencies, allowing the model to attend to relative positions easily [81].

3. RELATED WORK

This chapter provides an overview of the foundational research and models that have contributed to the development of advanced natural language processing systems. It explores notable models such as BERT, GREEK-BERT, DeBERTa, and their variations, highlighting the evolution of transformer-based models and their impact on NLP tasks.

3.1 BERT

BERT [45] (Bidirectional Encoder Representations from Transformers) is a transformer-based model implemented by Google's AI Language team that pre-trains deep bidirectional representations from unlabeled text by conditioning on both left and right context in all layers. Transformers, the core architecture of BERT, utilize self-attention mechanism, which allows the model to weigh the importance of different words in a sentence relative to each other, regardless of their position. The attention mechanism in BERT is characterized by multiple attention heads within its transformer layers. These heads focus on different parts of a sentence, capturing various aspects of the linguistic structure and meaning. Each attention head in BERT can attend to different elements within the input sequence, such as specific tokens, positional offsets, or even broad sentence-wide patterns [51].

BERT can be fine-tuned with a single additional output layer to achieve state-of-the-art results on various NLP tasks without substantial task-specific architecture modifications. BERT builds on existing techniques by addressing the limitations of unidirectional language models in pre-training representations, which are particularly restrictive for fine-tuning approaches. To overcome this, BERT employs a masked language model (MLM) pre-training objective, where some tokens in the input are randomly masked, and the model predicts these tokens based on their surrounding context. This enables BERT to pre-train deep bidirectional representations, enhancing its ability to capture relationships from both directions. In addition to the MLM, BERT introduces a "next sentence prediction" task to pre-train text-pair representations, further improving its utility for tasks involving pairs of sentences. BERT follows a two-step procedure : pre-training and fine-tuning. During pre-training, BERT is trained on unlabeled data across various tasks, and for fine-tuning, it is initialized with pre-trained parameters and adjusted using labeled data from downstream tasks. Each downstream task has its own fine-tuned model, though they all start from the same pre-trained parameters.

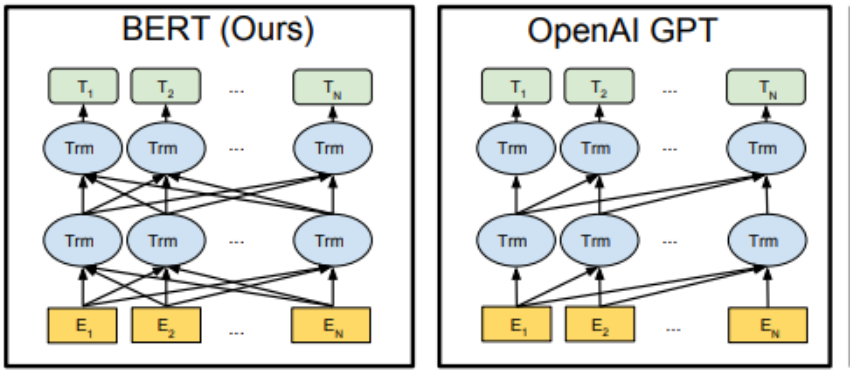


Figure 3.1: Bidirectional vs left-to-right

3.1.1 Pre-training

Two unsupervised tasks are used to pre-train BERT:

- **Masked LM** It is reasonable to believe that a deep bidirectional model is inherently more powerful than unidirectional models or a combination of both directions. However, traditional language models can only be trained in one direction to avoid the model predicting the target word by ”seeing itself.” To train a bidirectional representation, BERT employs a technique called masked language modeling (MLM), where random tokens in the input are masked and the model predicts these masked tokens based on the context provided by the surrounding words. While this approach allows for effective bidirectional pre-training, it creates a conflict between pre-training and fine-tuning because the [MASK] token is absent during fine-tuning. To address this, BERT replaces masked tokens with the [MASK] token only 80% of the time, with random tokens 10% of the time, and leaves them unchanged 10% of the time. This strategy deals with the mismatch and ensures that the model learns representations that are effective during fine-tuning for downstream tasks. The advantage of this procedure is that the Transformer encoder must maintain a contextual representation for every input token, as it doesn't know which tokens it will need to predict or which have been randomly replaced, and because random replacements only occur for 1.5% of all tokens, it minimally impacts the model's language understanding capability.
- **Next Sentence Prediction (NSP)** Understanding the relationship between two sentences is crucial for many downstream tasks such as Question Answering (QA) and Natural Language Inference (NLI). To train a model that can comprehend sentence relationships, BERT uses a pre-training task called next sentence prediction (NSP). In this task, pairs of sentences are created from a monolingual corpus where 50% of the time, the second sentence follows the first (labeled as IsNext), and the other 50% of the time, it is a random sentence (labeled as NotNext). This task helps the model learn sentence relationships effectively.

3.1.2 Embeddings

BERT uses WordPiece embeddings [85], these embeddings, developed by Google, eliminate the problem of out-of-vocabulary words that often plague other models such as Word2Vec and GloVe [66].Each sequence in BERT begins with a special classification token ([CLS]), and sentence pairs are combined into a single sequence, distinguished by a special token ([SEP]) and a learned embedding indicating whether each token belongs to sentence A or B. For any given token, its input representation is constructed by summing its corresponding token, segment, and position embeddings. Token embeddings capture the identity of the subword units, segment embeddings differentiate between different sentences in tasks involving pairs of sentences, and position embeddings encode the position of each token in the sequence. By summing these embeddings, BERT constructs a rich input representation for each token that includes context, segment, and positional information. In BERT, a fixed vocabulary of 30,000 tokens is used for WordPiece embeddings. These embeddings break down words into smaller, more manageable pieces, allowing the model to represent rare words as combinations of more common subwords. This approach not only mitigates the out-of-vocabulary problem but also enhances the model's ability to understand and generate language more effectively [45].

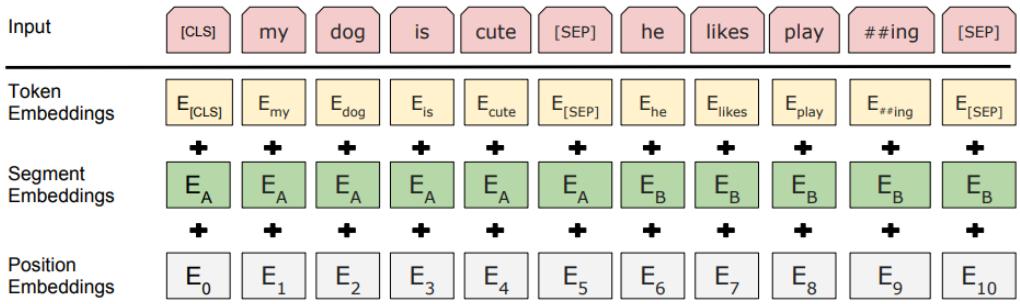


Figure 3.2: BERT input representation.

3.1.3 Fine-tuning

Fine-tuning BERT is straightforward due to the self-attention mechanism in the Transformer, which allows BERT to adapt to various downstream tasks by simply adjusting the inputs and outputs. During fine-tuning, task-specific inputs and outputs are plugged into BERT, and the model parameters are updated end-to-end. Fine-tuning BERT is relatively inexpensive compared to pre-training. It can be completed within an hour on a single Cloud TPU or a few hours on a GPU, starting from the pre-trained model. This efficiency makes BERT highly adaptable and practical for various NLP tasks, enabling it to achieve state-of-the-art results with minimal additional computational cost.

3.1.4 Models

The BERT models provided from the original paper are :

- **BERT$_{\text{base}}$**:
  - Layers (L): 12
  - Hidden size (H): 768
  - Attention heads (A): 12
  - Total Parameters: 110M
- **BERT$_{\text{large}}$**:
  - Layers (L): 24
  - Hidden size (H): 1024
  - Attention heads (A): 16
  - Total Parameters: 340M

3.1.5 Experiments

The BERT fine-tuning experiments present impressive results across a range of NLP tasks, demonstrating significant advancements in natural language understanding (NLU) [45].

- GLUE Benchmark The General Language Understanding Evaluation (GLUE) benchmark consists of diverse NLP tasks that measure the model's understanding and interpretation capabilities. BERT's fine-tuning approach uses the final hidden vector corresponding to the [CLS] token as the aggregate representation, introducing only the classification layer weights during fine-tuning. The results indicate that both $\text{BERT}_{\text{base}}$ and $\text{BERT}_{\text{large}}$ outperform all prior state-of-the-art models across all tasks in the GLUE benchmark. $\text{BERT}_{\text{large}}$, in particular, shows significant improvements, especially in tasks with smaller datasets, due to its larger model size and capacity to capture more nuanced information.
- The Stanford Question Answering Dataset (SQuAD) tasks involve predicting answer spans within a given passage. BERT employs a simple method, representing the input question and passage as a single sequence. The model introduces start and end vectors to predict the answer span, achieving improvements over existing systems. In SQuAD v1.1, $\text{BERT}_{\text{large}}$, both as a single model and an ensemble, outperforms the top leaderboard systems. SQuAD v2.0 extends the task by allowing for questions with no answer, where BERT also excels, showing a notable +5.1 F1 improvement over the previous best system.
- The Situations With Adversarial Generations (SWAG) dataset tests grounded commonsense inference by requiring the model to choose the most plausible continuation of a given sentence. BERT's approach constructs input sequences with sentence pairs and uses the [CLS] token's representation to score each choice. $\text{BERT}_{\text{large}}$ achieves a remarkable improvement over the ESIM+ELMo baseline and the OpenAI GPT model.

3.2 GREEK-BERT

GREEK-BERT [48] is a monolingual BERT-based language model specifically designed for the modern Greek language. Its architecture follows the BERT-BASE-UNCASED model, which consists of 12 layers of Transformer encoders, 768 hidden units, and 12 attention heads. GREEK-BERT was pre-trained on 29 GB of Greek text from diverse sources, including Greek Wikipedia, Europarl, and OSCAR, a cleaned version of Common Crawl. This extensive pre-training allows GREEK-BERT to capture the unique linguistic characteristics of modern Greek, resulting in high-quality, context-aware representations of sub-word tokens and entire sentences.

3.2.1 Motivation and NLP Resource Gap for Greek

The development of GREEK-BERT addresses the gap in natural language processing (NLP) resources for the Greek language, which has traditionally lagged behind resource-rich languages like English. Multilingual models, such as M-BERT and XLM-R, include Greek but only allocate a small portion of their vocabulary to Greek sub-words. This limited allocation leads to excessive word fragmentation in these models. GREEK-BERT, in contrast, uses a vocabulary of 35,000 sub-words specifically tailored for the Greek language, significantly reducing fragmentation and improving text understanding.

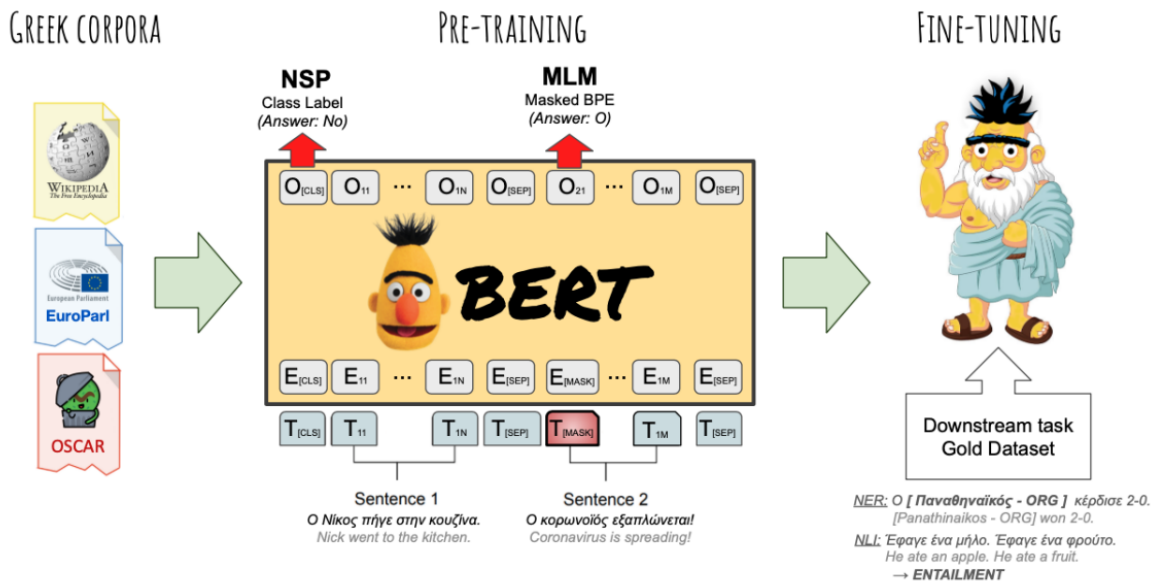


Figure 3.3: GREEK-BERT Architecture

3.2.2 Pre-training Corpus and Vocabulary

GREEK-BERT was pre-trained on a diverse corpus of 29 GB of Greek text, derived from the following sources:

- Greek Wikipedia
- The Greek portion of the European Parliament Proceedings Parallel Corpus (Europarl)
- OSCAR, a cleaned version of Common Crawl

The model's vocabulary consists of 35,000 sub-words created using the Byte-Pair Encoding (BPE) technique, tailored specifically for the Greek language. This focus on a single language enhances GREEK-BERT's ability to process and generate accurate Greek text.

3.2.3 Performance and Comparison with Multilingual Models

GREEK-BERT has demonstrated state-of-the-art performance across several NLP tasks, including part-of-speech (PoS) tagging, named entity recognition (NER), and natural language inference (NLI). It outperforms multilingual models, such as M-BERT and XLM-R, by margins of 5-10%.

After pre-training, GREEK-BERT was fine-tuned and evaluated on three core downstream tasks:

- **Part-of-Speech (PoS) Tagging:** Achieved an accuracy of 98.02% - 98.18%.
- **Named Entity Recognition (NER):** Achieved an F1 score between 84.7% - 86.7%.
- **Natural Language Inference (NLI):** Reached accuracy scores between 77.98% - 79.2%.

Despite its strong performance in NER and NLI, GREEK-BERT showed slightly lower results in PoS tagging when compared to other models, though the difference was marginal.

3.2.4 Future Directions

The release of GREEK-BERT, along with its training code, offers a valuable resource for the NLP community. The developers aim to encourage further research and applications in Greek NLP, providing an opportunity to develop more sophisticated language processing tools. Future work may include pre-training on larger corpora or extending GREEK-BERT for earlier forms of the Greek language, such as classical Greek.

3.3 DistilBERT: A Distilled Version of BERT

In recent years, large-scale pre-trained models have revolutionized NLP, significantly improving performance across a wide range of tasks. However, these models, such as BERT come with the challenge of high computational costs and large memory requirements, which limit their deployment in resource-constrained environments. To address this, *DistilBERT* [82] was introduced as a distilled version of BERT. Distillation is a compression technique where a smaller model, referred to as the *student*, is trained to replicate the behavior of a larger *teacher* model or an ensemble of models [26].

DistilBERT reduces the size of the original BERT model by 40%, while retaining 97% of its performance on various downstream tasks. Additionally, it is 60% faster at inference time, making it a more practical option for real-time applications on edge devices. This is achieved through knowledge distillation applied during the pre-training phase, rather than post-training, enabling the smaller model to capture the inductive biases of the larger BERT model. The result is a model that maintains the flexibility of BERT in a more computationally efficient form.

3.3.1 Architecture and Training Process

The architecture of DistilBERT follows the Transformer model [81], similar to BERT, but with several modifications. The number of layers is reduced by half, while components like the token-type embeddings and the pooler are removed to streamline the architecture. Importantly, the student network is initialized from the teacher by selecting every other layer from the original BERT model. This initialization helps the student model converge more efficiently during training. DistilBERT is trained using a triple loss function that combines the masked language modeling loss (L_{MLM}), the distillation loss (L_{CE}), and a cosine distance loss (L_{COS}) that aligns the hidden states of the student and teacher models. This combination of losses ensures that the distilled model captures both the linguistic knowledge from the training data and the knowledge transfer from the larger model.

