Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Large Language Models, Adapters and Perplexity Scores for Multigenerator, Multidomain, and Multilingual Black-Box Machine-Generated Text Detection

Diploma Thesis

by

Dimitrios Andreadis

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2024

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Συστημάτων Τεχνητής Νοημοσύνης και Μάθησης

Large Language Models, Adapters and Perplexity Scores for Multigenerator, Multidomain, and Multilingual Black-Box Machine-Generated Text Detection

Diploma Thesis

by

Dimitrios Andreadis

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 16^{η} Ιουλίου, 2024.

....	....	....
Γεώργιος Στάμου	Αθανάσιος Βουλόδημος	Ανδρέας-Γεώργιος Σταφυλοπάτης
Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π

Αθήνα, Ιούλιος 2024

....
Δημητριος Ανδρεαδης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Τα μεγάλα γλωσσικά μοντέλα (Large Language Models - LLMs) γίνονται κυρίαρχα και εύκολα προσβάσιμα, οδηγώντας σε μια έκρηξη περιεχομένου που δημιουργείται από μηχανές σε διάφορα κανάλια επικοινωνίας, όπως ειδήσεις, κοινωνικά μέσα, φόρουμ απαντήσεων ερωτήσεων, εκπαιδευτικά και ακόμη και ακαδημαϊκά πλαίσια. Πρόσφατα LLMs όπως το ChatGPT και το GPT-4, δημιουργούν εξαιρετικά άπταιστες απαντήσεις σε μια μεγάλη ποικιλία ερωτημάτων χρηστών. Η αρθρωτή φύση τέτοιων παραγόμενων κειμένων καθιστά τα LLMs ελκυστικά για την αντικατάσταση της ανθρώπινης εργασίας σε πολλά σενάρια. Ωστόσο, ανησυχίες έχουν προκύψει σχετικά με την πιθανή κατάχρησή τους, όπως η διάδοση παραπληροφόρησης και η πρόκληση διαταραχών στο εκπαιδευτικό σύστημα. Δεδομένου ότι οι άνθρωποι αποδίδουν μόνο ελαφρώς καλύτερα από την τύχη στην ανίχνευση κειμένου που δημιουργείται από μηχανή, υπάρχει ανάγκη ανάπτυξης αυτόματων συστημάτων με στόχο τον μετριασμό της πιθανής κακής χρήσης των κειμένων αυτών. Αυτή την ανάγκη πραγματεύεται το 8^{o} ερώτημα του Συνεδρίου Σημαντικής Αξιολόγησης (SemEval Workshop 2024).

Σε αυτή τη διατριβή, στοχεύσαμε να κάνουμε ένα ουσιαστικό βήμα προς τη διερεύνηση αυτού του ενδιαφέροντος ερωτήματος, επιχειρώντας να απαντήσουμε στα υποερωτήματα A και B του $8^{ου}$ ερωτήματος του συνεδρίου. Ως σημείο εκκίνησης, πειραματιστήκαμε με την εκπαίδευση προ-εκπαιδευμένων γλωσσικών μοντέλων (Pretrained Language Models - PLMs) για την ανίχνευση κειμένου παραγόμενου από μηχανές (Machine-Generated Text Detection - MGTD), εξετάζοντας την επίδραση των υπερπαραμέτρων εκπαίδευσης στην μετρική της ακρίβειας. Προτείνουμε τη χρήση του προσαρμογέα συντονισμού προτροπής ως αποτελεσματική τεχνική εκπαίδευσης προσαρμογέα που ενισχύει περαιτέρω την επίδοση. Επιπλέον, προσπαθήσαμε να εφαρμόσουμε τα ευρήματά μας στο πιο δύσκολο υποερώτημα της απόδοσης συγγραφέα (Author Atribution - AA). Για την πολύγλωσση περίπτωση του MGTD, προσπαθήσαμε να εντοπίσουμε τη γλώσσα-πηγή και να μεταφράσουμε πολύγλωσσα κείμενα καθώς και να χρησιμοποιήσουμε προσαρμογείς γλώσσας για να ελέγξουμε εάν μπορούν να επιτευχθούν περαιτέρω βελτιώσεις. Εκτός από την εκπαίδευση μοντέλων και προσαρμογέων, διερευνήσαμε επίσης μια άλλη προσέγγιση. Χρησιμοποιώντας πολλαπλά PLMs υπολογίσαμε την περιπλοκότητα σταθερού μήκους.

Συνολικά, αυτή η διατριβή επιχειρεί να αποκαλύψει την προοπτική διαφόρων μεθόδων προς την λύση των προβλημάτων MGTD και AA, συνάγοντας ενδιαφέροντα συμπεράσματα.

Λέξεις-κλειδιά — Ανίχνευση Κειμένου που δημιουργείται από Μηχανή, Απόδοση Συγγραφέα, Προεκπαιδευμένα Γλωσσικά Μοντέλα, Μεγάλα Γλωσσικά Μοντέλα, Πολύγλωσση Περίπτωση, Περιπλοκότητα, Περιπλοκότητα Σταθερού Μήκους, Εκπαίδευση Προσαρμογέα, Προσαρογεάς Συντονισμού Προτροπής

Abstract

Large language models (LLMs) are becoming mainstream and easily accessible, ushering in an explosion of machine-generated content over various channels, such as news, social media, question-answering forums, educational, and even academic contexts. Recent LLMs, such as ChatGPT and GPT-4, generate remarkably fluent responses to a wide variety of user queries. The articulate nature of such generated texts makes LLMs attractive for replacing human labor in many scenarios. However, this has also resulted in concerns regarding their potential misuse, such as spreading misinformation and causing disruptions in the education system. Since humans perform only slightly better than chance when classifying machine-generated vs. human-written text, there is a need to develop automatic systems to identify machine-generated text with the goal of mitigating its potential misuse. This need is addressed by the 8$^{\text{th}}$ task of the SemEval Workshop 2024.

In this thesis, we aimed to make a substantial step towards exploring this interesting task by addresing subtasks A and B for the 8$^{\text{th}}$ SemEval task. As a starting point, we experimented on fine-tuning pre-trained language models (PLMs) for machine-generated text detection (MGTD), examining the effect of the hyperparameters on the accuracy. We suggest the use of prompt tuning as an effective adapter technique that further boosts performance. Moreover, we tried to apply our findings to the more difficult subtask of author attribution (AA). For the multilingual track of MGTD, we attempted to detect the source language of the texts and then translated them as well as used language adapters to test if further improvements can be achieved. Apart from model and adapter tuning, we also explored another approach. By making use of multiple PLMs, we calculated fixed-length perplexities.

Overall, this thesis attempts to unveil the potential of methods towards the solution of the problems of MGTD and AA, reaching insightful conclusions.

Keywords — Machine Generated Text Detection, Author Attribution, Pretrained Language Models, Large Language Models, Multilingual, Fixed-length Perplexity, Adapter Tuning, Prompt Tuning

Ευχαριστίες

Θα ήθελα να τονίσω ότι το έργο αυτό δεν θα ήταν δυνατό χωρίς την υποστήριξη πολλών ανθρώπων. Θα ήθελα να ευχαριστήσω ιδιαίτερα τον επιβλέποντα μου, κ. Στάμου Γεώργιο, για την πολύτιμη καθοδήγηση του στην εκπόνηση αυτής της διπλωματικής, καθώς και την Μαρία Λυμπεραίου για τη στενή συνεργασία, την υποστήριξη και την καθοριστική συμβολή της στην εργασία αυτή.

Πολύ σημαντική για εμένα ήταν ακόμα η συναισθηματική συνεισφορά της οικογένειας και των φίλων μου που μου παρείχαν αγάπη, στήριξη και γέλιο.