3.3.2 Performance and Efficiency

DistilBERT was benchmarked on various NLP tasks, demonstrating impressive results. For example, it retained 97% of BERT's performance on the GLUE benchmark and in

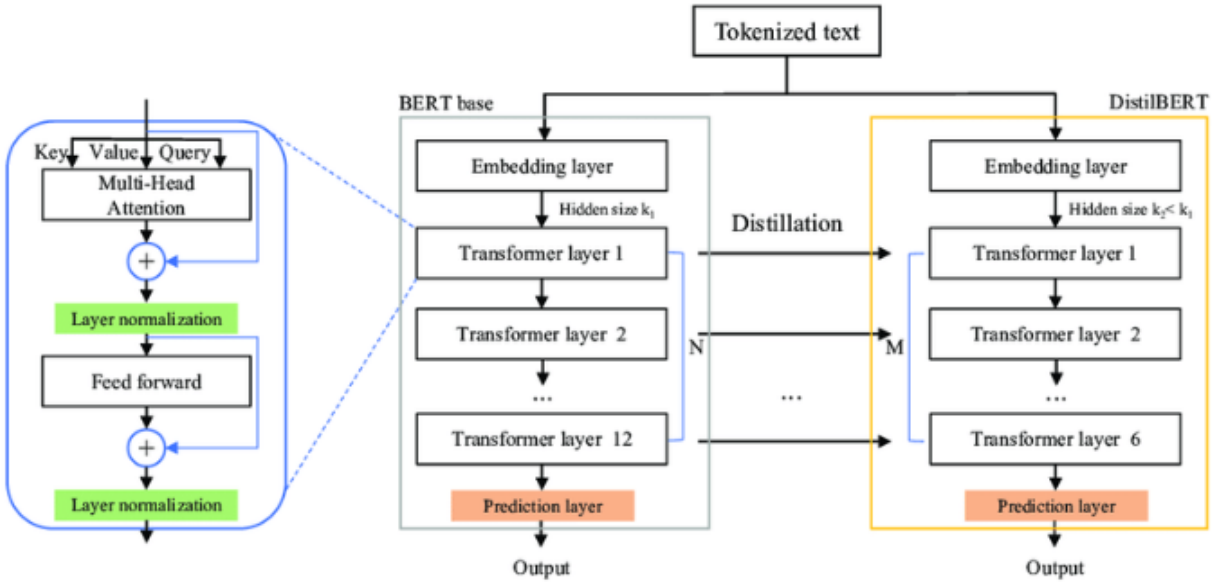


Figure 3.4: The DistilBERT model architecture and components.

question answering tasks such as SQuAD v1.1, DistilBERT achieved competitive results with BERT, despite being significantly smaller.

One of the key advantages of DistilBERT is its efficiency. With 40% fewer parameters than BERT, it is 60% faster in inference, making it ideal for scenarios where computational resources are limited, such as mobile devices. This efficiency was further demonstrated in a proof-of-concept mobile application, where DistilBERT achieved a 71% speed-up over BERT in real-time question answering tasks on an iPhone 7 Plus .

3.3.3 Distillation Procedure

The distillation process utilizes large batch training with dynamic masking and gradient accumulation over up to 4,000 examples per batch. The student model is trained on the same corpus as BERT, including English Wikipedia and the Toronto Book Corpus [86], and requires significantly less computational power compared to models like RoBERTa [84]. Specifically, DistilBERT's training was completed on 8 V100 GPUs over 90 hours, highlighting the cost-effectiveness of the model in terms of both computational and energy resources.

In conclusion, DistilBERT presents a compelling solution for deploying state-of-the-art NLP models in resource-constrained environments, offering a practical trade-off between model size, computational efficiency, and performance.

3.4 DistilGREEK-BERT

In this chapter, we introduce *DistilGREEK-BERT* [50], a distilled version of the GREEK-BERT model, which aims to offer a lighter and less complex language model for modern Greek. The following sections outline the architecture, pre-training, and fine-tuning processes of DistilGREEK-BERT, as well as its comparison to the original GREEK-BERT model.

3.4.1 Architecture and Data Sources

DistilGREEK-BERT follows the architecture of the DistilBERT model [82]. The goal of this distillation process is to reduce the complexity of GREEK-BERT while retaining most of its performance, making it suitable for applications where computational efficiency is essential. DistilGREEK-BERT is pre-trained under the supervision of GREEK-BERT, with knowledge distillation as the central technique.

Both GREEK-BERT and DistilGREEK-BERT were pre-trained on data sourced from Greek-language corpora, including the Greek version of Wikipedia, the European Parliament Proceedings Parallel Corpus (Europarl) , and OSCAR [**?**], a clean version of Common Crawl. The total dataset used for pre-training was 27GB, consisting of 2.473 billion words, which was split into training (25.5GB) and evaluation (1.5GB) sets.

3.4.2 Data Preprocessing

Before pre-training, the raw data underwent extensive preprocessing to ensure consistency and usability. HTML-like tags, such as document, speaker, and paragraph markers, were removed. Sentences were tokenized using the `nltk` package , and non-Greek characters, including numbers, URLs, and symbols, were filtered out. Additional cleaning steps included removing single characters (except for Greek articles like ”o” and ”η”), converting all text to lowercase, and truncating long sentences to a maximum of 512 tokens. After preprocessing, the final dataset consisted of 10,304 files, with 9,581,283 sentences in the training set and 710,883 sentences in the evaluation set.

3.4.3 Data Encoding

The preprocessed data was then tokenized using a modified version of the `BertTokenizer` from HuggingFace. The tokenizer is based on the WordPiece algorithm [85], which splits words into subword tokens. For consistency, the vocabulary used by GREEK-BERT, which contains 35,000 tokens, was adopted for DistilGREEK-BERT. This approach prevented discrepancies between the two models and ensured that the tokens shared between them remained aligned, avoiding the need to re-pretrain GREEK-BERT with a new vocabulary.

3.4.4 Fine-Tuning of GREEK-BERT

In order to supervise the training of DistilGREEK-BERT, GREEK-BERT was fine-tuned on the updated dataset. Fine-tuning was conducted on the training set (25.5GB), with periodic evaluations on the 1.5GB evaluation set. The training utilized a batch size of 8 and applied gradient accumulation to overcome the limited capacity of the GPU. During this process, MLM loss decreased from 2.53 to 2.03, indicating steady progress. Despite limited resources, the fine-tuned GREEK-BERT was able to impart substantial knowledge to DistilGREEK-BERT, which performed comparably on downstream tasks.

3.4.5 Pre-Training DistilGREEK-BERT

The pre-training of DistilGREEK-BERT was conducted using the fine-tuned GREEK-BERT as a teacher model. This involved initializing the student model by copying specific layers

from the teacher model. DistilGREEK-BERT, with approximately 40% fewer parameters than GREEK-BERT, underwent pre-training for three epochs using a combination of MLM loss, knowledge distillation loss, and cosine embedding loss. Despite the reduced complexity of DistilGREEK-BERT, the model achieved an accuracy of 0.56, only 4% lower than its teacher.

In summary, DistilGREEK-BERT demonstrates that it is possible to maintain competitive performance in Greek-language NLP tasks while significantly reducing model size and training time.

3.4.6 Results

After pre-training, DistilGREEK-BERT was evaluated on several common NLP tasks. These included PoS Tagging, NER, and NLI. The same datasets were used for the fine-tuning and evaluation of both DistilGREEK-BERT and GREEK-BERT to ensure fair comparison.

For PoS tagging, the Greek Universal Dependencies Treebank (GUDT) was used. The model achieved high accuracy (97%) on this task, showing that it can perform similarly to GREEK-BERT, which achieved 98.2% [48]. The fine-tuning process took about 2 minutes per experiment with DistilGREEK-BERT, compared to 4 minutes for GREEK-BERT, showing that DistilGREEK-BERT was approximately 60% faster.

In the NER task, the model identified and classified entities like Person, Organization, and Location from Greek text. DistilGREEK-BERT achieved an F1-score of 83.5%, closely matching the performance of GREEK-BERT, which had an F1-score of 85.7% [48]. DistilGREEK-BERT was also 66% faster than GREEK-BERT in fine-tuning, making it a compelling alternative for tasks requiring faster performance without significant loss in accuracy.

For the NLI task, the model was tested on the Cross-lingual Natural Language Inference (XNLI) corpus , which requires determining whether two sentences are related by entailment, contradiction, or neutrality. DistilGREEK-BERT achieved an accuracy of 74.47%, retaining about 95% of the performance of GREEK-BERT, which had an accuracy of 78.6%. While this task proved more challenging than the others, DistilGREEK-BERT was still 52% faster in fine-tuning compared to GREEK-BERT.

3.5 DeBERTa

DeBERTa (Decoding-enhanced BERT with disentangled attention) [61] is a model that improves upon the BERT and RoBERTa models through two innovative techniques. These techniques are the disentangled attention mechanism and an enhanced mask decoder. The disentangled attention mechanism represents each word with two vectors encoding its content and position, respectively, and computes attention weights using separate matrices for content and relative position. The enhanced mask decoder incorporates absolute positions in the decoding layer for predicting masked tokens during model pre-training. Additionally, a new virtual adversarial training method is employed during fine-tuning to improve the model's generalization capabilities.

3.5.1 Disentangled attention

Unlike BERT, where each word in the input layer is represented by a single vector combining its content and position embeddings, DeBERTa uses two separate vectors for each word to encode its content and position. The attention weights among words are computed using disentangled matrices based on their contents and relative positions. This method acknowledges that the attention weight between a pair of words depends not only on their contents but also on their relative positions. For example, the dependency between the words ”deep” and ”learning” is stronger when they occur next to each other compared to when they are in different sentences. This approach allows for a more nuanced understanding of word relationships.

DeBERTa introduces a more sophisticated approach [22] by using two separate vectors for each token to encode its content (H_i) and relative position ($P_{i|j}$) with respect to another token at position j. This method allows for a more nuanced representation of tokens and their relationships.

The calculation of cross-attention scores between tokens i and j in DeBERTa is decomposed into four components:

$$A_{i,j} = H_i H_j^T + H_i P_{j|i}^T + P_{i|j} H_j^T + P_{i|j} P_{j|i}^T \tag{3.1}$$

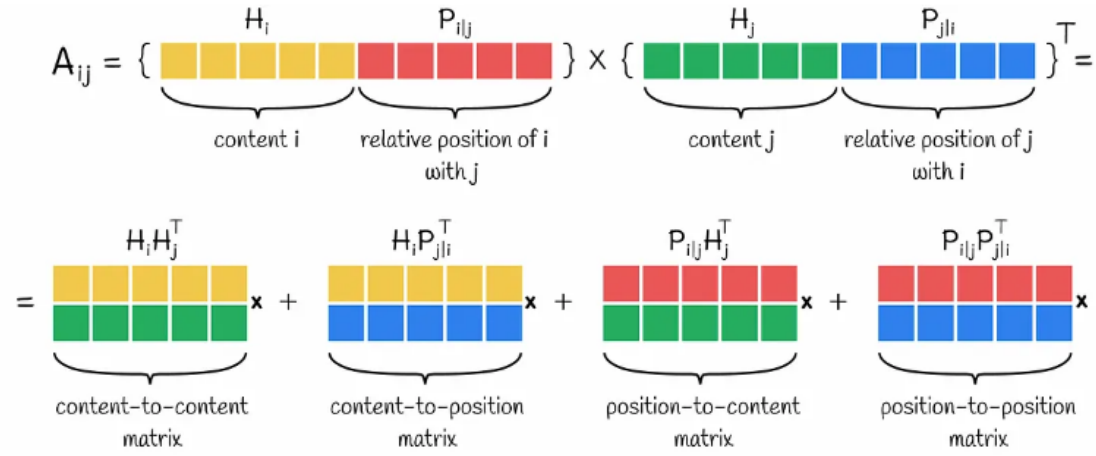


Figure 3.5: Computation of cross-attention score between two embedding vectors.

This decomposition includes content-to-content, content-to-position, position-to-content, and position-to-position interactions. However, because the position-to-position term does not provide significant additional information in the context of relative position embeddings, it is omitted in the implementation.

In existing models, relative position encoding typically uses a separate embedding matrix to compute relative position biases, which corresponds to using only the content-to-content and content-to-position terms. DeBERTa argues for the importance of the position-to-content term as well, which ensures that attention weights reflect not just the content but also the relative positions of word pairs.

For single-head attention, the disentangled self-attention mechanism with relative position bias is formulated as:

$$Q = HW_q, \quad K = HW_k, \quad V = HW_v, \quad A = \frac{QK^T}{\sqrt{d}} \tag{3.2}$$

$$H_o = \text{softmax}(A)V \tag{3.3}$$

Here, H represents the input hidden vectors, H_o the output of self-attention, and W_q, W_k, W_v are projection matrices. The relative distance $\delta(i, j)$ between tokens i and j is defined and used to compute position embeddings.

DeBERTa incorporates relative position embeddings into the self-attention mechanism by projecting content vectors and position vectors separately:

$$Q_c = HW_{q,c}, \quad K_c = HW_{k,c}, \quad V_c = HW_{v,c}, \quad Q_r = PW_{q,r}, \quad K_r = PW_{k,r} \tag{3.4}$$

The attention matrix is then:

$$\tilde{A}_{i,j} = Q_{c,i}K^T_{c,j} + Q_{c,i}K^T_{r,\delta(i,j)} + K_{c,j}Q^T_{r,\delta(j,i)} \tag{3.5}$$

This formulation ensures that attention scores incorporate both content and positional information accurately. A scaling factor $\frac{1}{\sqrt{3d}}$ is applied to stabilize training, especially for large-scale pre-trained language models (PLM).

By setting the maximum relative distance k to 512 during pre-training, DeBERTa efficiently computes disentangled attention weights, reducing the space complexity to $O(kd)$ for storing relative position embeddings. This method leverages the reuse of relative position embeddings across all layers, significantly optimizing memory usage and computational efficiency.

3.5.2 Enhanced Mask Decoder

DeBERTa utilizes a MLM approach similar to BERT, where the model predicts masked words based on the surrounding context. However, DeBERTa enhances this process by incorporating absolute position embeddings in the softmax layer, which decodes masked words using aggregated contextual embeddings of word contents and positions. By accounting for absolute positions, DeBERTa more accurately predicts masked tokens. The disentangled attention mechanism already considers the contents and relative positions of the context words, but not their absolute positions, which can be crucial for accurate predictions.

Consider the sentence ”a new store opened beside the new mall” with the words ”store” and ”mall” masked for prediction. Using only the local context (i.e., relative positions and surrounding words) is insufficient for the model to distinguish between ”store” and ”mall” since both follow the word ”new” with the same relative positions. To address this limitation, the model needs to consider absolute positions as complementary information to the relative positions. For instance, the subject of the sentence is ”store,” not ”mall.” These syntactical nuances largely depend on the words’ absolute positions in the sentence.

There are two methods for incorporating absolute positions. The BERT model incorporates absolute positions in the input layer. In contrast, DeBERTa incorporates them right after all the Transformer layers but before the softmax layer for masked token prediction. This approach allows DeBERTa to capture relative positions throughout the Transformer layers and use absolute positions as complementary information when decoding

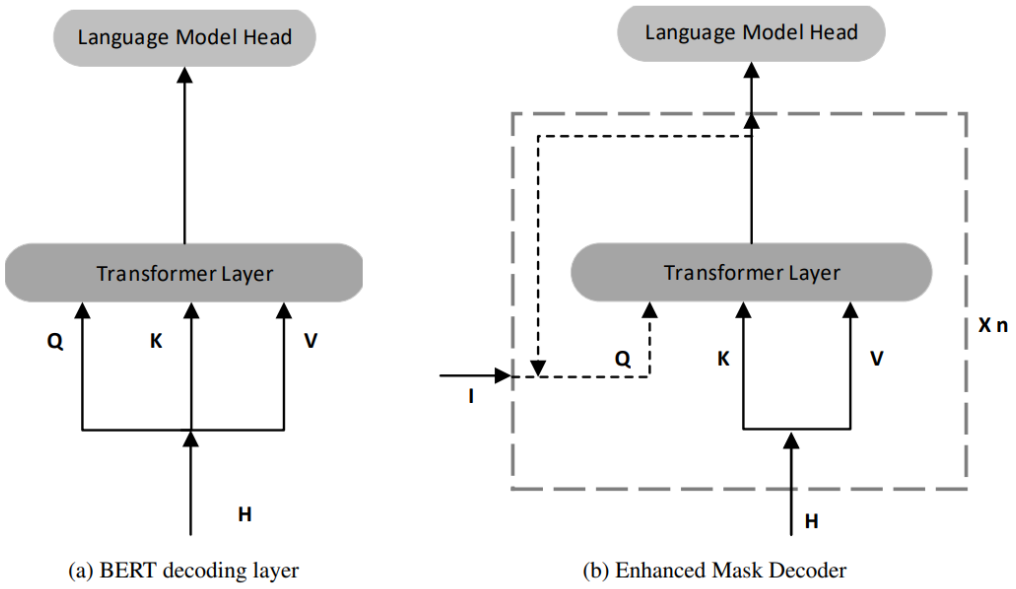


Figure 3.6: Absolute Position Incorporation

the masked words. Thus, DeBERTa's decoding component is termed an Enhanced Mask Decoder (EMD).