Ανδρεάδης Δημήτριος, Ιούλης 2024

Contents

List of Figures

Chapter 1

Εκτεταμένη Περίληψη στα Ελληνικά

Contents

1.1 Θεωρητικό Υπόβαθρο

Η ανίχνευση κειμένου παραγόμενου από μηχανή αντιμετωπίζεται κυρίως ως πρόβλημα δυαδικής ταξινόμησης (Zellers et al., 2019 [47], Gehrmannetal et al., 2019a [50], Solaiman et al., 2019 [25], Ippolito et al., 2019 [16]). Γενικά , υπάρχουν δύο κύριες προσεγγίσεις: οι επιβλεπόμενες μέθοδοι (Wang et al., 2024a[63], Wang et al., 2024b[62], Uchendu et al., 2021 [4], Zellers et al., 2019 [47], Zhong et al., 2020 [59], Liu et al., 2022 [60]) και οι μη επιβλεπόμενες μέθοδοι, όπως οι μέθοδοι μηδενικής-βολής (Solaiman et al., 2019 [25], Ippolito et al., 2019 [16], Mitchell et al., 2023 [20], Su et al., 2023 [28], Hans et al., 2024 [1]). Ενώ οι επιβλεπόμενες μέθοδοι δίνουν σχετικά καλύτερα αποτελέσματα, είναι επιρρεπείς στην υπερπροσαρμογή (Mitchell et al., 2023 [20], Su et al., 2023 [28]). Ωστόσο, οι μη επιβλεπόμενες μέθοδοι μπορεί να απαιτούν μη ρεαλιστική πρόσβαση λευκού-κουτιού στην γεννήτρια των κειμένων. Το Detect-GPT [20], το οποίο χρησιμοποιεί μόνο λογαριθμικές πιθανότητες υπολογιζόμενες από το μοντέλο GPT-3 [57] και τυχαίες διαταράξεις του κειμένου από το μοντέλο T5 [15] , αποδεικνύεται να έχει μεγαλύτερη ικανότητα μεταξύ των μεθόδων μηδενικής-βολής.

Αυτή η διατριβή περιλαμβάνει ένα σύνολο επιβλεπόμενων μεθόδων για το πρόβλημα της ανίχνευσης κειμένου παραγόμενου από μηχανή και το πρόβλημα της απόδοσης συγγραφέα. Μεταξύ αυτών περιλαμβάνεται η χρήση προ-εκπαιδευμένων γλωσσικών μοντέλων και προσαρμογέων τελευταίας τεχνολογίας, η ανίχνευση και η μετάφραση των πολύγλωσσων κειμένων στα Αγγλικά καθώς και ο υπολογισμός της περιπλοκότητας από πολλά μοντέλα για κάθε κείμενο.

1.2 Προτεινόμενες Πρεσεγγίσεις

1.2.1 Συνεισφορά

Οι συνεισφορές αυτής της διπλωματικής εργασίας είναι πολλαπλές και μπορούν να συνοψιστούν ως εξής:

- Εκπαιδεύσαμε PLMs με στόχο να βελτιστοποιήσουμε τις παραμέτρους τους και να τις προσαρμόσουμε στα δεδομένα του ερωτήματος στόχου. Μεταβάλλαμε τις υπερπαραμέτρους εκπαίδευσης για να δούμε πως επηρεάζεται η μετρική αξιολόγησης των ερωτημάτων MGTD και AA, δηλαδή η ακρίβεια.
- Χρησιμοποιήσαμε προσαρμογείς προκειμένου να μειώσουμε τις παραμέτρους εκπαίδευσης που πρέπει να προσαρμοστούν στα δεδομένα εκπαίδευσης. Συγκεκριμένα, εστιάσαμε στην εκπαίδευση ενός συγκεκριμένου προσαρμογέα, αυτού του συντονισμού προτροπής, καθώς αυτή η τεχνική δίνει τα καλύτερα αποτελέσματα (καλύτερα και από την εκπαίδευση όλων των παραμέτρων) για το ερώτημα MGTD. Είδαμε πως προεκτείνονται αυτές οι τεχνικές για το πιο δύσκολο πρόβλημα της AA.
- Για τα κείμενα της πολύγλωσσης περίπτωσης του MGTD, προσπαθήσαμε να εντοπίσουμε τη γλώσσα-πηγή και να μεταφράσουμε τα κείμενα, μελετώντας έτσι πως η μετάφραση μπορεί να διευκολύνει το ερώτημα αυτό. Παραλάλληλα, χρησιμοποιήσαμε προσαρμογείς γλώσσας και προσαρμογείς εργασίας για να ελέγξουμε εάν μπορούν να επιτευχθούν περαιτέρω βελτιώσεις.
- Τέλος, υπολογίσαμε την περιπλοκότητα συγκεκριμένου μήκους ακολουθίας και για τα δύο υποερωτήματα MGTD και AA. Για να το πετύχουμε αυτό, χωρίσαμε το μήκος της ακολουθίας εισόδου σε κομμάτια για καθένα από τα οποία υπολογίσαμε τη μετρική της περιπλοκότητας, μεταβάλλοντας κάθε φορά το μήκος συμφραζομένων και το βήμα.

1.2.2 Σύνολο Δεδομένων

Παρακάτω, περιγράφουμε τα σύνολα δεδομένων των 3 διαφορετικών ερωτημάτων του διαγωνισμού SemEval2024. Περιλαμβάνονται στατιστικά για τα μεγέθη, τα πεδία και τις γεννήτριες των συνόλων δεδομένων εκπαίδευσης, δοκιμής και αξιολόγησης και για τα 3 ερωτήματα.

1.2.3 Υποερώτημα A: Μονόγλωσση περίπτωση

Δεδομένα: Ο πίνακας 1.1 παρουσιάζει στατιστικά μεταξύ γεννητριών , πεδίων και συνόλων δεδομένων.
Μετρική Αξιολόγησης: Η ακρίβεια χρησιμοποιείται για την αξιολόγηση των μοντέλων.

Σύνολο	Πηγή	davinci-003	ChatGPT	Cohere	Dolly-v2	BLOOMz	GPT-4	Μηχανή	Άνθρωπος
Εκπαίδευση	Wikipedia	3,000	2,995	2,336	2,702	-	-	11,033	14,497
	Wikihow	3,000	3,000	3,000	3,000	-	-	12,000	15,499
	Reddit	3,000	3,000	3,000	3,000	-	-	12,000	15,500
	arXiv	2,999	3,000	3,000	3,000	-	-	11,999	15,498
	PeerRead	2,344	2,344	2,342	2,344	-	-	9,374	2,357
Δοκιμή	Wikipedia	-	-	-	-	500	-	500	500
	Wikihow	-	-	-	-	500	-	500	500
	Reddit	-	-	-	-	500	-	500	500
	arXiv	-	-	-	-	500	-	500	500
	PeerRead	-	-	-	-	500	-	500	500
Αξιολόγηση	Outfox	3,000	3,000	3,000	3,000	3,000	3,000	18,000	16,272

Table 1.1: **Υποερώτημα A: Μονόγλωσση Δυαδική Ταξινόμηση.** Δεδομένα στατιστικών για τα σύνολα Εκπαίδευσης/Δοκιμής/Αξιολόγησης

1.2.4 Υποερώτημα A: Πολύγλωσση περίπτωση

Δεδομένα: Ο πίνακας 1.2 παρουσιάζει τα στατιστικά των συνόλων δεδομένων **Μετρική Αξιολόγησης:** Η ακρίβεια χρησιμοποιείται για την αξιολόγηση των μοντέλων.

Σύνολο	Γλώσσα	davinci-003	ChatGPT	LlaMa2	Jais	Άλλο	Μηχανή	Άνθρωπος
Εκπαίδευση	Αγγλικά	11,999	11,995	-	-	35,036	59,030	62,994
	Κινέζικα	2,964	2,970	-	-	-	5,934	6,000
	Ουρντού	-	2,899	-	-	-	2,899	3,000
	Βουλγάρικα	3,000	3,000	-	-	-	6,000	6,000
	Ινδονησιακά	-	3,000	-	-	-	3,000	3,000
Δοκιμή	Russian	500	500	-	-	-	1,000	1,000
	Αραβικά	-	500	-	-	-	500	500
	Γερμανικά	-	500	-	-	-	500	500
Αξιολόγηση	Αγγλικ΄	3,000	3,000	-	-	9,000	15,000	13,200
	Αραβικά	-	1,000	-	100	-	1,100	1,000
	Γερμανικά	-	3,000	-	-	-	3,000	3,000
	Ιταλικά	-	-	3,000	-	-	3,000	3,000

Table 1.2: **Υποερώτημα A: Πολύγλωσση Δυαδική Ταξινόμηση.** Δεδομένα στατιστικών για τα σύνολα Εκπαίδευσης/Δοκιμής/Αξιολόγησης (Άλλες γεννήτριες είναι Cohere, Dolly-v2 και BLOOMz)

1.2.5 Υποερώτημα B

Δεδομένα: Ο Πίνακας 1.3 παρουσιάζει τον αριθμό κειμένων για κάθε γεννήτρια. **Μετρική Αξιολόγησης:** Η ακρίβεια χρησιμοποιείται για την αξιολόγηση των μοντέλων.