3.5.3 Scale Invariant fine-tuning

Virtual adversarial training is a regularization method aimed at improving a model's generalization by enhancing its robustness to adversarial examples. These examples are generated by making small perturbations to the input, and the model is regularized to produce the same output distribution for both the original and the perturbed examples.

For NLP tasks, the perturbation is applied to the word embedding rather than the original word sequence. However, the norms of the embedding vectors can vary significantly among different words and models. This variance becomes larger for models with billions of parameters, leading to instability in adversarial training, so DeBERTa utilizes the SiFT algorithm, which enhances training stability by applying perturbations to the normalized word embeddings. Specifically, when fine-tuning DeBERTa for a downstream NLP task, SiFT first normalizes the word embedding vectors into stochastic vectors and then applies the perturbation to the normalized embedding vectors.

3.5.4 Experiments and Results

DeBERTa was evaluated on several NLU tasks, comparing its performance with existing models like BERT, RoBERTa [84] , and XLNet. The models were tested using both large and base configurations.

$\text{DeBERTa}_{\text{large}}$ outperformed other models in various tasks:

- **GLUE Benchmark:** DeBERTa achieved an average score of 90.00, surpassing models like RoBERTa and XLNet.

- **MNLI:** Achieved 91.1% accuracy, setting a new state-of-the-art.
- **SQuAD v2.0:** Scored 90.7% (F1) and 88.0% (EM), outperforming RoBERTa and XLNet.
- **RACE:** Improved performance by 1.4% over XLNet, with an accuracy of 86.8%.

DeBERTa also showed superior performance in the base model setting:

- **MNLI-m:** Scored 88.8% accuracy, improving over RoBERTa and XLNet.
- **SQuAD v1.1:** Achieved 93.1% (F1) and 87.2% (EM).
- **SQuAD v2.0:** Achieved 86.2% (F1) and 83.1% (EM).

DeBERTa was scaled up to 1.5 billion parameters, achieving remarkable results in SuperGLUE:

- **Macro-average score:** 89.9, surpassing the human baseline of 89.8 for the first time.
- **Ensemble Model:** Achieved 90.3, topping the SuperGLUE leaderboard.

These results highlight DeBERTa's efficiency in pre-training and its superior performance across a wide range of NLP tasks compared to other state-of-the-art models.

3.5.5 Model Pre-training and Parameters

Several configurations of the DeBERTa model were pre-trained, including the base, large, and an even larger model with 1.5 billion parameters. These models vary in terms of architecture, with the base model consisting of 12 layers, the large model having 24 layers, and the 1.5B model incorporating 48 layers. Similarly, the hidden size increases from 768 in the base model to 1024 in the large model, and 1536 in the 1.5B model. The number of attention heads also scales with model size, from 12 in the base model to 16 in the large model, and 24 in the 1.5B model. All models were trained with the Adam optimizer, a batch size of 2048, and 1 million total steps, but the training duration varied from 10 days for the base model to 30 days for the 1.5B model, reflecting the increasing complexity and data size used in training. These varying configurations enable the models to capture different levels of linguistic complexity and representation depth. The details of these models are outlined in Table 3.1. For all models, the Adam optimizer was used with the following values: $\epsilon = 1 \times 10^{-6}$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$.

3.6 DeBERTaV3

The DeBERTaV3 [62] model represents a significant advancement in the field of PLM, building upon the foundation laid by its predecessor, DeBERTa. One of the primary enhancements in DeBERTaV3 is the replacement of the traditional MLM task with the more efficient replaced token detection (RTD) task, inspired by ELECTRA[52]. This shift addresses the limitations of MLM, offering a more sample-efficient pre-training approach that

Attribute	$\text{DeBERTa}_{\text{base}}$	DeBERTa	DeBERTa_1.5B
Number of Layers	12	24	48
Hidden Size	768	1024	1536
FNN Inner Hidden Size	3072	4096	6144
Attention Heads	12	16	24
Attention Head Size	64	64	64
Dropout	0.1	0.1	0.1
Learning Rate	2e-4	2e-4	1.5e-4
Warmup Steps	10k	10k	10k
Batch Size	2048	2048	2048
Weight Decay	0.01	0.01	0.01
Total Steps	1M	1M	1M
Data size	78GB	78GB	154GB
Gradient Clipping	1.0	1.0	1.0
Training Duration	10 days	20 days	30 days

Table 3.1: Specifications of DeBERTa models.

enhances both the training efficiency and the overall performance of the model. The core innovation lies in the gradient-disentangled embedding sharing (GDES) method, which resolves the ”tug-of-war” dynamics observed in ELECTRA’s vanilla embedding sharing. This method stops the gradients from the discriminator from affecting the generator’s embeddings, thus improving the balance between embedding sharing and training efficiency.

The DeBERTaV3 model performs exceptionally well on various NLU tasks, setting new records. When trained with the same setup as the older DeBERTa model, DeBERTaV3 shows better results, especially on the GLUE benchmark, where the $\text{DeBERTaV3}_{\text{large}}$ version scores an impressive 91.37%. This score is higher than both DeBERTa and ELECTRA models. Furthermore, the new multilingual version, mDeBERTaV3, performs much better in understanding different languages, as seen in the XNLI benchmark. These improvements highlight the model’s efficiency and effectiveness, making it a more powerful tool for language-related tasks.

The authors of DeBERTaV3 also provide two scripts at `https://github.com/microsoft/DeBERTa/blob/master/experiments/language_model` that allow training models of various configurations for Replaced Token Detection (RTD) and MLM. We will use these scripts to train our models as well.

3.6.1 Motivation and Historical Context

The evolution of PLM has been marked by continuous improvements in efficiency and effectiveness. BERT’s introduction of the MLM task marked a significant milestone, but its limitations in training efficiency led to the development of ELECTRA’s RTD approach. DeBERTaV3 builds on these advancements, aiming to address the inefficiencies and performance bottlenecks observed in previous models.

3.6.1.1 ELECTRA

ELECTRA is a novel pre-training method that improves upon the standard MLM approach by using a more sample-efficient task called Replaced Token Detection RTD. Traditional models like BERT rely on MLM to predict masked tokens, which can be computationally expensive and inefficient. ELECTRA introduces a discriminator-generator framework, similar to GANs, where a generator replaces some tokens in a sentence, and the discriminator's role is to identify which tokens have been replaced. This approach enables ELECTRA to learn from every token in the input sequence rather than just the masked ones, significantly improving the model's efficiency and downstream task performance.

3.6.1.1.1 Masked Language Model

Large-scale Transformer-based PLMs are typically trained on vast amounts of text to learn contextual word representations using MLM [45]. Specifically, given a sequence $X = \{x_i\}$, it is corrupted into $\tilde{X}$ by randomly masking 15% of its tokens. A language model parameterized by θ is then trained to reconstruct X by predicting the masked tokens $\tilde{x}$ conditioned on $\tilde{X}$:

$$\max_{\theta} \log p_\theta(X|\tilde{X}) = \max_{\theta} \sum_{i \in C} \log p_\theta(\tilde{x}_i = x_i|\tilde{X})$$

where C is the index set of the masked tokens in the sequence. The authors of BERT propose to keep 10% of the masked tokens unchanged, another 10% replaced with randomly picked tokens, and the rest replaced with the [MASK] token.

3.6.1.1.2 Replaced Token Detection

Unlike BERT, which uses a single transformer encoder and is trained with MLM, ELECTRA is trained with two transformer encoders in a GAN-style setup. One encoder, called the generator, is trained with MLM, while the other, called the discriminator, is trained with a token-level binary classifier. The generator produces ambiguous tokens to replace masked tokens in the input sequence. The modified input sequence is then fed to the discriminator, which uses its binary classifier to determine if each token is original or replaced by the generator. We denote the parameters of the generator and discriminator as θ_G and θ_D, respectively.

The training objective for the discriminator is RTD. The generator's loss function is written as:

$$L_{MLM} = \mathbb{E}\left[-\sum_{i \in C} \log p_{\theta_G}(\tilde{x}_i = x_i|\tilde{X}_G)\right]$$

where $\tilde{X}_G$ is the input to the generator with 15% of the tokens in X randomly masked. The input sequence for the discriminator is constructed by replacing masked tokens with new tokens sampled according to the output probability from the generator:

$$\tilde{x}_{i,D} = \begin{cases} \tilde{x}_i \sim p_{\theta_G}(\tilde{x}_i = x_i | \tilde{X}_G) & \text{if } i \in C \\ x_i & \text{if } i \notin C \end{cases}$$

The loss function of the discriminator is written as:

$$L_{RTD} = \mathbb{E}\left[-\sum_i \log p_{\theta_D}(1(\tilde{x}_{i,D} = x_i) | \tilde{X}_D, i) \right]$$

where $1(\cdot)$ is the indicator function and $\tilde{X}_D$ is the input to the discriminator constructed via the previous equation. In ELECTRA, L_{MLM} and L_{RTD} are optimized jointly:

$$L = L_{MLM} + \lambda L_{RTD}$$

where λ is the weight of the discriminator loss L_{RTD}, set to 50 in ELECTRA.

3.6.2 Implementation and Evaluation

DeBERTaV3 improves upon its predecessor, DeBERTa, by incorporating the RTD objective instead of the traditional MLM objective. This change leverages the sample efficiency of RTD as demonstrated in the ELECTRA model.

In the implementation of DeBERTaV3, Wikipedia and the BookCorpus are used as training data. The model configuration follows the base model setup of BERT, where the generator is designed to have the same width as the discriminator but only half its depth. The training setup includes a batch size of 2048 and a training duration of 125,000 steps with a learning rate of 5e-4 and 10,000 warmup steps. Consistent with Clark et al. (2020), a hyperparameter $\lambda = 50$ is utilized.

Further enhancements in DeBERTaV3 are achieved by replacing the token Embedding Sharing (ES) mechanism used for RTD with a new GDES method. This method addresses the multitask learning problem observed in the ES approach by preventing the gradients from the discriminator from interfering with the generator's embeddings. This innovation ensures that the training is more efficient and that the learned embeddings are of higher quality.

The GDES method involves re-parameterizing the discriminator embeddings as $E_D = \text{sg}(E_G) + E_\Delta$, where the stop gradient operator sg prevents the generator embeddings E_G from being updated by the discriminator's loss. Instead, only the residual embeddings E_Δ are updated. This approach combines the advantages of both ES and NES (No Embedding Sharing) while mitigating their drawbacks. Extensive experiments have demonstrated that GDES not only converges faster than ES but also produces embeddings that retain more semantic information and achieve superior performance on downstream tasks.

DeBERTaV3 validates its effectiveness through evaluations on two notable NLU tasks: MNLI and SQuAD v2.0. The results indicate significant improvements over the original DeBERTa, with an increase of 2.5% in MNLI-m accuracy and 3.8% in SQuAD v2.0 F1 score.

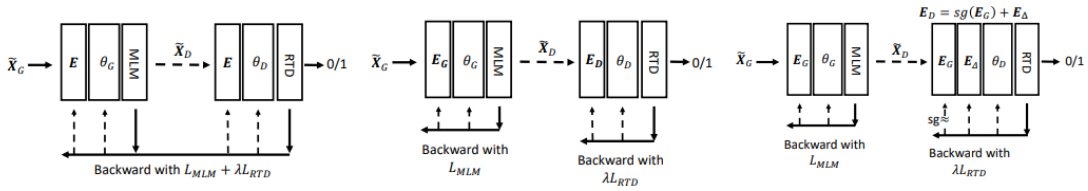


Figure 3.7: Gradient-Disentangled Embedding Sharing (GDES) method illustration.

3.6.3 Model pre-training and parameters

There are several models in the DeBERTaV3 family, each designed with different hyper-parameters to suit a range of NLP tasks. The key differences between the models, such as DeBERTaV3$_{\text{large}}$, DeBERTaV3$_{\text{base}}$, and DeBERTaV3$_{\text{xsmall}}$, lie in the number of layers, hidden size, and the feed-forward network inner hidden size. For instance, DeBERTaV3$_{\text{large}}$ is configured with 24 layers and a hidden size of 1024, making it suitable for more complex tasks requiring deeper representations, while DeBERTaV3$_{\text{base}}$ has 12 layers and a hidden size of 768, balancing performance and efficiency. DeBERTaV3$_{\text{xsmall}}$ is a smaller variant with 12 layers, 6 heads and a hidden size of 384 designed for tasks with lower computational resources while maintaining competitive performance.

Additionally, the models share common optimization parameters such as a batch size of 8k, dropout of 0.1, and learning rate warmup steps of 10k. However, the learning rate differs across models, with DeBERTaV3$_{\text{large}}$ using 3e-4, while DeBERTaV3$_{\text{base}}$ and DeBERTaV3$_{\text{xsmall}}$ both use 6e-4. The models are pre-trained for 500k steps with a linear learning rate decay. As shown in Table 3.2, these variations in hyper-parameters allow the DeBERTaV3 models to perform effectively across a wide range of natural language understanding tasks. For all models, the Adam optimizer was used with the following values: $\epsilon = 1 \times 10^{-6}$, $\beta_1 = 0.9$, and $\beta_2 = 0.98$.

Attribute	**DeBERTaV3$_{\text{base}}$**	**DeBERTaV3$_{\text{large}}$**	**DeBERTaV3$_{\text{xsmall}}$**
Number of Layers	12	24	12
Number of Heads	12	12	6
Hidden Size	768	1024	384
FNN Inner Hidden Size	3072	4096	3072
Attention Heads	12	12	12
Attention Head Size	64	64	64
Dropout	0.1	0.1	0.1
Warmup Steps	10k	10k	10k
Learning Rate	6e-4	3e-4	6e-4
Batch Size	8k	8k	8k
Weight Decay	0.01	0.01	0.01
Total Steps	500k	500k	500k
Data size	160GB	160GB	160GB
Gradient Clipping	1.0	1.0	1.0

Table 3.2: Specifications of DeBERTaV3 models.

3.7 Conclusion

In this chapter, we reviewed several key advancements in the development of transformer-based language models, focusing on BERT, GREEK-BERT, DeBERTa, and DeBERTaV3. BERT's introduction marked a significant shift in NLP by utilizing bidirectional training to capture context from both directions. GREEK-BERT extended this approach to the Greek language, addressing specific linguistic challenges and outperforming multilingual models in Greek NLP tasks. DeBERTa introduced novel techniques such as the disentangled attention mechanism and an enhanced mask decoder, further improving the performance of transformer models. DeBERTaV3 built upon these innovations by adopting the RTD task from ELECTRA and introducing the GDES method, achieving state-of-the-art results on various NLP benchmarks.

These advancements collectively highlight the continuous evolution and refinement of transformer models, each iteration addressing the limitations of its predecessors and contributing to improved language understanding and generation capabilities. As we move forward, these foundational models and techniques provide a solid framework for further innovations in NLP.

In the next chapter, we will delve into our work on developing GreekDeBERTa, a model that integrates the strengths of DeBERTa with specific adaptations for the Greek language, aiming to push the boundaries of Greek NLP even further.

4. GREEKDEBERTA

After discussing the foundational concepts and exploring advancements in NLP models, it is time to introduce our model, GreekDeBERTa. This model builds on the powerful DeBERTa architecture, tailored specifically for the Greek language. With enhancements designed to handle the unique aspects of Greek, GreekDeBERTa aims to improve performance in Greek NLP tasks.

4.1 Server Resources

For all program executions Amazon Web Services (AWS) was used. We employed three instances of the AWS `g5.16xlarge` servers, each located in different regions: Frankfurt, Ireland, and London. The specifications of the `g5.16xlarge` instances are shown in Table 4.1. The `g5.16xlarge` instance is equipped with a single GPU, providing 24 GiB

Specification	Details
Name	AWS g5.16xlarge
GPU	1
GPU Memory	24 GiB
vCPUs	64
Memory	256 GiB
Storage	1x1900 GB
Network Bandwidth	25 Gbps
EBS Bandwidth	16 Gbps
On Demand Price/hr	$4.096

Table 4.1: AWS `g5.16xlarge` Server Specifications

of GPU memory, and is well-suited for compute-intensive tasks. It includes 64 vCPUs and 256 GiB of memory, thereby offering considerable computational power. Additionally, the network bandwidth can reach up to 25 Gbps, and the EBS bandwidth is 16 Gbps, ensuring efficient data transfer and storage performance.

The operating system was Linux and the main programming language was Python with some bash scripting to automate some processes. The server was also used to store the data and pre-process it. Many different libraries and frameworks were used, such as PyTorch, Sklearn, numpy, nltk and transformers.

4.2 Data

The dataset used for pre-training GreekDeBERTa was the same as that utilized for GREEK-BERT and DistilGREEK-BERT [48][50]:

- **Greek Wikipedia:** This dataset includes 1GB of data spread across 1,018 files. It consists of various articles and entries, providing a rich source of general knowledge in Greek.

- **European Parliament Proceedings Parallel Corpus (Europarl):** The Greek portion of the Europarl corpus encompasses 0.45GB and includes 9,271 files. This resource consists of transcriptions and translations of European Parliament sessions, offering valuable parliamentary language data.
- **Greek OSCAR Dataset:** Derived from a cleaned version of the Common Crawl, this dataset provides a substantial 27.0GB of deduplicated Greek text across 15 files. It covers a wide range of topics and serves as a broad representation of the Greek language on the internet.

4.2.1 Cleaning and preprocessing

Following the partitioning of the dataset into training, validation, and test subsets with respective proportions of 0.90, 0.05, and 0.05, we proceed to the essential stages of data cleaning and preprocessing:

1. **Text Lowercasing**: Convert all text to lowercase. This standardizes the text, removing case sensitivity and ensuring uniformity in text processing.
2. **Accent Removal**: Remove diacritical marks (accents) from characters. This simplifies text and reduces variations, making analysis more straightforward.
3. **Punctuation Normalization**: Standardize punctuation by collapsing repeated marks into a single instance. This helps in maintaining consistent punctuation usage across the dataset.
4. **Removal of URLs and Special Symbols**: Eliminate URLs, special symbols, dates, and times. This step removes extraneous information that could distract from the core text content.
5. **Single-Character Word Removal**: Remove single-character words, except specific exceptions. This reduces noise and irrelevant data.
6. **Target Language Filtering**: Retain sentences containing target language characters. This focuses the dataset on the relevant linguistic content.
7. **Numeric Sentence Filtering**: Discard sentences composed solely of numbers. This excludes data that is likely not useful for text analysis.
8. **Short Sentence Filtering**: Exclude sentences with fewer than three words. This removes potentially uninformative content, enhancing dataset quality.
9. **Encoding Verification**: Check for non-UTF-8 characters and address any discrepancies. This ensures consistent text encoding, crucial for accurate data processing and analysis.

4.2.2 Vocabulary Creation

The DeBERTa model configuration requires a specific type of vocabulary that is compatible with SentencePiece tokenization. The vocabulary size was set to 50,000 tokens to cover a broad spectrum of Greek words and subwords, enabling the model to effectively handle

various linguistic nuances, including morphological variations and inflectional forms. This vocabulary is structured to be compatible with BERT-style tokenization, where tokens can represent whole words or subword units. Such a structure allows the model to manage both common and rare words efficiently, improving its ability to generalize across different contexts and domains.

Some of the tokens generated from the trained vocabulary are presented in the image below.

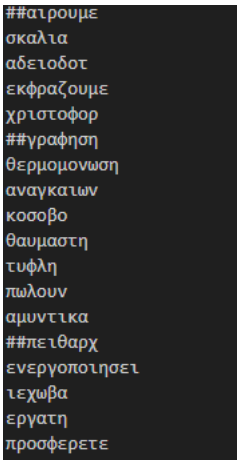


Figure 4.1: Sample tokens from the trained vocabulary.

4.3 Models Pre-training

To comprehensively evaluate the performance and efficiency of different DeBERTa models architectures on Greek language tasks, we trained multiple model variants with distinct hyperparameters. These included a GreekDeBERTa$_{base}$ model, a GreekDeBERTaV3$_{xsmall}$ model and a GreekDeBERTaV3$_{base}$ model with the default configurations as discussed in chapter 3 and shown in Table 4.2

Attribute	**GreekDeBERTa$_{base}$**	**GreekDeBERTaV3$_{xsmall}$**	**GreekDeBERTaV3$_{base}$**
Number of Layers	12	12	12
Hidden Size	768	384	768
FFN Inner Hidden Size	3072	3072	3072
Attention Heads	12	6	12
Attention Head Size	64	64	64
Dropout	0.1	0.1	0.1
Learning Rate	2e-4	6e-4	6e-4
Weight Decay	0.01	0.01	0.01
Gradient Clipping	1.0	1.0	1.0

Table 4.2: Specifications of GreekDeBERTa and GreekDeBERTaV3 models

All the models were trained for 10 epochs and with warmup steps equal to 1% of the total training steps. Batch size was 256 with accumulation update since the GPU could only treat batches of size 8 or 16 based on the model.

The training durations and total parameter counts for the different GreekDeBERTa model variants were :

- **GreekDeBERTa$_{\text{base}}$:**
  - **Total Parameters:** 124,962,996
  - **Training Duration:** 21 days
- **GreekDeBERTaV3$_{\text{xsmall}}$:**
  - **Total Parameters:** 71,554,357
  - **Training Duration:** 16 days
- **GreekDeBERTaV3$_{\text{base}}$:**
  - **Total Parameters:** 207,349,429
  - **Training Duration:** 30 days

These training periods and parameter sizes underscore the trade-offs between model complexity, resource requirements, and training time, guiding the selection of an appropriate model for various practical applications.

The selection of the GreekDeBERTa models variants for this study was driven by the need to balance model complexity with resource efficiency while maintaining competitive performance in Greek language tasks. The **GreekDeBERTa$_{\text{base}}$** model was chosen due to its strong baseline architecture, which has been widely adopted for NLP tasks, making it an appropriate starting point for adapting DeBERTa to the Greek language.

The **GreekDeBERTaV3$_{\text{xsmall}}$** model was selected to explore a more lightweight architecture. With fewer parameters and a smaller number of layers, this model offers a trade-off between computational resource demands and training speed, making it suitable for scenarios where limited hardware resources are available.

Finally, the **GreekDeBERTaV3$_{\text{base}}$** model, which introduces improvements from the DeBERTa V3 architecture, was chosen to assess the performance gains that could be achieved by utilizing enhanced pre-training techniques.

For all models, pre-training was evaluated on the validation set at each 1% increment of the total steps. The loss curves for MLM and RTD pre-training for GreekDeBERTa and GreekDeBERTaV3 accordingly are presented in the figures below. Also there is a table with the starting accuracy-loss and the final ones. The pre-training progress of the GreekDeBERTa models is illustrated through the accuracy and loss curves across multiple iterations. Both the MLM and RTD objectives exhibit significant improvements early in training, with the curves showing steep increases in accuracy and sharp decreases in loss within the first 10% of the total training steps. As training progresses, the improvements become more gradual, indicating a typical learning curve where most of the model's learning occurs in the early stages. For the GreekDeBERTa$_{\text{base}}$ and GreekDeBERTaV3 models, both accuracy and loss stabilize towards the later iterations, suggesting convergence. This behavior is consistent across all models, highlighting efficient learning and successful optimization during the pre-training phase.

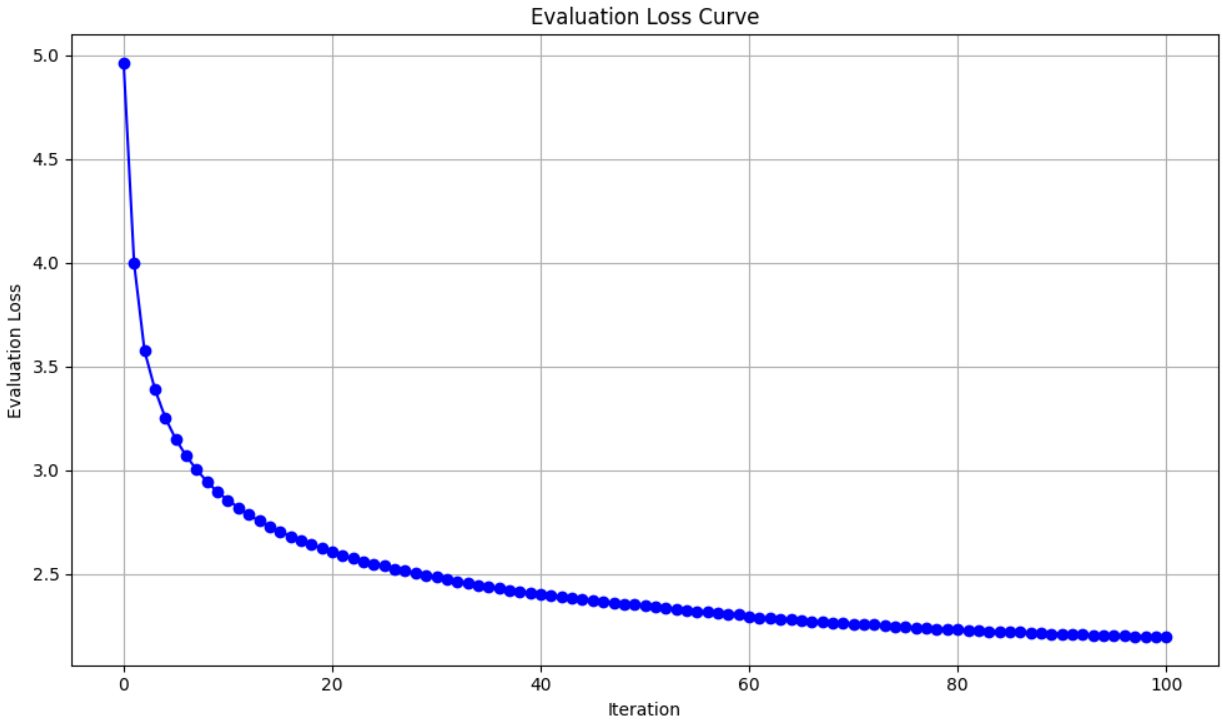


Figure 4.2: GreekDeBERTa$_{base}$ loss curve

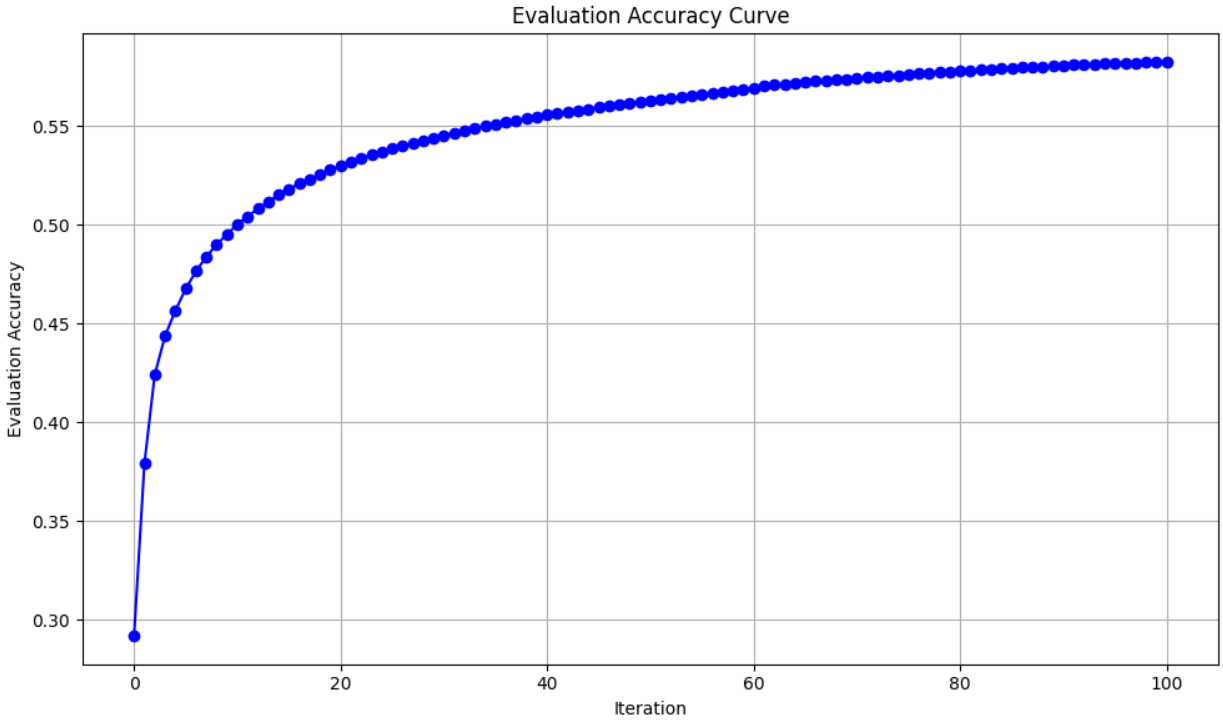


Figure 4.3: GreekDeBERTa$_{base}$ accuracy curve

Model	Starting Accuracy	Starting Loss	Final Accuracy	Final Loss
GreekDeBERTa$_{base}$	0.2921	4.9622	0.5824	2.1989
GreekDeBERTaV3$_{xsmall}$	0.3596	3.8594	0.6407	1.7159
GreekDeBERTaV3$_{base}$	0.3223	4.2907	0.6796	1.4774

Table 4.3: Performance metrics for different GreekDeBERTa models.