Σύνολο	Πηγή	davinci-003	ChatGPT	Cohere	Dolly-v2	BLOOMz	Άνθρωπος
Εκπαίδευση	Wikipedia	3,000	2,995	2,336	2,702	2,999	3,000
	Wikihow	3,000	3,000	3,000	3,000	3,000	2,995
	Reddit	3,000	3,000	3,000	3,000	2,999	3,000
	arXiv	2,999	3,000	3,000	3,000	3,000	2,998
Δοκιμή	PeerRead	500	500	500	500	500	500
Αξιολόγηση	Outfox	3,000	3,000	3,000	3,000	3,000	3,000

Table 1.3: **Υποερώτημα B: Ανίχνευση Γεννήτριας.** Δεδομένα στατιστικών για τα σύνολα Εκπαίδευσης/Δοκιμής/Αξιολόγησης

1.2.6 Υποερώτημα Γ

Δεδομένα: Ο πίνακας 1.4 παρουσιάζει τα στατιστικά των συνόλων δεδομένων **Μετρική Αξιολόγησης:** Το Μέσο Απόλυτο Σφάλμα (ΜΑΣ) χρησιμοποιείται για την αξιολόγηση της ανίχνευσης του ορίου από τα μοντέλα.

Πεδίο	**Γεννήτρια**	**Εκπαίδευση**	**Δοκιμή**	**Αξιολόγηση**	**Σύνολο**
PeerRead	ChatGPT	3,649 (232)	505 (23)	1,522 (89)	5,676 (344)
	LlaMA-2-7B*	3,649 (5)	505 (0)	1,035 (1)	5,189 (6)
	LlaMA-2-7B	3,649 (227)	505 (24)	1,522 (67)	5,676 (318)
	LlaMA-2-13B	3,649 (192)	505 (24)	1,522 (84)	5,676 (300)
	LlaMA-2-70B	3,649 (240)	505 (21)	1,522 (88)	5,676 (349)
Outfox	GPT-4	-	-	1,000 (10)	1,000 (10)
	LlaMA-2-7B	-	-	1,000 (8)	1,000 (8)
	LlaMA-2-13B	-	-	1,000 (5)	1,000 (5)
	LlaMA-2-70B	-	-	1,000 (19)	1,000 (19)
Σύνολο	Όλα	18,245	2,525	11,123	31,893

Table 1.4: **Υποερώτημα Γ: Ανίχνευση Αλλαγής Σημείου.** Χρησιμοποιούμε γεννήτριες GPT και LLaMA-2 για πεδία όπως αξιολόγηση ακαδημαικών δημοσιεύσεων (PeerRead) και φοιτητικές εκθέσεις (OUTFOX). Ο αριθμός στην παρένθεση "()" είναι ο αριθμός των παραδειγμάτων που παράγονται αποκλειστικά από LLMs, δηλ., ο δείκτης ορίου μεταξύ ανθρώπου και μηχανής ισούται με 0. LLaMA-2-7B* και LLaMA-2-7B χρησιμοποίησαν διαφορετικές προτροπές κατά την παραγωγή κειμένου.

1.2.7 Μέθοδος

Ακολουθήσαμε τις παρακάτω τεχνικές για να εξερευνήσουμε το πρόβλημα MGTD.

Εκπαίδευση προεκπαιδευμένων γλωσσικών μοντέλων

Κάναμε τη βελτιστοποίηση χρησιμοποιώντας την κλάση AutoModelForSequenceClassification για να φορτώσουμε τα διάφορα μοντέλα. Αυτή η κλάση προσθέτει ένα εξωτερικό γραμμικό στρώμα διαστάσεων στοιχεία_εισόδου x (στοιχεία_εξόδου = αριθμός κλάσεων). Οι υπεραπαράμετροι που μεταβάλαμε είναι οι εποχές της εκπαίδευσης βελτιστοποίησης, το μέγιστο μήκος ακολουθίας και το μέγεθος παρτίδας. Το μέγιστο μήκος ακολουθίας είναι το μήκος διακριτικών εισόδου που το μοντέλο δέχεται σαν είσοδο. Τα μοντέλα RoBERTa έχουν ένα μέγιστο μήκος ακολουθίας 512 διακριτικών (δηλαδή μπορούν να χρησιμοποιηθούν με μήκος ακολουθίας μέχρι 512). Τα μοντέλα DeBERTa-V3 δεν έχουν προκαθορισμένο μέγιστο μήκος αολουθίας. Τα μοντέλα GPT-2 έχουν μέγιστο μήκος ακολουθίας 1024. Για όλα τα πειράματα χρησιμοποιήσαμε βαθμό μάθησης 2e-5 and απόσβεση βάρους 0.01. Επίσης, επιλέξαμε να προσαρμόσουμε την συνάρτηση απώλειας προκειμένου να περιέχει τα βάρη των κλάσεων, ενώ διατηρούμε την επιλογή της Απώλειας Διασταυρούμενης Εντροπίας ως συνάρτησης λάθους. Η παραπάνω τροποποίηση περιγράφεται από:

$$p(i) = \frac{-1}{N} \sum_{i \in N} \sum_{j \in C} w_j y_{i,j} log(p_{i,j}) \tag{1.2.1}$$

Τα βάρη για κάθε κλάση i υπολογίζονται με βάση την:

$$w_i = \frac{\#\delta\epsilon\iota\gamma\mu\alpha\tau\omega\nu}{\#\delta\epsilon\iota\gamma\mu\alpha\tau\omega\nu\_\sigma\tau\eta\nu\_\kappa\lambda\alpha\sigma\eta\_i \cdot \#\kappa\lambda\alpha\sigma\epsilon\omega\nu} \tag{1.2.2}$$

Τα γλωσσικά μοντέλα που χρησιμοποιήσαμε στα πειράματα αναφέρονται παρακάτω:

- **roberta-base** με 125 εκατομμύρια παραμέτρους και 9,387,342 κατωφορτώσεις τον Ιούνη του 2024 [1]

[1] https://huggingface.co/FacebookAI/roberta-base

- **roberta-large** με 355 εκατομμύρια παραμέτρους και 10,043,550 κατωφορτώσεις τον Ιούνη του 2024 [2]
- **deberta-v3-base** με 184 εκατομμύρια παραμέτρους (86 εκατομμύρια στο σκελετό + 98 εκατομμύρια παραμετρους στο στρώμα ενσωμάτωσης) και 2,156,043 κατωφορτώσεις τον Ιούνη του 2024 [3]
- **deberta-v3-large** με 435 εκατομμύρια παραμέτρους (304 εκατομμύρια στο σκελετό + 131 εκατομμύρια παραμέτρους στο στρώμα ενσωμάτωσης) και 1,370,872 κατωφορτώσεις τον Ιούνη του 2024 [4]
- **openai-community/gpt2** με 137 εκατομμύρια παραμέτρους [5] και 6,820,036 κατωφορτώσεις τον Ιούνη του 2024
- **openai-community/gpt2-medium** με 355 εκατομμύρια παραμέτρους [6] και 249,991 κατωφορτώσεις τον Ιούνη του 2024

Εκπαίδευση με χρήση προσαρμογέων

Διεξάγαμε εκπαίδευση με χρήση προσαρμογέων χρησιμοποιώντας πάλι την κλάση AutoModelForSequenceClassification για το φόρτωμα των μοντέλων ώστε να διατηρήσουμε την κεφαλή ταξινόμησης που εισάγει η κλάση αυτή. Προσθέσαμε και εκπαιδεύσαμε προσαρμογείς χρησιμοποιώντας τις μεθόδους add_adapter και train_adapter. Αξιολογήσαμε τις αρχιτεκτονικές προσαρμογέων για 6 με 8 εποχές με το μοντέλο roberta-base. Αυτό είναι το μοντέλο που προτείνεται σαν βάση από το SemEval2024. Η επιλογή από μέρους μας αυτού του μοντέλου δικαιολογείται και από το ότι ειναι ελαφρύ και έχει καλή απόδοση σύμφωνα και με την βιβλιογραφία.