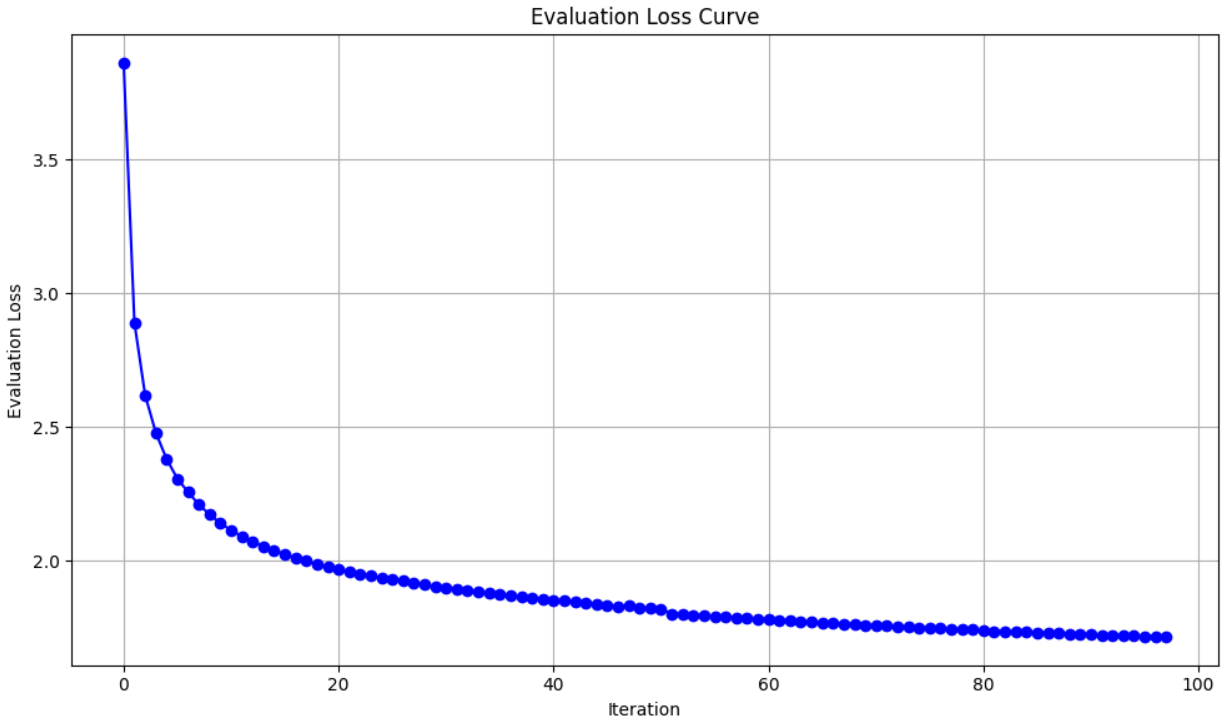


Figure 4.4: GreekDeBERTaV3$_{xsmall}$ loss curve

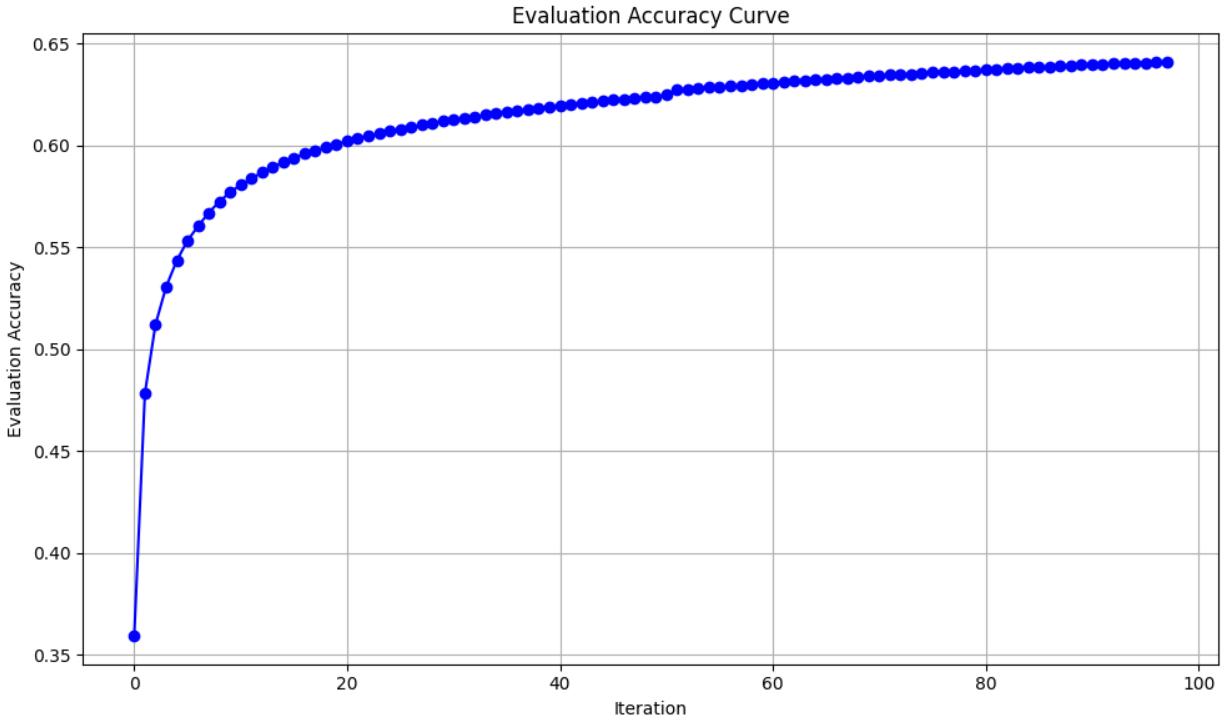


Figure 4.5: GreekDeBERTaV3$_{xsmall}$ accuracy curve

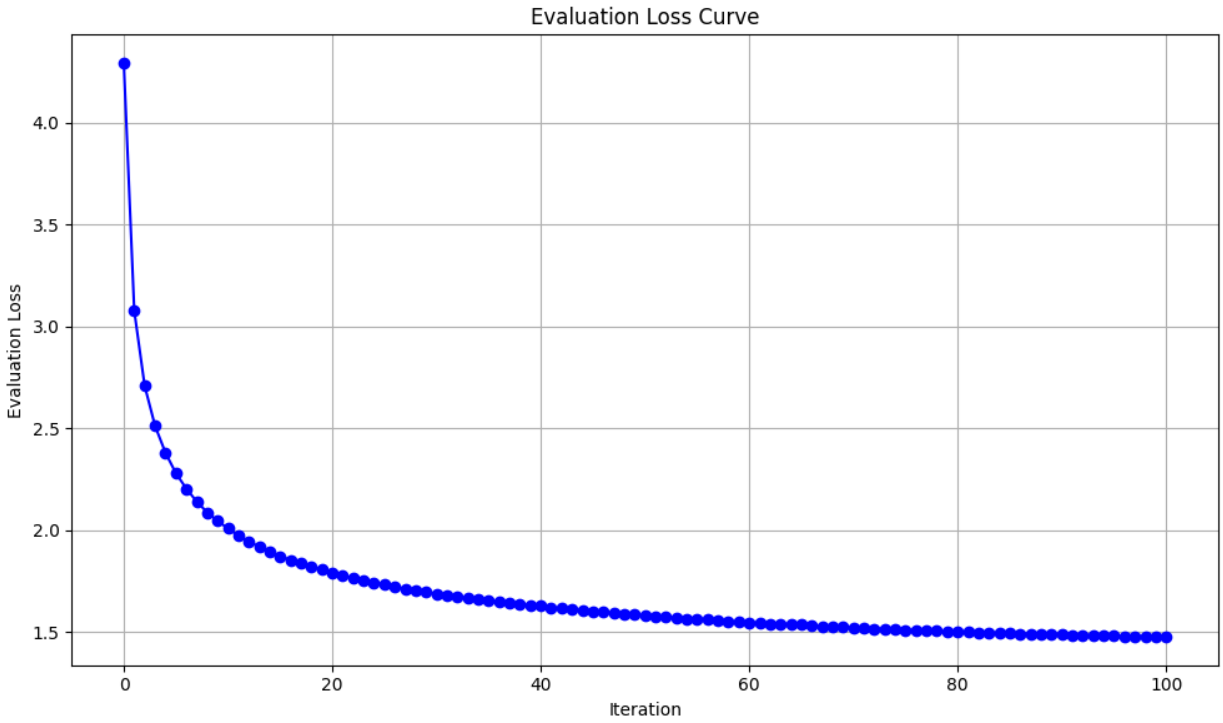


Figure 4.6: GreekDeBERTaV3$_{base}$ loss curve

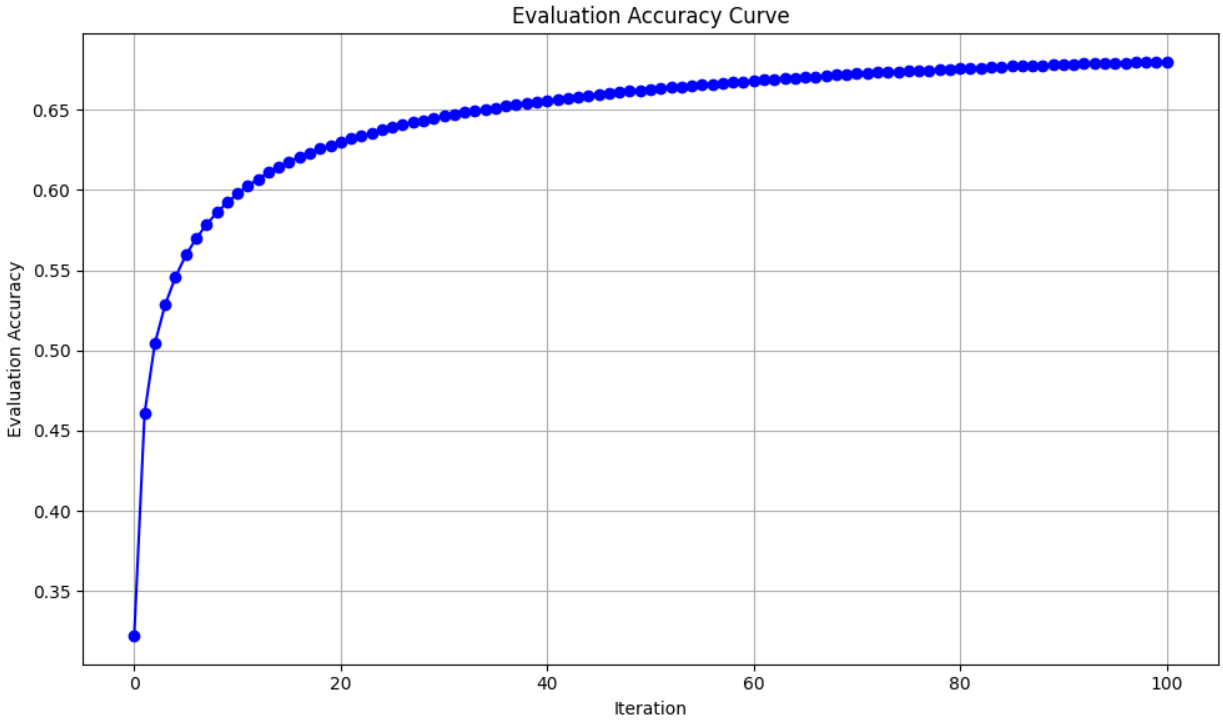


Figure 4.7: GreekDeBERTaV3$_{base}$ accuracy curve

5. EVALUATION TASKS

After pre-training our models, it's time to evaluate their performance on some downstream tasks. The tasks and datasets are the same with those of GREEK-BERT so we can compare the results. These tasks are :

- Part-Of-Speech Tagging (PoS Tagging)
- Named Entity Recognition (NER)
- Natural Language Inference (NLI)

For fine-tuning the models, we used scripts that loaded the tokenizers and model configurations from the DeBERTa-v2 family, as provided by Hugging Face [56]. We used DeBERTa-v2 for the original MLM model because there was a shift from version 1 to version 2, where the authors replaced the GPT-2 tokenizer with a SentencePiece tokenizer. Since we also used an SPM model for the creation of the vocabulary, it was necessary to use the DeBERTa-v2 model. Additionally, for DeBERTa-v3, the authors did not provide any new or separate classes, as the architecture and tokenizers remained unchanged from DeBERTa-v2.

During pre-training of DeBERTa-v3, two models were created: a generator and a discriminator, following the RTD framework. For fine-tuning tasks such as PoS tagging, NER, and NLI, we chose to use the discriminator instead of the generator. This is because the discriminator is specifically trained to identify and distinguish between original and replaced tokens, making it more effective for tasks that require precise token-level and sentence-level understanding, such as classification in NLI or token identification in PoS tagging and NER

For the fine-tuning, Trainer class and AdamW optimizer was used from hugging face [25] [24]. For loading the datasets, the datasets library was used [23]. The preprocessing of the datasets consisted of two steps : lowercasing and removing accents. For all datasets, padding was done to the longest sequence found in each dataset. For the evaluation of the tasks, F1 Score, Precision and Recall from Sklearn was used [71].

Also, GREEK-BERT and DistilGREEK-BERT were evaluated on the same downstream tasks for comparison purposes, using the default hyperparameters provided by the authors of each model [48] [50]. Both models were tested using their original settings without any additional fine-tuning or hyperparameter optimization, to ensure a fair comparison with our DeBERTa-based models. In addition to accuracy, the time required for fine-tuning and inference was also measured for all models, including the time needed for both training and evaluation on each task. Time efficiency is a crucial factor in real-world applications, as it provides insights into the computational performance of the models. By considering both accuracy and time, we ensure a comprehensive comparison between our GreekDeBERTa models and the baseline models like GREEK-BERT and DistilGREEK-BERT, allowing us to assess the trade-offs between performance and computational efficiency.

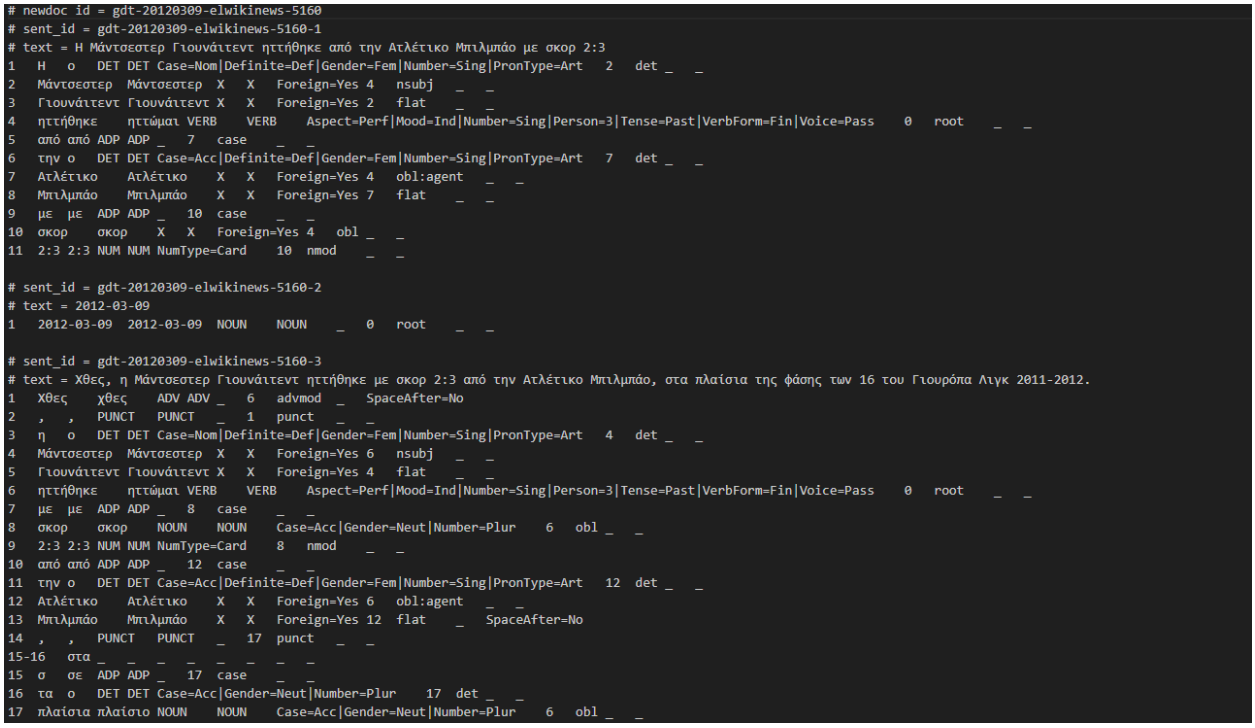
5.1 PoS Tagging

POS tagging, or Part-of-Speech tagging, is a task in NLP that involves marking up words in a text as corresponding to a particular part of speech, based on both their definition and

context. It helps in understanding the grammatical structure of sentences and is used in various NLP applications, such as speech recognition, machine translation, and information retrieval. By tagging each word with its appropriate part of speech tag, such as noun, verb, adjective, etc., POS tagging helps the extraction of semantic meaning and syntactic relationships from text. POS tagging is important because it aids in disambiguating words that may have multiple meanings, thus improving the accuracy and performance of NLP systems[57].

5.1.1 Dataset

The Greek Universal Dependencies Treebank (GUDT)[63], a dataset sourced from the Greek Dependency Treebank [77], was utilized for this task. This treebank, created and maintained by the Institute for Language and Speech Processing at the Research Center 'Athena'[13], contains a total of 2,521 sentences. These sentences are divided into three sets: 1,622 for training, 403 for evaluation, and 456 for testing. The dataset includes 16 available tags. In accordance with GREEK-BERT, syntactic dependencies were not considered in this study. The code related to this dataset can be found here[79]. In the figure below is an example of the dataset. For easier manipulation of the dataset, CoNLL-



```
# newdoc id = gdt-20120309-elwikinews-5160
# sent_id = gdt-20120309-elwikinews-5160-1
# text = Η Μάντσεστερ Γιουνάιτεντ ηττήθηκε από την Ατλέτικο Μπιλμπάο με σκορ 2:3
1   Η   ο   DET DET Case=Nom|Definite=Def|Gender=Fem|Number=Sing|PronType=Art   2   det _   _
2   Μάντσεστερ  Μάντσεστερ  X   X   Foreign=Yes 4   nsubj   _   _
3   Γιουνάιτεντ Γιουνάιτεντ X   X   Foreign=Yes 2   flat    _   _
4   ηττήθηκε    ηττώμαι VERB    VERB    Aspect=Perf|Mood=Ind|Number=Sing|Person=3|Tense=Past|VerbForm=Fin|Voice=Pass    0   root    _   _
5   από από ADP ADP _   7   case    _   _
6   την ο   DET DET Case=Acc|Definite=Def|Gender=Fem|Number=Sing|PronType=Art   7   det _   _
7   Ατλέτικο    Ατλέτικο    X   X   Foreign=Yes 4   obl:agent   _   _
8   Μπιλμπάο    Μπιλμπάο    X   X   Foreign=Yes 7   flat    _   _
9   με  με  ADP ADP _   10  case    _   _
10  σκορ    σκορ    X   X   Foreign=Yes 4   obl _   _
11  2:3 2:3 NUM NUM NumType=Card    10  nmod    _   _

# sent_id = gdt-20120309-elwikinews-5160-2
# text = 2012-03-09
1   2012-03-09  2012-03-09  NOUN    NOUN    _   0   root    _   _

# sent_id = gdt-20120309-elwikinews-5160-3
# text = Χθες, η Μάντσεστερ Γιουνάιτεντ ηττήθηκε με σκορ 2:3 από την Ατλέτικο Μπιλμπάο, στα πλαίσια της φάσης των 16 του Γιουρόπα Λιγκ 2011-2012.
1   Χθες    χθες    ADV ADV _   6   advmod  _   SpaceAfter=No
2   ,   ,   PUNCT   PUNCT   _   1   punct   _   _
3   η   ο   DET DET Case=Nom|Definite=Def|Gender=Fem|Number=Sing|PronType=Art   4   det _   _
4   Μάντσεστερ  Μάντσεστερ  X   X   Foreign=Yes 6   nsubj   _   _
5   Γιουνάιτεντ Γιουνάιτεντ X   X   Foreign=Yes 4   flat    _   _
6   ηττήθηκε    ηττώμαι VERB    VERB    Aspect=Perf|Mood=Ind|Number=Sing|Person=3|Tense=Past|VerbForm=Fin|Voice=Pass    0   root    _   _
7   με  με  ADP ADP _   8   case    _   _
8   σκορ    σκορ    NOUN    NOUN    Case=Acc|Gender=Neut|Number=Plur    6   obl _   _
9   2:3 2:3 NUM NUM NumType=Card    8   nmod    _   _
10  από από ADP ADP _   12  case    _   _
11  την ο   DET DET Case=Acc|Definite=Def|Gender=Fem|Number=Sing|PronType=Art   12  det _   _
12  Ατλέτικο    Ατλέτικο    X   X   Foreign=Yes 6   obl:agent   _   _
13  Μπιλμπάο    Μπιλμπάο    X   X   Foreign=Yes 12  flat    _   SpaceAfter=No
14  ,   ,   PUNCT   PUNCT   _   17  punct   _   _
15-16   στα _   _   _   _   _   _   _
15  σ   σε  ADP ADP _   17  case    _   _
16  τα  ο   DET DET Case=Acc|Gender=Neut|Number=Plur    17  det _   _
17  πλαίσια πλαίσιο NOUN    NOUN    Case=Acc|Gender=Neut|Number=Plur    6   obl _   _
```

Figure 5.1: PoS dataset example

U Parser was used [64].

5.1.2 Hyperparameters and Results

The hyperparameters for all the models are shown in the Table 5.1. These hyperparameters were set after several experiments done. The **Learning Rate** is set to a low value of 1.5e-5 for all models to ensure stable convergence during training. This value was chosen after experimenting with higher rates, which led to unstable training and lower performance. A consistent **Batch Size** of 16 is used across all models. Higher or lower

Hyperparameter	GreekDeBERTa$_{base}$	GreekDeBERTaV3$_{xsmall}$	GreekDeBERTaV3$_{base}$
Learning Rate	1.5e-5	1.5e-5	1.5e-5
Batch Size	16	16	16
Epochs	7	6	7
Weight Decay	0	0	0
Warmup Steps	10% of total steps	10% of total steps	10% of total steps
Scheduler Type	Cosine	Linear	Cosine

Table 5.1: Hyperparameters for the GreekDeBERTa models for PoS

values did not yield better results. The number of **Epochs** varies slightly among the models, with GreekDeBERTa$_{base}$ and GreekDeBERTaV3$_{base}$ RTD trained for 7 epochs, while GreekDeBERTaV3$_{xsmall}$ is trained for 6 epochs. This variation is due to the different convergence behaviors observed during training. No **Weight Decay** is applied in this setup, as initial tests showed that regularization via weight decay was not necessary for these models to generalize well. The **Warmup Steps** are set to 10% of the total steps for each model. This practice helps in gradually increasing the learning rate at the beginning of training, preventing large updates that could destabilize the training process. Different **Scheduler Types** are utilized: the GreekDeBERTa$_{base}$ and GreekDeBERTaV3$_{base}$ models use a cosine scheduler, which adjusts the learning rate in a smooth and decaying manner. In contrast, GreekDeBERTaV3$_{xsmall}$ employs a linear scheduler, which decreases the learning rate linearly over time. The **Max Sequence Length** is fixed at 128 tokens for all models.

The accuracy of the models is presented in the Table 5.2, f1 micro was used for this task.

Model	Accuracy (%)	Execution Time (mm:ss)
GREEK-BERT	98.1 $\pm$ 0.08	04:01
GreekDeBERTa$_{base}$	**98.4** $\pm$ 0.07	05:10
GreekDeBERTaV3$_{xsmall}$	98.3 $\pm$ 0.07	02:38
GreekDeBERTaV3$_{base}$	98.3 $\pm$ 0.06	05:45
DistilGREEK-BERT	97.0	01:25

Table 5.2: Performance of the GreekDeBERTa models on PoS task

The results of the PoS tagging task reveal that the DeBERTa models, developed during this research, outperform the previously best model, GREEK-BERT. Specifically, **GreekDeBERTa$_{base}$** achieves the highest accuracy of 98.4% with a standard deviation of 0.07, indicating a slight but consistent improvement over GREEK-BERT's 98.1% accuracy. Similarly, both **GreekDeBERTaV3$_{xsmall}$** and **GreekDeBERTaV3$_{base}$** achieve an accuracy of 98.3%, with standard deviations of 0.07 and 0.06, respectively. These results demonstrate the effectiveness of the DeBERTa models in handling PoS tagging tasks, showcasing improved accuracy and lower variability compared to GREEK-BERT.

In terms of execution time, the results show a trade-off between model performance and speed. While **GreekDeBERTa$_{base}$** delivers the highest accuracy at 98.4%, it also takes the longest time to execute at 5 minutes and 10 seconds. On the other hand, **GreekDeBERTaV3$_{xsmall}$** achieves a competitive accuracy of 98.3%, but with a significantly shorter execution time of 2 minutes and 38 seconds, making it a more efficient option for scenarios where time is a critical factor. **DistilGREEK-BERT** is the fastest, completing

the task in just 1 minute and 25 seconds, but this speed comes at the cost of reduced accuracy, with a score of 97.0%. Thus, there is a clear balance between execution time and model accuracy, where DeBERTa models, although slower than **DistilGREEK-BERT**, offer superior accuracy for PoS tagging task.

5.2 NER

Named Entity Recognition (NER) involves identifying and categorizing key information in text, such as names of people, organizations, locations, and dates. It's like having a smart assistant that can quickly sift through large amounts of text and pick out the important bits. NER is particularly useful in making sense of unstructured data, such as news articles, emails, or social media posts, by extracting relevant entities and classifying them into predefined categories. The evolution of NER has been driven by advancements in ML and deep learning. Initially, NER systems relied on rule-based methods, which involved manually crafted rules to identify entities. However, these systems were limited in their ability to generalize across different contexts and languages. With the advent of ML, NER models began to learn patterns from large datasets, improving their accuracy and flexibility. In practical applications, NER can help businesses automate tasks like information retrieval, data organization, and content filtering. For instance, a news aggregator can use NER to categorize articles by identifying key entities mentioned in the text, such as politicians, companies, or geographic locations. This automation not only saves time but also enhances the accuracy of information extraction, enabling more efficient data analysis and decision-making processes [38].

5.2.1 Dataset

The dataset used was sourced from the prior work on DistilGREEK-BERT[50]. The dataset had been pre-processed, with splits into training, evaluation, and test sets already completed. Standardization of the format and labeling of entities using the IOB2 scheme were also previously carried out. The dataset was received in a fully prepared state, requiring no additional modifications. This preparation ensured consistent entity annotations, such as the merging of "LOC" and "GPE" under the unified "GPE" label. The original dataset was developed by A. Romanou and I. Darras during student projects at NTUA and AUEB, respectively[19]. In the figure below is an example of the final dataset. The dataset consists of JSON files containing sentences, where each sentence is represented as a list of tokens and corresponding NER tags. Our data manipulation involves reading these JSON files, extracting the tokens and NER tags, and concatenating them into a unified format suitable for creating Hugging Face datasets.

5.2.2 Hyperparameters and Results

The hyperparameters for all the models are shown in the Table 5.3. The **Learning Rate** for GreekDeBERTa$_{\text{base}}$ and GreekDeBERTaV3$_{\text{base}}$ is set at a low value of 1×10^{-5}, while a higher rate of 1×10^{-4} is used for GreekDeBERTaV3$_{\text{xsmall}}$. A uniform **Batch Size** of 16 was employed across all models. The number of **Epochs** varies, with GreekDeBERTa$_{\text{base}}$ trained for 4 epochs, GreekDeBERTaV3$_{\text{xsmall}}$ for 6 epochs, and GreekDeBERTaV3$_{\text{base}}$ for 7 epochs. This variation accommodates the different convergence rates, allowing the

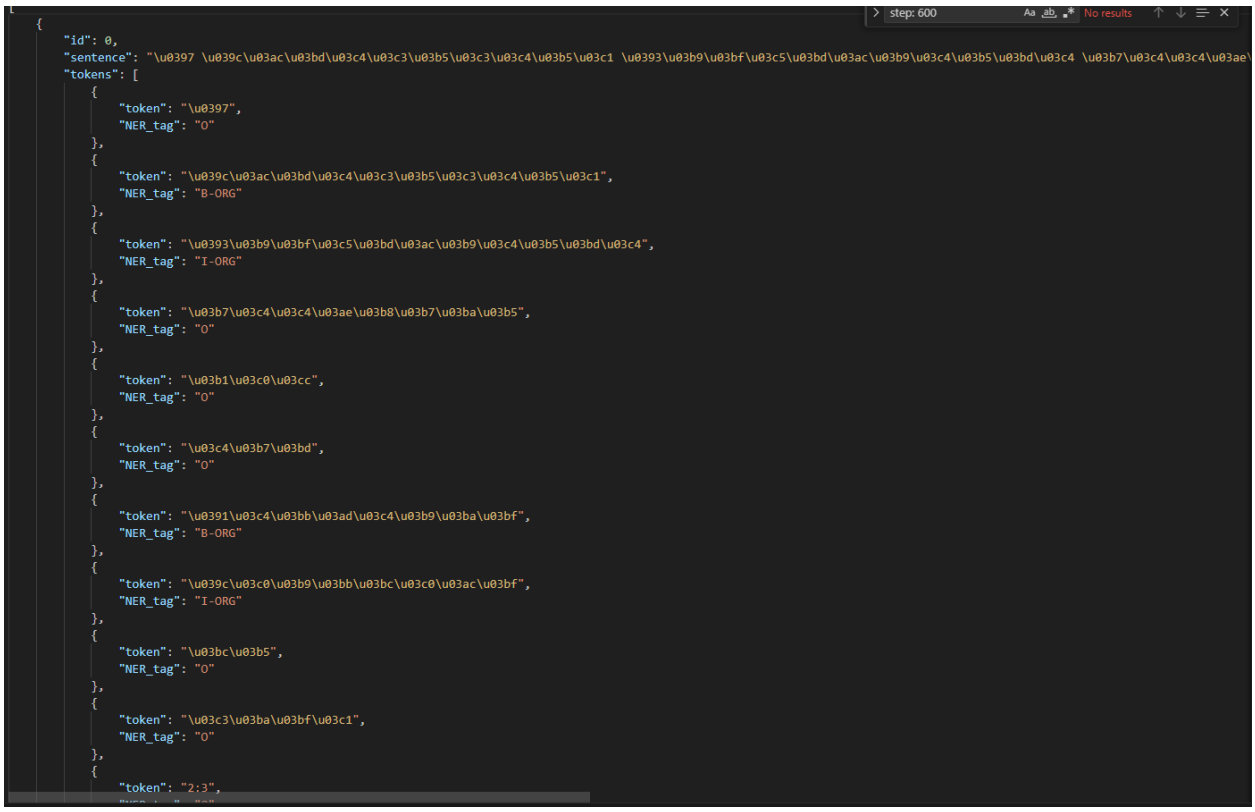


Figure 5.2: NER dataset example

Hyperparameter	GreekDeBERTa$_{base}$	GreekDeBERTaV3$_{xsmall}$	GreekDeBERTaV3$_{base}$
Learning Rate	1e-5	1e-4	1e-5
Batch Size	16	16	16
Epochs	4	6	7
Weight Decay	0.01	0	0
Warmup Steps	12.5% of total steps	13.6% of total steps	10% of total steps
Scheduler Type	linear	linear	linear

Table 5.3: Hyperparameters for the GreekDeBERTa models on NER

smaller model more epochs to learn effectively, while the larger models, which may take longer to converge, are given more epochs for thorough training without risking overfitting. **Weight Decay** is applied selectively; GreekDeBERTa$_{base}$ uses a decay of 0.01 to prevent overfitting. The other models do not use weight decay, suggesting that regularization was not necessary for achieving good performance, possibly due to different optimization dynamics or the nature of the datasets used. **Warmup Steps** are proportionally set, with 12.5%, 13.6%, and 10% of the total steps for GreekDeBERTa$_{base}$, GreekDeBERTaV3$_{xsmall}$, and GreekDeBERTaV3$_{base}$, respectively. These warmup steps help the models start training gradually, which stabilizes the initial learning phase. All models use a **linear Scheduler Type**. The **Max Sequence Length** is fixed at 408 tokens for all models.

The accuracy of the models is presented in the Table 5.4, f1 macro was used for this task.

The results of the NER task reveal that the DeBERTa models outperform the previously best model, GREEK-BERT. Specifically, **GreekDeBERTa$_{base}$** achieves the highest accuracy of 86.5% with a standard deviation of 0.5, indicating a notable improvement over GREEK-BERT's 85.7% accuracy. The reduced standard deviation also suggests that the

Model	Accuracy (%)	Execution Time (mm:ss)
GREEK-BERT	85.7 $\pm$ 1.0	08:30
GreekDeBERTa$_{base}$	**86.5** $\pm$ 0.5	08:42
GreekDeBERTaV3$_{xsmall}$	85.8 $\pm$ 0.5	06:55
GreekDeBERTaV3$_{base}$	86.0 $\pm$ 0.4	21:39
DistilGREEK-BERT	83.5	06:33

Table 5.4: Performance of the GreekDeBERTa models on NER task.

GreekDeBERTa$_{base}$ model performs more consistently. Similarly, **GreekDeBERTaV3$_{xsmall}$** and **GreekDeBERTaV3$_{base}$** achieve accuracies of 85.8% and 86.0%, respectively, with standard deviations of 0.5 and 0.4. These results indicate that even the smaller model, GreekDeBERTaV3$_{xsmall}$, maintains competitive performance, offering a balance between efficiency and accuracy. The slight differences in standard deviations further highlight the consistency of these models compared to GREEK-BERT.