Έστω ένα υποσύνολο των παραμέτρων του μοντέλου Θ (σταθερό) και ένα σύνολο παραμέτρων Φ (όπου το Φ μπορεί να εισαχθεί επιπρόσθετα ή να είναι υποσύνολο του Θ, δηλαδή $\Phi \subset \Theta$). Κατά την βελτιστοποίηση, οι προσαρμογείς βελτιστοποιούν μόνο το Φ σύμφωνα με την συνάρτηση απώλειας L στο σύνολο δεδομένων D:

$$\Phi* \leftarrow argmin_{\Phi} L(D; \Theta, \Phi) \tag{1.2.3}$$

Οι αρχιτεκτονικές προσαρμογέων που δοκιμάσαμε είναι [7] :

- **Προσαρμογείς συμφόρησης**

  Οι προσαρμογείς συμφόρησης εισάγουν στρώματα συμφόρησης πρόσθιας τροφοδότησης σε οποιοδήποτε στρώμα. Γενικά, αυτά τα στρώματα αποτελούνται από πίνακες κάτω προβολής W_{down} που προβάλουν τα στρώματα κρυφών καταστάσεων στη χαμηλότερη διάσταση $d_{bottleneck}$, μια μη-γραμμικότητα f, μια πάνω προβολή W_{up} που προβάλει πίσω στην αρχική διάσταση κρυφών καταστάσεων και μια υπολοιπόμενη σύνδεση r:

$$h \leftarrow W_{up} \cdot f(W_{down} \cdot h) + r \tag{1.2.4}$$

  Ο συντελεστής μείωσης ορίζει τον λόγο μεταξύ της διάστασης του κρυφού στρώματος του μοντέλου και της διάστασης συμφόρησης:

$$reduction\_factor = \frac{d_{hidden}}{d_{bottleneck}} \tag{1.2.5}$$

- **Γλωσσικοί προσαρμογείς -Αναστρέψιμοι Προσαρμογείς**

  Η διαμόρφωση MAD-X (Pfeiffer et al., 2020) [29] προτείνει γλωσσικούς προσαρμογείς προκειμένου το μοντέλο να μάθει μετασχηματισμούς που αφορούν μια συγκεκριμένη γλώσσα. Αφού το μοντέλο εκπαιδευτεί σε μια εργασία γλωσσικής μοντελοποίησης, ένας γλωσσικός προσαρμογέας μπορεί να στοιβαχτεί πριν από ένα προσαρμογέα εργασίας που προορίζεται για την εκπαίδευση πάνω σε μια εργασία-στόχο. Έτσι ώστε

[2] https://huggingface.co/FacebookAI/roberta-large
[3] https://huggingface.co/microsoft/deberta-v3-base
[4] https://huggingface.co/microsoft/deberta-v3-large
[5] https://huggingface.co/openai-community/gpt2
[6] https://huggingface.co/openai-community/gpt2-medium
[7] https://docs.adapterhub.ml/methods.html

να επιτύχουμε διαγλωσσική μεταφορά γνώσης μηδενικής βολής, ένας γλωσσικός προσαρμογέας μπορεί να αντικατασταθεί από έναν άλλο. Αρχιτεκτονικά, οι γλωσσικοί προσαρμογείς είναι σε μεγάλο βαθμό παρόμιοι με τους συνήθεις προσαρμογείς συμφόρησης, εκτός από ένα έξτρα αντιστρέψιμο στρώμα μετά το στρώμα ενσωμάτωσης.

- **Συντονισμός Προθέματος**

  Ο συντονισμός προθέματος (Li and Liang, 2021) [36] εισάγει νέες παράμετρους στα μπλοκ προσοχής πολλαπλών κεφαλών σε κάθε στρώμα του μετασχηματιστή. Πιο συγκεκριμένα, προσαρτά εκπαιδεύσιμα διανύσματα προθέματος P^K και P^V στα κλειδιά και τις τιμές της εισόδου κάθε κεφαλής προσοχής. Η κάθε κεφαλή έχει ένα διαμορφώσιμο μήκος προθέματος (μήκος_προθέματος):

$$\kappa\varepsilon\varphi\alpha\lambda\eta\_i = \Pi\rho o\sigma o\chi\eta(QW_i^Q, [P_i^K, KW_i^K], [P_i^V, VW_i^V]) \tag{1.2.6}$$

- **Συμπυκνωτής**

  Η αρχιτεκτονική του Συμπυκνωτή προτάθηκε από Mahabadi et al., 2021 [44]. Είναι παρόμοια με την αρχιτεκτονική των προσαρμογέων συμφόρησης. Η μόνη διαφορά είναι ότι αντικαθιστά τις γραμμικές πάνω- και κάτω- προβολές με ένα στρώμα PHM. Σε αντίθεση με το γραμμικό στρώμα, το στρώμα PHM κατασκευάζει τον πίνακα βαρών του από δύο μικρότερους πίνακες, το οποίο μειώνει τον αριθμό των παραμέτρων. Αυτοί οι πίνακες μπορούν να παραγοντοποιηθούν και να μοιραστούν μεταξύ όλων των στρωμάτων του προσαρμογέα.

- **Προσαρμογή Χαμηλού Βαθμού**

  Η προσαρμογή χαμηλού βαθμού είναι μια αποδοτική μέθοδος συντονισμού που προτάθηκε από τον Hu et al. (2021) [19]. Η προσαρμογή χαμηλού βαθμού εισάγει εκπαιδεύσιμους χαμηλού βαθμού πίνακες αποσύνθεσης μέσα στα στρώματα ενός προεκπαιδευμένου μοντέλου. Για οποιοδήποτε στρώμα του μοντέλου εκφρασμένου ως πολλαπλασιασμός πινάκων, ο πίνακας βαρών $W_o \subset R^{dxk}$ μεταβάλλεται σε $W_o + \Delta W = W_o + BA$ όπου B$\subset R^{d\times r}$ και A$\subset R^{r\times k}$. Η επαναπαραμετροποίηση γίνεται ως εξής:

$$h = W_0 x + \frac{a}{r} BAx \tag{1.2.7}$$

- $(IA)^3$

  Ο προσαρμογέας έγχυσης με αναστολή και ενίσχυση των εσωτερικών ενεργοποιήσεων $(IA)^3$ είναι μια αποτελεσματική μέθοδος συντονισμού που προτάθηκε από Liu et al. (2022) [60]. $(IA)^3$ εισάγει εκπαιδεύσιμα διανύσματα σε διαφορετικές συνιστώσες του μοντέλου Μετασχηματιστή, πραγματοποιώντας στοιχείο προς στοιχείο επανακλιμάκωση των εσωτερικών ενεργοποιήσεων. Για οποιοδήποτε στρώμα μοντέλου εκφρασμένο ως πολλαπλασιασμός πινάκων, ο στοιχείο προς στοιχείο πολλαπλασιασμός περιγράφεται από:

$$h = l_W \otimes Wx \tag{1.2.8}$$

  όπου, $\otimes$ δηλώνει τον στοιχείο προς στοιχείο πολλαπλασιασμό .

- **Συντονισμός προτροπής**

  Ο συντονισμός προτροπής είναι μια αποδοτική τεχνική συντονισμού βελτιστοποίησης που προτάθηκε από Lester et al. (2021) [11]. Ο συντονισμός προτροπής προσθέτει συντονίσιμα διακριτικά, τα οποία αποκαλούνται μαλακές προτροπές, και τα οποία προσαρτούνται στο κείμενο εισόδου. Πρώτα, η ακολουθία εισόδου $x_1, x_2, \cdots x_n$ ενσωματώνεται, καταλήγοντας στον πίνακα $X_e \subset R^{nxe}$ όπου e είναι η διάσταση τυο χώρου ενσωμάτωσης. Οι μαλακές προτροπές με μήκος p αναπαριστώνται ως $P_e \subset R^{p\times e}$. Τα P_e και X_e συνενώνονται, σχηματίζοντας την είσοδο του ακόλουθου κωδικοποιητή ή αποκωδικοποιητή:

$$[P_e; X_e] \subset R^{(p+n)\times e} \tag{1.2.9}$$

  Η κλάση PromptTuningConfig δέχεται τις ακόλουθες παράμετρους:

– μήκος_προτροπής: για να τεθεί το μήκος της μαλακής προτροπής p

– αρχικοποιήση_προτροπής: για να τεθεί η μέθοδος αρχικοποίησης βαρών, που είναι είτε "τυχαία_ομοιόμορφη" είτε "από_κείμενο" για να αρχικοποίησει τα διακριτικά της προτοπής με μια ενσωμάτωση από το λεξιλόγιο του μοντέλου.