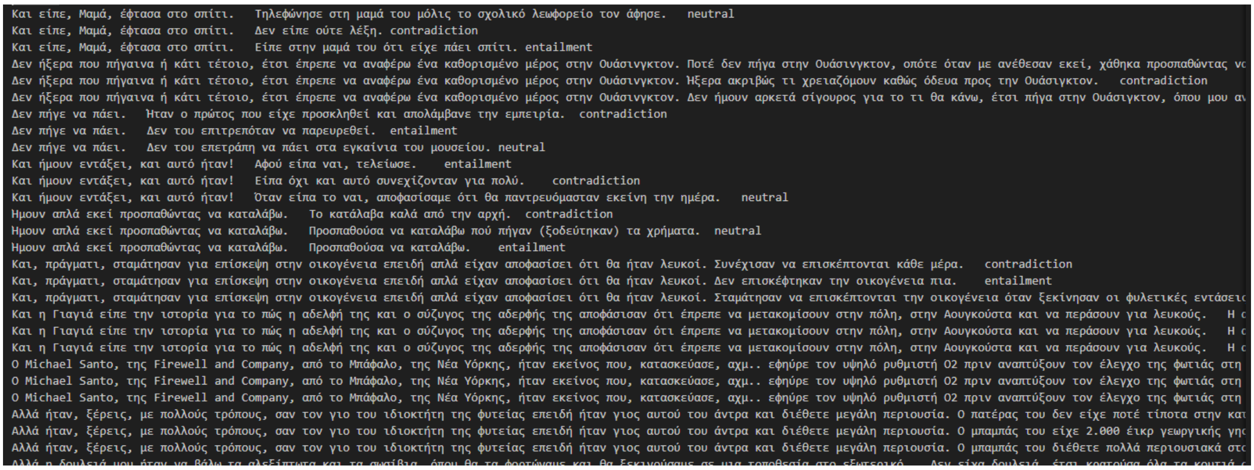
When considering execution time, **GreekDeBERTaV3$_{xsmall}$** demonstrates efficiency, taking 6 minutes and 55 seconds to complete, which is faster than the larger **GreekDeBERTa$_{base}$**, which requires 8 minutes and 42 seconds, despite offering only a minor difference in accuracy. The longest execution time is observed with **GreekDeBERTaV3$_{base}$**, which, while achieving a competitive accuracy of 86.0%, takes 21 minutes and 39 seconds to complete the task. **DistilGREEK-BERT**, though the fastest at 6 minutes and 33 seconds, delivers the lowest accuracy at 83.5%. This demonstrates a clear trade-off between execution time and model accuracy, where the DeBERTa models prioritize accuracy, while DistilGREEK-BERT focuses on speed.

5.3 NLI

Natural Language Inference (NLI) is the task of determining whether a given ”hypothesis” logically follows from a ”premise.” The goal is to classify the relationship between these two statements into three categories: entailment, contradiction, or neutral. In entailment, the hypothesis is true if the premise is true; in contradiction, the hypothesis is false if the premise is true; and in the neutral case, the truth value of the hypothesis cannot be determined from the premise alone. NLI is closely related to fact-checking, where the task includes searching for relevant premises before determining the relationship[53].

5.3.1 Dataset

For this task the Cross-lingual Natural Language Inference (XNLI) [3] corpus was used, which consists of 5,000 test pairs and 2,500 development pairs derived from the MultiNLI corpus[1]. These pairs were originally in English and categorized by human annotators. Afterwards, they were translated into 14 languages, including Greek, by professional translators, ensuring consistent class labels across all languages. The MultiNLI dataset includes a training set of 340,000 pairs. For XNLI, this training set was automatically translated into the other languages, leading to potential quality variations. In this study, we focused solely on the Greek pairs for fine-tuning our model. Following DistilGREEK-BERT [50] only Greek pairs in each set were kept and unnecessary fields were removed. An example of the final dataset is displayed in the figure below. The dataset was loaded



```
Και είπε, Μαμά, έφτασα στο σπίτι.   Τηλεφώνησε στη μαμά του μόλις το σχολικό λεωφορείο τον άφησε.   neutral
Και είπε, Μαμά, έφτασα στο σπίτι.   Δεν είπε ούτε λέξη. contradiction
Και είπε, Μαμά, έφτασα στο σπίτι.   Είπε στην μαμά του ότι είχε πάει σπίτι. entailment
Δεν ήξερα που πήγαινα ή κάτι τέτοιο, έτσι έπρεπε να αναφέρω ένα καθορισμένο μέρος στην Ουάσινγκτον. Ποτέ δεν πήγα στην Ουάσινγκτον, οπότε όταν με ανέθεσαν εκεί, χάθηκα προσπαθώντας να
Δεν ήξερα που πήγαινα ή κάτι τέτοιο, έτσι έπρεπε να αναφέρω ένα καθορισμένο μέρος στην Ουάσινγκτον. Ηξερα ακριβώς τι χρειαζόμουν καθώς όδευα προς την Ουάσιγκτον.   contradiction
Δεν ήξερα που πήγαινα ή κάτι τέτοιο, έτσι έπρεπε να αναφέρω ένα καθορισμένο μέρος στην Ουάσινγκτον. Δεν ήμουν αρκετά σίγουρος για το τι θα κάνω, έτσι πήγα στην Ουάσιγκτον, όπου μου α
Δεν πήγε να πάει.   Ηταν ο πρώτος που είχε προσκληθεί και απολάμβανε την εμπειρία.  contradiction
Δεν πήγε να πάει.   Δεν του επιτρεπόταν να παρευρεθεί.  entailment
Δεν πήγε να πάει.   Δεν του επετράπη να πάει στα εγκαίνια του μουσείου. neutral
Και ήμουν εντάξει, και αυτό ήταν!   Αφού είπα ναι, τελείωσε.    entailment
Και ήμουν εντάξει, και αυτό ήταν!   Είπα όχι και αυτό συνεχίζονταν για πολύ.    contradiction
Και ήμουν εντάξει, και αυτό ήταν!   Όταν είπα το ναι, αποφασίσαμε ότι θα παντρευόμασταν εκείνη την ημέρα.   neutral
Ημουν απλά εκεί προσπαθώντας να καταλάβω.   Το κατάλαβα καλά από την αρχή.  contradiction
Ημουν απλά εκεί προσπαθώντας να καταλάβω.   Προσπαθούσα να καταλάβω πού πήγαν (ξοδεύτηκαν) τα χρήματα.  neutral
Ημουν απλά εκεί προσπαθώντας να καταλάβω.   Προσπαθούσα να καταλάβω.    entailment
Και, πράγματι, σταμάτησαν για επίσκεψη στην οικογένεια επειδή απλά είχαν αποφασίσει ότι θα ήταν λευκοί. Συνέχισαν να επισκέπτονται κάθε μέρα.   contradiction
Και, πράγματι, σταμάτησαν για επίσκεψη στην οικογένεια επειδή απλά είχαν αποφασίσει ότι θα ήταν λευκοί. Δεν επισκέφτηκαν την οικογένεια πια.    entailment
Και, πράγματι, σταμάτησαν για επίσκεψη στην οικογένεια επειδή απλά είχαν αποφασίσει ότι θα ήταν λευκοί. Σταμάτησαν να επισκέπτονται την οικογένεια όταν ξεκίνησαν οι φυλετικές εντάσεις
Και η Γιαγιά είπε την ιστορία για το πώς η αδελφή της και ο σύζυγος της αδερφής της αποφάσισαν ότι έπρεπε να μετακομίσουν στην πόλη, στην Αουγκούστα και να περάσουν για λευκούς.   Η
Και η Γιαγιά είπε την ιστορία για το πώς η αδελφή της και ο σύζυγος της αδερφής της αποφάσισαν ότι έπρεπε να μετακομίσουν στην πόλη, στην Αουγκούστα και να περάσουν για λευκούς.   Η
Και η Γιαγιά είπε την ιστορία για το πώς η αδελφή της και ο σύζυγος της αδερφής της αποφάσισαν ότι έπρεπε να μετακομίσουν στην πόλη, στην Αουγκούστα και να περάσουν για λευκούς.   Η
Ο Michael Santo, της Firewell and Company, από το Μπάφαλο, της Νέα Υόρκης, ήταν εκείνος που, κατασκεύασε, αχμ.. εφηύρε τον υψηλό ρυθμιστή Ο2 πριν αναπτύξουν τον έλεγχο της φωτιάς στη
Ο Michael Santo, της Firewell and Company, από το Μπάφαλο, της Νέα Υόρκης, ήταν εκείνος που, κατασκεύασε, αχμ.. εφηύρε τον υψηλό ρυθμιστή Ο2 πριν αναπτύξουν τον έλεγχο της φωτιάς στη
Ο Michael Santo, της Firewell and Company, από το Μπάφαλο, της Νέα Υόρκης, ήταν εκείνος που, κατασκεύασε, αχμ.. εφηύρε τον υψηλό ρυθμιστή Ο2 πριν αναπτύξουν τον έλεγχο της φωτιάς στη
Αλλά ήταν, ξέρεις, με πολλούς τρόπους, σαν τον γιο του ιδιοκτήτη της φυτείας επειδή ήταν γιος αυτού του άντρα και διέθετε μεγάλη περιουσία. Ο πατέρας του δεν είχε ποτέ τίποτα στην κα
Αλλά ήταν, ξέρεις, με πολλούς τρόπους, σαν τον γιο του ιδιοκτήτη της φυτείας επειδή ήταν γιος αυτού του άντρα και διέθετε μεγάλη περιουσία. Ο μπαμπάς του είχε 2.000 έικρ γεωργικής γης
Αλλά ήταν, ξέρεις, με πολλούς τρόπους, σαν τον γιο του ιδιοκτήτη της φυτείας επειδή ήταν γιος αυτού του άντρα και διέθετε μεγάλη περιουσία. Ο μπαμπάς του διέθετε πολλά περιουσιακά στο
```

Figure 5.3: NLI dataset example

from the tab-separated file, the labels were mapped according to a custom dictionary, and the resulting data was converted into a `DatasetDict` with train, validation, and test splits.

5.3.2 Hyperparameters and Results

The hyperparameters for all the models are shown in the Table 5.5. Based on the trials

Model	**GreekDeBERTa$_{\text{base}}$**	**GreekDeBERTaV3$_{\text{xsmall}}$**	**GreekDeBERTa$_{\text{base}}$**
Batch Size	16	16	16
Learning Rate	1e-5	1e-5	1e-5
Epochs	5	5	5
Warmup Steps	1% of total steps	1% of total steps	1% of total steps
Weight Decay	0.01	0.01	0.01
Scheduler Type	Linear	Linear	Linear

Table 5.5: Hyperparameters for the GreekDeBERTa models for NLI.

conducted on this larger dataset, it was observed that the hyperparameters remained consistent across all the GreekDeBERTa models used for NLI. Despite experimenting with changes in the hyperparameters, such as adjustments in the batch size, learning rate, epochs, warmup steps, weight decay, and scheduler type, there was no significant impact on the accuracy. As a result, the final hyperparameters adopted were uniform across all models: a batch size of 16, a learning rate of 1e-5, 5 epochs, warmup steps set to 1% of total steps, a weight decay of 0.01, and a linear scheduler type. The consistency in hyperparameters reflects a balanced optimization strategy for this task. The **Max Sequence Length** is fixed at 356 tokens for all models.

Since the dataset was balanced, micro macro and weighted accuracy was almost identical, in any case, we are using the f1-macro. The results are displayed in the Table 5.6.

The results of the NLI task reveal that the DeBERTa models outperform the previously best model, GREEK-BERT. Specifically, **GreekDeBERTaV3$_{\text{base}}$** achieves the highest accuracy of 80.0% with a standard deviation of 0.3, marking a significant improvement over GREEK-BERT's 78.6% accuracy. The reduced standard deviation suggests that the **GreekDeBERTaV3$_{\text{base}}$** model performs more consistently. Similarly, **GreekDeBERTa$_{\text{base}}$** and **GreekDeBERTaV3$_{\text{xsmall}}$** achieve accuracies of 79.4% and 78.9%, respectively, with

Model	Accuracy (%)	Execution Time (hh:mm:ss)
GREEK-BERT	78.6 $\pm$ 0.62	11:53:11
GreekDeBERTa$_{base}$	79.4 $\pm$ 0.4	16:00:51
GreekDeBERTaV3$_{xsmall}$	78.9 $\pm$ 0.5	07:47:12
GreekDeBERTaV3$_{base}$	**80.0** $\pm$ 0.3	16:00:44
DistilGREEK-BERT	74.47	04:02:41

Table 5.6: Performance of the GreekDeBERTa models on NLI task.

standard deviations of 0.4 and 0.5. These results indicate that even the smaller model, GreekDeBERTaV3$_{xsmall}$, maintains competitive performance, offering a balance between efficiency and accuracy. The slight differences in standard deviations across the models further highlight the consistency and reliability of the GreekDeBERTa models compared to GREEK-BERT.

When considering execution time, the results show a significant trade-off between accuracy and efficiency. While **GreekDeBERTaV3$_{base}$** delivers the highest accuracy, it also has a lengthy execution time of 16 hours, similar to **GreekDeBERTa$_{base}$**, which also takes 16 hours to run. On the other hand, **GreekDeBERTaV3$_{xsmall}$** , with an accuracy of 78.9%, is much faster, completing the task in 7 hours and 47 minutes. **DistilGREEK-BERT**, although the fastest model with an execution time of 4 hours and 2 minutes, sacrifices accuracy, achieving only 74.47%. This illustrates that while the GreekDeBERTa models are more accurate, they come at the cost of increased execution time, with smaller models like **GreekDeBERTaV3$_{xsmall}$** offering a balance between accuracy and efficiency.

5.3.3 Overall Performance Assessment

In this work, we have explored the application of several DeBERTa variants for the Greek language, benchmarking them against the previously leading GREEK-BERT model across three key NLP tasks: Part-of-Speech (PoS) tagging, Named Entity Recognition (NER), and Natural Language Inference (NLI). The results demonstrate that the GreekDeBERTa models consistently outperform GREEK-BERT in terms of accuracy, indicating their superior capability in handling Greek language tasks.

The GreekDeBERTa$_{base}$ model achieved the highest performance in the PoS task with an accuracy of 98.4%, showing a slight but significant improvement over GREEK-BERT's 98.1%. For the NER task, the same model again led the pack with an accuracy of 86.5%, surpassing GREEK-BERT's 85.7%. In the NLI task, the GreekDeBERTaV3$_{base}$ variant topped the results with an accuracy of 80.0%, a notable advancement over the baseline model's 78.6%.

While the base variants of GreekDeBERTa offer top-tier performance, they are relatively resource-intensive due to their larger size. In contrast, the GreekDeBERTaV3$_{xsmall}$ model, though slightly behind in accuracy compared to its larger counterparts, offers a practical trade-off by being approximately 51% faster. This model's fine-tuning speed is comparable to that of GREEK-BERT, making it a compelling choice for scenarios where computational efficiency and speed are critical considerations.

In terms of execution time, we observe a clear trade-off between accuracy and computational efficiency. For instance, the highest accuracy for the NLI task, achieved by the GreekDeBERTaV3$_{base}$ model, required over 16 hours of processing, highlighting the significant resource demand for state-of-the-art models. On the other hand, GreekDeBERTaV3$_{xsmall}$

, while slightly less accurate, completed the same task in under 8 hours, offering a much more feasible solution for time-sensitive or resource-constrained applications. This trend is consistent across all tasks, where the smaller DeBERTa variants maintain a balance between accuracy and speed, making them suitable for real-time or large-scale applications where minimizing execution time is crucial.

Additionally, models like DistilGREEK-BERT, although the fastest with execution times often below 5 hours, suffer from a considerable drop in accuracy. This further underscores the importance of choosing models based on the specific requirements of the task—whether the priority is maximizing accuracy or optimizing execution time and computational efficiency.

Overall, the integration of DeBERTa models into the Greek NLP landscape presents a significant advancement, combining state-of-the-art performance with varying levels of computational efficiency. This study highlights the potential of these models to enhance NLP applications for the Greek language, offering both accuracy improvements and, in the case of the smaller variants, efficiency gains. For practical applications, the choice between models should be guided by the trade-offs between accuracy and processing time, depending on the nature and constraints of the task.

6. CONCLUSIONS

In this thesis, we explored several significant advancements in NLP, focusing on transformer-based models. We began with an overview of BERT, which revolutionized NLP by employing bidirectional training to capture context from both directions. This was a pivotal shift from previous models that were limited by unidirectional constraints. The introduction of BERT demonstrated a significant leap in the ability of models to understand and generate natural language with high accuracy, especially in tasks requiring nuanced understanding and contextual awareness.

The background chapter provided a thorough exploration of fundamental concepts in AI, ML, and NLP. This chapter laid the groundwork for understanding the technical details and innovations presented in the subsequent sections, including various types of NN, learning methodologies, and the challenges involved in NLP.

We extended our exploration to the adaptation of BERT for the Greek language, with the development of GREEK-BERT. This model addressed specific linguistic challenges inherent to Greek and outperformed multilingual models by providing a more precise and contextually relevant representation for Greek language tasks. The focus on creating a specialized vocabulary for Greek, rather than relying on fragmented word representations, enabled GREEK-BERT to achieve superior performance in tasks such as Part-of-Speech tagging, Named Entity Recognition, and Natural Language Inference.

Further advancements were discussed with the introduction of DeBERTa and its successor, DeBERTaV3. DeBERTa incorporated innovative mechanisms such as disentangled attention and an enhanced mask decoder, which improved the efficiency and effectiveness of the model. DeBERTaV3 advanced these innovations by integrating the RTD task from ELECTRA and introducing GDES. These enhancements allowed DeBERTaV3 to set new benchmarks in various NLP tasks, showcasing significant improvements in both training efficiency and model performance.

The final development was the creation of the **GreekDeBERTa** family, consisting of three models specifically optimized for the Greek language: **GreekDeBERTa$_{\text{base}}$**, **GreekDeBERTaV3$_{\text{xsmall}}$**, and **GreekDeBERTaV3$_{\text{base}}$**. By utilizing the strengths of DeBERTa's architecture and adapting it to the unique linguistic features of Greek, these models achieved state-of-the-art results across several key NLP benchmarks. Each variant of GreekDeBERTa offers a balance between performance and efficiency, allowing for flexibility depending on the specific needs of a task. This work demonstrated the importance of customized language models in achieving high performance in specialized language contexts, particularly for underrepresented languages in NLP research.

In conclusion, the advancements presented in this thesis highlight the continuous evolution of transformer models and their application in NLP. Each model iteration built upon the limitations and strengths of its predecessors, contributing to a deeper understanding of natural language and improved capabilities in language-related tasks. Future research can further enhance these models by exploring more compact and efficient architectures, broadening their applicability, and ensuring they are accessible even in resource-constrained environments.

ABBREVIATIONS - ACRONYMS

AI	Artificial Intelligence
ML	Machine Learning
NLP	Natural Language Processing
NN	Neural Networks
GPT	Generative Pre-trained Transformer
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
BERT	Bidirectional Encoder Representations for Transformers
M-BERT	Multilingual BERT
MLM	Masked Language Modeling
NSP	Next Sentence Prediction
PoS Tag-ging	Part of Speech Tagging
NER	Named Entity Recognition
NLI	Natural Language Inference
Europarl	European Parliament Proceedings Parallel Corpus
SVM	Support Vector Machines
AWS	Amazon Web Services
ANN	Artificial Neural Networks
GLUE	General Language Understanding Evaluation
mDeBERTa	Multilingual DeBERTa
NLU	Natural Language Understanding
GAN	Generative Adversarial Network
BPTT	Backpropagation Through Time
GREEK-BERT	Greek edition of BERT
DeBERTa	Decoding-enhanced BERT with disentangled attention
MSE	Mean Squared Error
ELECTRA	Efficiently Learning an Encoder that Classifies Token Replacements Accurately

GDES	Gradient-Disentangled Embedding Sharing
GDT	Greek Dependency Treebank
RTD	Replaced Token Detection
OSCAR	Open Super-large Crawled Aggregated coRpus
GloVe	Global Vectors for Word Representation
WSD	Word Sense Disambiguation
PLM	Pre-trained Language Models
ELMo	Embeddings from Language Models
SQuAD	Stanford Question Answering Dataset

BIBLIOGRAPHY

[1] Samuel Bowman Adina Williams, Nikita Nangia. A broad-coverage challenge corpus for sentence understanding through inference. "https://aclanthology.org/N18-1101/.

[2] Charu C. Aggarwal. Neural networks and deep learning. https://eclass.upatras.gr/modules/document/file.php/EE935/%CE%97%CE%BB%CE%B5%CE%BA%CF%84%CF%81%CE%BF%CE%BD%CE%B9%CE%BA%CE%AC%20%CE%92%CE%B9%CE%B2%CE%BB%CE%AF%CE%B1/2018_Book_NeuralNetworksAndDeepLearning.pdf.

[3] Guillaume Lample Adina Williams Samuel Bowman Holger Schwenk Veselin Stoyanov Alexis Conneau, Ruty Rinott. Xnli: Evaluating cross-lingual sentence representations. "https://aclanthology.org/D18-1269/.

[4] Amanatullah. Vanishing gradient problem in deep learning: Understanding, intuition, and solutions. https://medium.com/@amanatulla1606/vanishing-gradient-problem-in-deep-learning-understanding-intuition-and-solutions-da90ef4ecb54.

[5] Amazon. Model fit: Underfitting vs. overfitting. https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html.

[6] Amazon. Validate a machine learning model. https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-model-validation.htmll.

[7] Amazon. What is deep learning? https://aws.amazon.com/what-is/deep-learning/.

[8] amazon. What is machine translation? https://aws.amazon.com/what-is/machine-translation/.

[9] Amazon. What is reinforcement learning? https://aws.amazon.com/what-is/reinforcement-learning/.

[10] amazon. What is sentiment analysis? https://aws.amazon.com/what-is/sentiment-analysis/.

[11] Babak Saboury Eliot Siegel Amirhosein Toosi, Andrea Bottino and Arman Rahmim. A brief history of ai: How to prevent another winter. https://arxiv.org/abs/2109.01517, 2022.

[12] Rockwell Anyoha. The history of artificial intelligence. https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/, 2017.

[13] Athina. Institue for language and speech processing. https://www.ilsp.gr/en/home-2/.

[14] Dzmitry Bahdanau. Neural machine translation by jointly learning to align and translate. https://arxiv.org/pdf/1409.0473.

[15] Isabelle M. Guyon Bernhard E. Boser and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. https://dl.acm.org/doi/10.1145/130385.130401, 1992.

[16] Vrunda Bhattbhatt. Learning rate and its strategies in neural network training. https://medium.com/thedeephub/learning-rate-and-its-strategies-in-neural-network-training-270a91ea0e5c.

[17] Alice Zheng Amanda Casari. Feature engineering for machine learning. https://www.repath.in/gallery/feature_engineering_for_machine_learning.pdf.

[18] Sasirekha Cota. Deep learning basics — feed forward neural networks (ffnn). https://medium.com/@sasirekharameshkumar/deep-learning-basics-part-10-feed-forward-neural-networks-ffnn-93a708f84a3

[19] I. Darras. Google summer of code 2018 project - spacy now speaks greek. https://explosion.ai/blog/spacy-now-speaks-greek.

[20] Peter Dayan. Unsupervised learning. https://www.gatsby.ucl.ac.uk/~dayan/papers/dun99b.pdf.

[21] Ketan Doshi. Transformers explained visually (part 1): Overview of functionality. https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd46

[22] Vyacheslav Efimov. Large language models: Deberta — decoding-enhanced bert with disentangled attention. https://towardsdatascience.com/large-language-models-deberta-decoding-enhanced-bert-with-disentangled-attention-90016668db4b.

[23] Hugging Face. Datasets hugging face. https://pypi.org/project/datasets/.

[24] Hugging Face. Optimization hugging face. https://huggingface.co/docs/transformers/main_classes/trainer.

[25] Hugging Face. Trainer hugging face. https://huggingface.co/docs/transformers/main_classes/trainer.

[26] Jeff Dean Geoffrey Hinton, Oriol Vinyals. Distilling the knowledge in a neural network. "https://arxiv.org/pdf/1503.02531.

[27] Rebeen Hamad. What is lstm? introduction to long short-term memory. https://medium.com/@rebeen.jaff/what-is-lstm-introduction-to-long-short-term-memory-66bd3855b9ce.

[28] Simon Haykin. Neural networks and learning machines. https://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf.

[29] Aaron Courville Ian Goodfellow, Yoshua Bengio. Deep learning. http://alvarestech.com/temp/deep/Deep%20Learning%20by%20Ian%20Goodfellow,%20Yoshua%20Bengio,%20Aaron%20Courville%20(z-lib.org).pdf.

[30] IBM. Five machine learning types to know. https://www.ibm.com/blog/machine-learning-types/.

[31] IBM. What are recurrent neural networks? https://www.ibm.com/topics/recurrent-neural-networks.

[32] IBM. What is a neural network? https://www.ibm.com/topics/neural-networks.

[33] IBM. What is a transformer model? https://www.ibm.com/topics/transformer-model.

[34] IBM. What is artificial intelligence? https://www.ibm.com/topics/artificial-intelligence.

[35] IBM. What is deep learning? https://www.ibm.com/topics/deep-learning.

[36] IBM. What is ml? https://www.ibm.com/topics/machine-learning.

[37] IBM. What is named entity recognition? https://www.ibm.com/topics/named-entity-recognition.

[38] IBM. What is named entity recognition? https://www.ibm.com/topics/named-entity-recognition.

[39] IBM. What is natural language processing (nlp)? https://www.ibm.com/topics/natural-language-processing.

[40] IBM. What is self-supervised learning? https://www.ibm.com/topics/self-supervised-learning.

[41] IBM. What is semi-supervised learning? https://www.ibm.com/topics/semi-supervised-learning.

[42] IBM. What is speech recognition? https://www.ibm.com/topics/speech-recognition.

[43] IBM. What is text generation? https://www.ibm.com/topics/text-generation.

[44] N. Rochester J. McCarthy, M. L. Minsky and C.E. Shannon. A proposal for the dartmouth summer research project on artificial intelligence. http://jmc.stanford.edu/articles/dartmouth/dartmouth.pdf, 1955.

[45] Kenton Lee Kristina Toutanova Jacob Devlin, Ming-Wei Chang. Bert: Pre-training of deep bidirectional transformers for language understanding. https://arxiv.org/pdf/1810.04805.

[46] Mirantha Jayathilaka. 25 nlp tasks at a glance. https://medium.com/@miranthaj/25-nlp-tasks-at-a-glance-52e3fdff32e2.

[47] Christopher D. Manning Jeffrey Pennington, Richard Socher. Glove: Global vectors for word representation. https://aclanthology.org/D14-1162.pdf.

[48] Prodromos Malakasiotis Ion Androutsopoulos John Koutsikakis, Ilias Chalkidis. Greek-bert: The greeks visiting sesame street. https://arxiv.org/abs/2008.12014.

[49] Karen Sparck Jones. Natural language processing: A historical review. https://aclanthology.org/www.mt-archive.info/Zampolli-1994-Sparck-Jones.pdf.

[50] Eftychia A. Karavangeli. Distilgreek-bert: A distilled version of the greek-bert model. https://pergamos.lib.uoa.gr/uoa/dl/object/3338746/file.pdf.

[51] Omer Levy Christopher D. Manning Kevin Clark, Urvashi Khandelwal. What does bert look at? an analysis of bert's attention. https://aclanthology.org/W19-4828.pdf.

[52] Quoc V. Le Christopher D. Manning Kevin Clark, Minh-Thang Luong. Electra: Pre-training text encoders as discriminators rather than generators. https://arxiv.org/pdf/2003.10555.

[53] Oleh Lokshyn. Natural language inference: An overview. "https://towardsdatascience.com/natural-language-inference-an-overview-57c0eecf6517.

[54] Marco Leonardi Ashraf Mohamed Alessandro Rozza Lorenzo Ciampiconi, Adam Elwood. A survey and taxonomy of loss functions in machine learning. https://ar5iv.labs.arxiv.org/html/2301.05579.

[55] L.C. Jain L.R. Medsker. Recurrent neural networks : Design and applications. https://theswissbay.ch/pdf/Gentoomen%20Library/Artificial%20Intelligence/Neural%20networks/Recurrent%20Neural%20Networks%20Design%20And%20Applications%20-%20L.R.%20Medsker.pdf.

[56] Microsoft. Deberta-v2 hugging face. https://huggingface.co/microsoft/deberta-v3-base/tree/main.

[57] Sujatha Mudadla. What is parts of speech (pos) tagging natural language processing?in what kind of applications we can use parts of speech (pos) tagging in natural language processing. https://medium.com/@sujathamudadla1213/what-is-parts-of-speech-pos-tagging-natural-language-processing-in-2b8f4b07b186.

[58] International Research Journal of Modernization in Engineering Technology and Science. Machine learning: A review of learning types. https://www.irjmets.com/uploadedfiles/paper//issue_9_september_2022/29824/final/fin_irjmets1662994184.pdf.

[59] om pramod. Disambiguating words: A deep dive into word sense disambiguation. https://medium.com/@ompramod9921/disambiguating-words-a-deep-dive-into-word-sense-disambiguation-b5f1a845747e.

[60] Simon Palma. Understanding weight update in neural networks. https://medium.com/@simon.palma/understanding-weight-update-in-neural-networks-part-2-d445cce09d67.

[61] Jianfeng Gao Weizhu Chen Pengcheng He, Xiaodong Liu. Deberta: Decoding-enhanced bert with disentangled attention. https://arxiv.org/pdf/2006.03654.

[62] Weizhu Chen Pengcheng He, Jianfeng Gao. Debertav3: Improving deberta using electra-style pretraining with gradientdisentangled embedding sharing. https://arxiv.org/pdf/2111.09543.

[63] Haris Papageorgiou Prokopis Prokopidis. Universal dependencies for greek. https://aclanthology.org/W17-0413.pdf.

[64] PyPI. conllu 5.0.1. https://pypi.org/project/conllu/.

[65] Pytorch. Lstm. https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html.

[66] Bijula Ratheesh. Word embeddings, wordpiece and language-agnostic bert (labse). https://bijular.medium.com/word-embeddings-wordpiece-and-language-agnostic-bert-labse-98c7626878c7.

[67] Stuart J. Russell and Peter Norvig. Artificial intelligence a modern approach. https://people.engr.tamu.edu/guni/csce421/files/AI_Russell_Norvig.pdf.

[68] Jürgen Schmidhuber. Who invented backpropagation? https://people.idsia.ch/~juergen/who-invented-backpropagation.html, 2014.

[69] Robin M. Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. https://arxiv.org/pdf/1912.05911.

[70] Jurgen Schmidhuber Sepp Hochreiter. Long short-term memory. https://www.bioinf.jku.at/publications/older/2604.pdf.

[71] Sklearn. Metrics and scoring: quantifying the quality of predictions. https://scikit-learn.org/stable/modules/model_evaluation.html.

[72] Prudhviraju Srivatsavaya. Lstm — implementation, advantages and diadvantages. https://medium.com/@prudhviraju.srivatsavaya/lstm-implementation-advantages-and-diadvantages-914a96fa0acb.

[73] Richard S. Sutton and Andrew G. Barto. Reinforcement learning:an introduction, 2014, 2015.

[74] TensorFlow. Lstm. https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM.

[75] John Terra. Regression vs. classification in machine learning for beginners. https://www.simplilearn.com/regression-vs-classification-in-machine-learning-article.

[76] Greg Corrado Jeffrey Dean Tomas Mikolov, Kai Chen. Efficient estimation of word representations in vector space. https://arxiv.org/pdf/1301.3781v3.

[77] Greek Dependency Treebank. Greek dependency treebank. http://gdt.ilsp.gr/.

[78] Alan M. Turing. Computing machinery and intelligence. https://academic.oup.com/mind/article/LIX/236/433/986238, 1950.

[79] UD. Greek-gdt. https://github.com/UniversalDependencies/UD_Greek-GDT.

[80] RAHULRAJ P V. What are the query, key, and value vectors? https://rahulrajpvr7d.medium.com/what-are-the-query-key-and-value-vectors-5656b8ca5fa0.

[81] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. https://arxiv.org/pdf/1706.03762v7.

[82] Julien CHAUMOND Thomas WOLF Victor SANH, Lysandre DEBUT. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. "https://arxiv.org/pdf/1910.01108.

[83] Rui Yang Viktoria Szabo, Marianna Plesiak. Modern approaches in natural language processing. https://slds-lmu.github.io/seminar_nlp_ss20/.

[84] Naman Goyal Jingfei Du Mandar Joshi Danqi Chen Omer Levy Mike Lewis Luke Zettlemoyer Veselin Stoyanov Yinhan Liu, Myle Ott. Roberta: A robustly optimized bert pretraining approach. "https://chatgpt.com/c/f6894443-fe1a-47bf-9254-123d85dfaea3.

[85] Zhifeng Chen Quoc V Le Mohammad Norouzi Wolfgang Macherey Maxim Krikun Yuan Cao Qin Gao Klaus Macherey et al. Yonghui Wu, Mike Schuster. Google's neural machine translation system: Bridging the gap between human and machine translation. https://arxiv.org/pdf/1609.08144.

[86] Richard Zemel Ruslan Salakhutdinov Raquel Urtasun1 Antonio Torralba Sanja Fidler Yukun Zhu, Ryan Kiros. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. "https://arxiv.org/pdf/1506.06724.

[87] Jiawei Zhang. Gradient descent based optimization algorithms for deep learning models training. https://arxiv.org/pdf/1903.03614.