– κείμενο_αρχικοποίησης_προτροπής ως το κείμενο που θα χρησιμοποιηθεί έαν θέσουμε ως μέθοδο αρχικοποίησης αρχικοποίηση_προτροπής="από_κείμενο"

1.2.8 Ανίχνευση Γλώσσας και Μετάφραση

Για την πολύγλωσση περίπτωση του υποερωτήματος Α, ήρθαμε αντιμέτωποι με κείμενα διαφορετικών γλωσσών. Πρόσφατες μελέτες (Hu et al., 2020 [30]) υποδεικνύουν ότι τα μοντέλα τελευταίας τεχνολογίας όπως τα μοντέλα XLM-roBERTa, έχουν κακή απόδοση στη διαγλωσσική μεταφορά γνώσης μεταξύ πολλών ζευγαριών γλωσσών. Ο κύριος λόγος πίσω από την κακή απόδοση φέρεται να είναι η τωρινή έλλειψη ικανότητας των μοντέλων να αναπαριστήσουν όλες τις γλώσσες ισοδύναμα. Έτσι, μια λύση για την πολύγλωσση περίπτωση του υποερωτήματος Α είναι να ελέγξουμε υπάρχοντες γλωσσικούς προσαρμογείς και προσαρμογείς γλώσσας για να αξιολογήσουμε σε ποιο βαθμό καταφέρνουν να επιτύχουν διαγλωσσική μεταφορά γνώσης. Μια άλλη προσέγγιση που δοκιμάσαμε είναι η μετάφραση όλων των κειμένων στα Αγγλικά προκειμένου στη συνέχεια να χρησιμοποιηθούν με μονόγλωσσα μοντέλα.

Ακολουθώντας την πρώτη προσέγγιση, ένας προσαρμογέας κάποιας συγκεκριμένης γλώσσας εκπαιδευμένος σε μια γλώσσα προέλευσης μπορεί να αντικατασταθεί με ένα προσαρμογέα εκπαιδευμένο στη γλώσσα ενδιαφέροντος. Επίσης, μπορούμε να εισάγουμε επιπλέον προσαρμογείς εργασίας για να συμπεριλάβει γνώση σχετική με την εργασία.

Το σύνολο αξιολόγησης δε περιείχε όμως ετικέτες που να υποδεικνύουν τη γλώσσα των κειμένων, οπότε αρχικά χρειάστηκε να ανιχνεύσουμε τη γλώσσα των κειμένων. Αυτή η εργασία διευκολύνθηκε από το γεγονός ότι είχαμε να επιλέξουμε μεταξύ τεσσάρων γλωσσών από το σύνολο αξιολόγησης (Αγγλικά, Ιταλικά, Αραβικά, Γερμανικά). Για την ανίχνευση, επιλέξαμε να χρησιμοποιήσουμε το μοντέλο "facebook/fasttext-language-identification" [43] από το αποθετήριο HuggingFace. Αυτή συνιστά την κυρίαρχη επιλογή για ανίχνευση γλώσσας στο αποθετήριο με 52,630,391 κατοφορτώσεις τον Ιούνη του 2024. Για να ελέγξουμε την ακρίβεια του, ελέγχουμε πόσο ακριβής είναι στην ανίχνευση γλώσσας στο σύνολο εκπαίδευσης και αξιολόγησης που ήδη είχαν πεδίο με την γλώσσα του κειμένου. Παρακάτω, παρουσιάζουμε τα λάθη για κάθε γλώσσα και για τα δύο σύνολα:

Language and Dataset Split	Errors	Percentage
English Train	289 out of 122,024	0.00237
Chinese Train	222 out of 11,934	0.01860
Urdu Train	19 out of 5,899	0.00322
Bulgarian Train	121 out of 12,000	0.01008
Indonesian Train	20 out of 6,000	0.00333
Russian Valid	16 out of 2,000	0.008
German Valid	0 out of 1,000	0
Arabic Valid	2 out of 1,000	0.002

Table 1.5: Language identification errors on train and valid splits using facebook/fasttext-language-identification

Στη συνέχεια, για την μετάφραση ακολουθήσαμε την στρατηγική μεταφρασης ανά πρόταση για να έχουμε καλύτερα αποτελέσματα. Έχοντας ήδη δημιουργήσει το πεδίο γλώσσας των κειμένων και για το σύνολο αξιολόγησης χρησιμοποιήσαμε τα μοντέλα της Ομάδας Έρευνας των Γλωσσικών Τεχνολογιών του Ελσίνκι, καθένα από τα οποία αφορά ένα ζεύγος γλωσσών:

- Helsinki-NLP/opus-mt-de-en [8]
- Helsinki-NLP/opus-mt-it-en [9]
- Helsinki-NLP/opus-mt-ar-en [10]

Η μετάφραση με αυτά τα μοντέλα έδωσε πολύ καλά αποτελέσματα. Σε κάποιες περιπτώσεις και συγκεκριμένα στα Αραβικά κείμενα, έπρεπε να περικόψουμε την κάθε πρόταση στους 1,200 χαρακτήρες. Εφόσον, το ερώτημα που καλούμαστε να αντιμετωπίσουμε δεν είναι σημασιολογικό πιστεύουμε ότι αυτή η επιλογή δε θα έχει κάποια επίδραση στο αποτέλεσμα.

1.2.9 Περιπλοκότητα

Η περιπλοκότητα μιας διακριτής κατανομής πιθανότητας p είναι μια έννοια που χρησιμοποιείται ευρέως στην θεωρία πληροφορίας, την μηχανική μάθηση και την στατιστική μοντελοποίηση. Ορίζεται ως:

$$PP(p) := 2^{H(p)} = 2^{-\sum_x p(x) log_2 p(x)} \tag{1.2.10}$$

Ένα μοντέλο μιας άγνωστης κατανομής πιθανότητας p, μπορεί να προταθεί με βάση ένα δείγμα εκπαίδευσης το οποίο αντλείται από την κατανομή p. Δεδομένου ενός προτεινόμενου πιθανοτικού μοντέλου q, κάποιος μπορεί να αξιολογήσει το q με βάση το πόσο καλά προβλέπει ένα ξεχωριστό δείγμα αξιολόγησης $x_1, x_2, ..., x_t$ το οποίο επίσης αντλείται από το p. Η περιπλοκότητα του μοντέλου q ορίζεται ως:

$$PP(q) := 2^{-\frac{1}{t}\sum_{i=1}^{t} log_2 q_\theta(x_i|x<i)} \tag{1.2.11}$$

Ο εκθέτης της παραπάνω σχέσης μπορεί να ερμηνευθεί ως η διασταυρούμενη εντροπία, όπου $p(x) = \frac{1}{N}$ είναι η εμπειρική κατανομή του δείγματος αξιολόγησης. Εάν θεωρήσουμε τα $x_1, x_2, ..., x_t$ να είναι μια ακολουθία διακριτικών τότε το $log_2 p_\theta(x_i|x < i)$ είναι η δεσμευμένη log-πιθανότητα του ι-οστού διακριτικού δεδομένων των προηγούμενων διακριτικών. Η διαδικασία μετατροπής του κειμένου σε διακριτικά έχει άμεση επίδραση στην περιπλοκή του μοντέλου, γεγονός που πρέπει να ληφθεί υπόψη όταν συγκρίνουμε διαφορετικά μοντέλα.

Εάν δε περιοριζόμασταν από το μήκος συμφραζόμενων του μοντέλου, θα υπολογίζαμε την περιπλοκότητα του μοντέλου αυτοπαλινδρομικά, λαμβάνοντας υπόψη όλη την ακολουθία διακριτικών που προηγείται. Αντίθετα, η ακολουθία διασπάται τυπικά σε υποακολουθίες ίσες με το μέγιστο μέγεθος εισόδου του μοντέλου. Εάν το μέγιστο μέγεθος εισόδου του μοντέλου είναι k, τότε προσεγγίζουμε την πιθανότητα ενός διακριτικού με δέσμευση ως προς τα προηγούμενα k-1 διακριτικά παρά σε όλα τα συμφραζόμενα που προηγούνται.

Όταν υπολογίζουμε την περιπλοκότητα ενός μοντέλου για μία ακολουθία, μια δελεαστική αλλά υποβέλτιστη προσέγγιση είναι να διασπάσουμε την ακολουθία σε ξεχωριστά κομμάτια και να προσθέσουμε τις αποσυντεθειμένες λογαριθμικές πιθανότητες του κάθε τμήματος ανεξάρτητα. Αυτή τείνει να είναι μια κακή προσέγγιση καθώς το μοντέλο θα έχει λιγότερα συμφραζόμενα στα περισσότερα βήματα της πρόβλεψης. Αντίθετα, η περιπλοκότητα ενός μοντέλου σταθερού μήκους συμφραζομένων πρέπει να υπολογιστεί με μία στρατηγική κυλιόμενου παραθύρου. Αυτό συνεπάγεται να ολισθήσουμε επαναλαμβανόμενα το παράθυρο συμφραζομένων έτσι ώστε το μοντέλο να έχει περισσότερα συμφραζόμενα όταν κάνει πρόβλεψη. Το μειονέκτημα είναι ότι αυτό απαιτεί ένα ξεχωριστό πρόσθιο πέρασμα για κάθε διακριτικό. Ένας καλός συμβιβασμός στην πράξη είναι να ολισθήσει κανείς το παράθυρο κατά έναν αριθμό διακριτικών παρά κατά ένα διακριτικό την φορά. Αυτό επιτρέπει τον υπολογισμό να συνεχίσει αρκετά πιο γρήγορα δίνοντας παράλληλα στο μοντέλο ένα μεγάλο αριθμό από συμφραζόμενα διακριτικά για να κάνει πρόβλεψη σε κάθε βήμα. Εάν όμως το εκτελέσουμε αυτό με ένα βήμα ίσο με το μέγιστο μήκος εισόδου, τότε η προσέγγιση αυτή εκφυλίζεται στην υποβέλτιστη στρατηγική χωρίς την ολίσθηση παραθύρου, που συζητήθηκε προηγουμένως. Όσο μικρότερο το βήμα, τόσα περισσότερα συμφραζόμενα θα έχει το μοντέλο για να κάνει την κάθε πρόβλεψη και άρα τόσο καλύτερος ο υπολογισμός της περιπλοκής θα είναι.

Για κάθε τμήμα, η μέση αρνητική λογαριθμική πιθανότητα για κάθε τμήμα επιστρέφεται ως απώλεια με βαση την γραμμή κώδικα: chunk_loss = model(chunk_input_ids, labels=chunk_input_ids).loss

[8] https://huggingface.co/Helsinki-NLP/opus-mt-de-en
[9] https://huggingface.co/Helsinki-NLP/opus-mt-it-en
[10] https://huggingface.co/Helsinki-NLP/opus-mt-ar-en

Στην προσέγγιση μας εξετάζουμε πως η επιλογή του μοντέλου, του βήματος και του μήκους των συμφραζομένων επηρεάζει την ακρίβεια και την ανάκληση.

1.3 Πειραματικό Μέρος

Στην ενότητα αυτή παρουσιάζουμε τα αποτελέσματα εκτεταμένων πειραμάτων που διεξαγάγαμε για τα υποπροβλήματα MGTD και AA. Για την αξιολόγηση των αποτελεσμάτων βασιζόμαστε στις ακόλουθες μετρικές: **Ακρίβεια (Accuracy)**, **Ακρίβεια (Precision)** και **Ανάκληση (Recall)** που ορίζονται ως ακολούθως:

$$\textit{Ακρίβεια (Accuracy)} = \frac{\text{αριθμός σωστών προβλέψεων}}{\text{συνολικός αριθμός προβλέψεων}} \tag{1.3.1}$$

$$\textit{Ακρίβεια (Precision)} = \frac{\text{Σχετικά δείγματα που ανακαλούνται}}{\text{Συνολικός αριθμός δειγμάτων που ανακαλούνται}} \tag{1.3.2}$$

$$\textit{Ανάκληση (Recall)} = \frac{\text{Σχετικά δείγματα που ανακαλούνται}}{\text{Σύνολο σχετικών δειγμάτων}} \tag{1.3.3}$$

1.3.1 Υποερώτημα Α: Μονόγλωσση Δυαδική Ταξινόμηση

Στον Πίνακα 1.6 και παρουσιάζουμε τα αποτελέσματα που προκύπτουν από την σύγκριση των προσαρμογέων. Χρησιμοποιήσαμε το μοντέλο roberta-base, μήκος ακολουθίας 512, ρυθμό μάθησης 2e-5, απόσβεση βάρους 0.01 και 8 εποχές εκπαίδευσης (εκτός εάν αναφέρεται άλλος αριθμός). Καταλήξαμε στο ενδιαφέρον αποτέλεσμα ότι ο συντονισμός προτροπής (prompt tuning) δίνει ακρίβεια σχεδόν 94% για prompt_length 50.

Προσαρμογέας και Υπερπαράμετροι	**Κλάση 0**		**Κλάση 1**		**Συνολική Ακρίβεια**
	Ακρίβεια	**Ανάκληση**	**Ακρίβεια**	**Ανάκληση**	
Χωρίς adapter config, 6 εποχές	0.8303	0.812	0.8334	0.8499	0.8319
DoubleSeqBnConfig	0.8964	0.4212	0.6463	0.956	0.7021
SeqBnConfig	0.9336	0.5453	0.7013	0.9649	0.7657
ParBnConfig	0.9056	0.7360	0.7959	0.9307	0.8382
SeqBnInvConfig	0.9363	0.5532	0.7052	0.966	0.77
PrefixTuningConfig (flat=False, prefix_length=10)	0.8504	0.7274	0.7821	0.8843	0.8098
PrefixTuningConfig (flat=False, prefix_length=50)	0.9042	0.8287	0.8560	0.9206	0.8770
PrefixTuningConfig (flat=False, prefix_length=100)	0.8714	0.6157	0.7254	0.9179	0.7744
PrefixTuningConfig (flat=False, prefix_length=200), 7 εποχές	0.8522	0.6214	0.7251	0.9025	0.7691
LoRAConfig(r=8, alpha=16)	0.9219	0.6256	0.7377	0.9521	0.7971
LoRAConfig(r=8, alpha=16)	0.9219	0.6256	0.7377	0.9521	0.7971
CompacterConfig	0.8880	0.5656	0.7044	0.9355	0.7599
IA3Config με merge adapter	0.9360	0.7241	0.7929	0.9553	0.8453
IA3Config χωρίς merge adapter	0.9420	0.6941	0.7766	0.9614	0.8345
PromptTuningConfig(prompt_length=20), 7 εποχές	0.9949	0.7542	0.8177	0.9965	0.8814
PromptTuningConfig(prompt_length=50), 7 εποχές	0.9510	0.9168	0.9271	0.9573	0.9381
PromptTuningConfig(prompt_length=100), 6 εποχές	0.9916	0.8431	0.8751	0.9936	0.9221

Table 1.6: **Υποερώτημα Α: Μονόγλωσση Δυαδική Ταξινόμηση.** Εκπαίδευση Προσαρμογέα

Στη συνέχεια, πειραματιζόμαστε με το κείμενο αρχικοποίησης της προτροπής συντονισμού προκειμένου να δούμε αν μπορούμε να επιτύχουμε ακόμα καλύτερα αποτελέσματα, πάλι με το μοντέλο roberta-base, χρησιμοποιώντας την διαμόρφωση config = PromptTuningConfig(prompt_length=50, prompt_init = "from_string", prompt_init_text = Text). Δε διαπιστώνεται περαιτέρω βελτίωση για το πλήθος κειμένων που δοκιμάσαμε όπως φαίνεται στον Πίνακα 1.7.

Κείμενο	**Κλάση 0**		**Κλάση 1**		**Συνολική Ακρίβεια**
	Ακρίβεια	**Ανάκληση**	**Ακρίβεια**	**Ανάκληση**	
"Question: Is the text generated by human or machines. The machines used for generation are davinci-003, ChatGPT, Cohere, Dolly-v2, BLOOMz and GPT-4. For human-generated text choose 0, else for machine-generated text choose 1." , σύνολο εκπαίδευσης	0.9631	0.8238	0.8591	0.9715	0.9014
"Question: Is the text generated by human or machines. The machines used for generation are davinci-003, ChatGPT, Cohere, Dolly-v2, BLOOMz and GPT-4. For human-generated text choose 0, else for machine-generated text choose 1." , σύνολο εκπαίδευσης + σύνολο δοκιμής	0.9931	0.7833	0.8355	0.9951	0.8945
"Question: Is the text generated by human or machines? For human-generated text choose 0, else for machine-generated text choose 1.") , σύνολο εκπαίδευσης	0.9289	0.9078	0.9183	0.9372	0.9232
"Question: Is the text generated by human or machines? For human-generated text choose 0, else for machine-generated text choose 1." , σύνολο εκπαίδευσης + σύνολο δοκιμής	0.9884	0.8464	0.8771	0.991	0.9224
"For human-generated text choose 0, else for machine-generated text choose 1." , σύνολο εκπαίδευσης	0.9385	0.8409	0.8685	0.9502	0.8983
"For human-generated text choose 0, else for machine-generated text choose 1." , σύνολο εκπαίδευσης + σύνολο δοκιμής	0.9857	0.7522	0.8155	0.9902	0.8772
"""Question: Is the text generated by human or language models? Context: For human-generated text choose 0, else for machine-generated text choose 1. The language models used for generation are davinci-003, ChatGPT, Cohere, Dolly-v2, BLOOMz and GPT-4. Machine-generated text tends to be misclassified as human-generated""", σύνολο εκπαίδευσης	0.9417	0.8456	0.8722	0.9526	0.9018
"""Question: Is the text generated by human or language models? Context: For human-generated text choose 0, else for machine-generated text choose 1. The language models used for generation are davinci-003, ChatGPT, Cohere, Dolly-v2, BLOOMz and GPT-4. Machine-generated text tends to be misclassified as human-generated""", σύνολο εκπαίδευσης	0.9459	0.8712	0.8913	0.9549	0.9152
"Question: Is the text generated by human or machine? 0: ḧuman, 1: m̈achine", σύνολο εκπαίδευσης	0.9353	0.87	0.8895	0.9455	0.9097

Table 1.7: **Υποερώτημα Α: Μονόγλωσση Δυαδική Ταξινόμηση.** Εκπαίδευση συντονισμού προτροπής μεταβάλλοντας το κείμενο αρχικοποίησης

Σε μια προσπάθεια να συνδυάσουμε τα παραπάνω ευρήματα, κάναμε εκτενή πειράματα εκπαίδευσης χρησιμοποιώντας πάλι ρυθμό μάθησης 2e-5 και απόσβεση βάρους 0.01. Όταν δεν διευκρινίζεται, η εκπαίδευση ήταν για 1 εποχή και χρησιμοποιήθηκε μόνο το σύνολο εκπαίδευσης για την εκπαίδευση. Τα μοντέλα που δοκιμάστηκαν (μαζί με τους αντίστοιχους χρόνους εκπαίδευσης που απαιτούν ανά εποχή) είναι τα: roberta-base (01:25h), roberta-large (5:10h), deberta-v3-base (2:15h), deberta-v3-large (7:30h), gpt2 (1:40h) and gpt2-medium (5:40h).

Μοντέλο	**Υπερπαράμετροι**	Class 0		Class 1		**Συνολική ακρίβεια**
		Ακρίβεια	**Ανάκληση**	**Ακρίβεια**	**Ανάκληση**	
roberta-base	max_len =512, bs=16,	0.8499	0.5688	0.6999	0.9092	0.7475
roberta-base	max_len=512, bs=16, PromptTuningConfig (prompt_length=50)	0.9272	0.9385	0.9438	0.9334	0.9358
roberta-base	max_len=512, bs=16, PromptTuningConfig (prompt_length=50), 7 εποχές	0.9499	0.9111	0.9225	0.9566	0.9350
roberta-base	max_len=512, bs=16, 8 εποχές, σύνολο εκπαίδευσης + σύνολο δοκιμής	0.9979	0.7733	0.8297	0.9985	0.8916
roberta-base	max_len=512, bs=16, 6 εποχές	0.8151	0.7237	0.7732	0.8516	0.7909
roberta-base	max_len=512, bs=16, PromptTuningConfig (prompt_length=50), 7 εποχές, σύνολο εκπαίδευσης + σύνολο δοκιμής	0.9950	0.7679	0.8261	0.9965	0.8880
roberta-base	max_len =256, bs=32	0.9147	0.8215	0.8523	0.9308	0.8789
roberta-base	max_len=256, bs=32, 3 εποχές	0.8728	0.8085	0.8376	0.8934	0.8531
roberta-base	max_len=256, bs=32, 7 εποχές,	0.8768	0.8641	0.8787	0.8903	0.8778
roberta-base	max_len =256, bs=16	0.8780	0.8575	0.8738	0.8923	0.8758
roberta-base	max_len=256, bs=32, PromptTuningConfig (prompt_length=50)	0.9228	0.9341	0.9398	0.9294	0.9316
roberta-base	max_len=256, bs=16, PromptTuningConfig (prompt_length=50), 7 εποχές	0.9337	0.9267	0.9342	0.9405	0.9340
roberta-base	max_len =128, bs=16	0.8702	0.8496	0.8669	0.8854	0.8684
roberta-base	max_len =128, bs=32	0.9063	0.8754	0.8907	0.9182	0.8979
roberta-base	max_len =128, bs=64	0.8917	0.8697	0.8848	0.9046	0.8880
roberta-base	max_len=128, bs=32, 7 εποχές	0.9000	0.8172	0.8474	0.9179	0.8701
roberta-base	max_len =64, bs=128	0.8997	0.6419	0.7429	0.9353	0.7960
roberta-large	max_len =512, bs=4	0.6618	0.2504	0.5662	0.8843	0.5833
roberta-large	max_len =256, bs=16	0.8778	0.8061	0.8368	0.8986	0.8547
roberta-large	max_len=256, bs=32, PromptTuningConfig (prompt_length=50)	0.8784	0.7435	0.7964	0.9069	0.8293
roberta-large	max_len =128, bs=32	0.9133	0.8380	0.8637	0.9281	0.8853
roberta-large	max_len =64, bs=64	0.8750	0.6731	0.7555	0.9131	0.7991
deberta-v3-base	max_len =1024, bs=8, peft	0.994	0.4991	0.6877	0.9973	0.7607
deberta-v3-base	max_len =1024, bs=4	0.9536	0.3419	0.6234	0.9849	0.6797

deberta-v3-base	max_len =1024, bs=4, peft	0.9854	0.4285	0.6581	0.9943	0.7256
deberta-v3-base	max_len =512, bs=8	0.8402	0.7303	0.7820	0.8744	0.8060
deberta-v3-base	max_len =512, bs=8, peft	0.9903	0.3144	0.6167	0.9972	0.6730
deberta-v3-large	max_len =512, bs=4, peft	0.9292	0.1541	0.5641	0.9894	0.5928
gpt2	max_len =1024, bs=4	0.8	0.7151	0.765	0.8384	0.7798
gpt2	max_len =512, bs=16, peft (c_attn, c_proj)	0.8627	0.4473	0.6519	0.9357	0.7038
gpt2	max_len =512, bs=8	0.8234	0.7769	0.8081	0.8493	0.8149
gpt2	max_len =512, bs=4	0.8268	0.7148	0.7703	0.8646	0.7935
gpt2	max_len =512, bs=2	0.8203	0.7555	0.7937	0.8504	0.8054
gpt2	max_len =256, bs=16	0.8863	0.8198	0.8475	0.9049	0.8645
gpt2	max_len=256, bs=16, PromptTuningConfig (prompt_length=50)	0.7255	0.8013	0.8016	0.7259	0.7617
gpt2	max_len =128, bs=64	0.8919	0.7577	0.8072	0.917	0.8414
gpt2	max_len =128, bs=32	0.9028	0.7421	0.7992	0.9278	0.8396
gpt2-medium	max_len =1024, bs=1,	0.7947	0.7128	0.7625	0.8335	0.7762
gpt2-medium	max_len =512, bs=4, peft(c_attn, c_proj)	0.872	0.6532	0.7445	0.9133	0.7898
gpt2-medium	max_len =512, bs=4	0.8064	0.6257	0.7186	0.8641	0.7510
gpt2-medium	max_len =512, bs=2, peft(c_attn, c_proj)	0.7804	0.4352	0.6352	0.8893	0.6737

Table 1.8: **Υποερώτημα Α: Μονόγλωσση Δυαδική Ταξινόμηση.** Εκπαίδευση

Όπως παρατηρούμε στον Πίνακα 1.10, η ακρίβεια των μοντέλων δεν αυξάνεται πέρα από την τιμή που επιτυγχάνεται κατά την 1η εποχή. Επίσης οι μεγαλύτερες εκδόσεις δεν αποδίδουν καλύτερα αποτελέσματα εξαιτίας και του περιορισμένου διαθέσιμου υλικού. Μια καλή τακτική φαίνεται να είναι η μείωση του μήκους ακολουθίας σε 128, μια αλλαγή που δίνει καλύτερα αποτελέσματα σε σημαντικά λιγότερο χρόνο. Η ακρίβεια που επιτυγχάνεται είναι γύρω στο 0.9. Η προτροπή συντονισμού φαίνεται να δίνει μια ακρίβεια γύρω στο 0.94 με μία μόνο εποχή εκπαίδευσης.

1.3.2 Υποερώτημα Α: Πολύγλωσση Δυαδική Ταξινόμηση

Σε αυτό το σημείο διεξάγαμε πειράματα για την αντιμετώπιση κειμένων διαφορετικών από τα Αγγλικά. Όπως παρατηρούμε από τον πίνακα 1.9 ότι το xlm-roberta-base έχει μεγάλη ακρίβεια για τα Γερμανικά (0.8714), τα Ιταλικά (0.8305) και τα Αραβικά (0.8716) ενώ αξιοσημείωτα χαμηλή για τα Αγγλικά (0.7059). Από την άλλη, το roberta-base έχει υψηλότερη απόδοση για τα Αγγλικά (0.7899) αλλά πολύ χαμηλή για τις άλλες γλώσσες. Επομένως, τα μοντέλα αυτά πρέπει να συνδυαστούν προκειμένου να έχουμε υψηλότερη ακρίβεια αφού τα Αγγλικά κείμενα αποτελούν το μεγαλύτερο μέρος του συνόλου αξιολόγησης.

Η μετάφραση των πολύγλωσσων κειμένων και η χρήση του roberta-base για τα μεταφρασμένα κείμενα δε δίνει καλά αποτελέσματα. Η χρήση γλωσσικών προσαρμογέων και προσαρμογέων εργασίας επίσης δε φαίνεται να έχει κάποια συμβολή.

1.3.3 Υποερώτημα Β

Για όλα τα πειράματα εκπαίδευσης χρησιμοποιήσαμε ρυθμό μάθησης 2e-5, μέγεθος παρτίδας 16 και μήκος ακολουθίας 512 (εκτός άμα διευκρινίζεται διαφορετικά)

Η πρώτη ιδέα ήταν η επαύξηση του υποερωτήματος Β με δείγματα από το υποερώτημα Α. Αυτό όμως δε καταλήγει σε καλύτερα αποτελέσματα. Ένας λόγος για αυτό είναι ότι η επαύξηση προκαλεί ανισσοροπία στο σύνολο δεδομένων του Β, η οποία ανισσοροπία δε φαίνεται να αντισταθμίζεται με την χρήση βαρών. Σε επόμενο πείραμα

Γλώσσα	**Κλάση 0**		**Κλάση 1**	
	Ακρίβεια	**Ανάκληση**	**Ακρίβεια**	**Ανάκληση**
xlm-roberta-base, 4 εποχές, πολύγλωσσο σύνολο εκπαίδευσης και πολύγλωσσο σύνολο αξιολόγησης				
Αγγλικά	1	0.3718	0.6440	1
Γερμανικά	0.9857	0.7557	0.8019	0.9890
Ιταλικά	1	0.6608	0.7468	1
Αραβικά	0.9959	0.7333	0.8045	0.9973
roberta-base, 4 εποχές, πολύγλωσσο σύνολο εκπαίδευσης και πολύγλωσσο σύνολο αξιολόγησης				
Αγγλικά	0.9999	0.5512	0.7169	0.9999
Γερμανικά	0.2850	0.0183	0.4929	0.9540
Ιταλικά	1	0.4554	0.6475	1
Αραβικά	0.1143	0.008	0.5116	0.9437
roberta-base, 5 εποχές, μονόγλωσσο σύνολο εκπαίδευσης και μεταφρασμένο πολύγλωσσο σύνολο αξιολόγησης				
Αγγλικά	0.8193	0.7978	0.8261	0.8451
Γερμανικά	0.9249	0.558	0.6836	0.9547
Ιταλικά	0.9989	0.6236	0.7265	0.9993
Αραβικά	0.8965	0.026	0.5299	0.9973
xlm-roberta-base, 4 εποχές, πολύγλωσσο σύνολο εκπαίδευσης (με και χωρίς la) πολύγλωσσο σύνολο αξιολόγησης (la)				
Αγγλικά	1	0.3827	0.648	1
Γερμανικά	0.9816	0.728	0.7839	0.9863
Ιταλικά	1	0.8136	0.8430	1
Αραβικά	0.9957	0.6903	0.78	0.9973
xlm-roberta-base, 5 εποχές, μονόγλωσσο σύνολο εκπαίδευσης (la) και πολύγλωσσο σύνολο αξιολόγησης (la)				
Αγγλικά	0.928	0.4081	0.6511	0.9721
Γερμανικά	0.9941	0.2807	0.5813	0.9983
Ιταλικά	1	0.3856	0.6195	1
Αραβικά	0.9966	0.2927	0.6086	0.9991
xlm-roberta-base, 5 εποχές, πολύγλωσσο σύνολο εκπαίδευσης (la+ta) και πολύγλωσσο test (la+ta)				
Αγγλικά	0.9658	0.5198	0.6995	0.9838
Γερμανικά	0.9735	0.6233	0.7230	0.9830
Ιταλικά	0.9990	0.9444	0.9473	0.9990
Αραβικά	0.9816	0.4256	0.6555	0.9927
xmod-base, 3 εποχές, πολύγλωσσο σύνολο εκπαίδευσης (la) και πολύγλωσσο σύνολο αξιολόγησης (la), bs=8				
Αγγλικά	0.9996	0.3687	0.6428	0.9999
Γερμανικά	0.6946	0.9986	0.9976	0.5611
Ιταλικά	0.9955	0.8749	0.8884	0.9961
Αραβικά	0.8834	0.3786	0.6284	0.9546

Table 1.9: **Υποερώτημα Α: Πολύγλωσση Δυαδική Ταξινόμηση.** Γλωσσικοί προσαρμογείς και προσαρμογείς εργασίας

Κλάση 0		Κλάση 1		Κλάση 2		Κλάση 3		Κλάση 4		Κλάση 5		Συνολική ακρίβεια
P	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	
roberta-base, 5 εποχές, προσαύξηση (A + B σύνολα δεδομένων)												
0.9986	0.939	0.6529	1	0.9923	0.6503	0.7450	0.701	0.9539	0.9993	0.9951	0.8797	0.8615
roberta-base, 5 εποχές, προσαύξηση (A + B σύνολα δεδομένων), χωρίς βάρη												
0.9975	0.8133	0.6929	0.9993	0.9936	0.728	0.7199	0.7163	0.9740	0.9983	0.8585	0.8497	0.8508
roberta-base, 1 εποχή												
0.9995	0.6893	0.6388	1.0	0.9896	0.6013	0.5761	0.5877	0.7980	0.9993	0.9803	0.848	0.7876
roberta-base, 8 εποχές												
0.9996	0.9203	0.7041	0.9993	0.9966	0.6896	0.6707	0.7123	0.9715	0.9997	0.9852	0.864	0.8642
roberta-base, 8 εποχές, max_len=256, bs=32												
1	0.8407	0.5399	1	0.9910	0.9517	0.8396	0.3436	0.9022	0.999	0.9570	0.7943	0.8215
roberta-base, 5 εποχές												
0.9996	0.872	0.713	1	0.989	0.633	0.664	0.715	0.9196	0.9993	0.9692	0.8923	0.8519
roberta-base, 8 εποχές												
0.9996	0.9203	0.7041	0.9993	0.9966	0.6896	0.6707	0.7123	0.9715	0.9997	0.9852	0.864	0.8642
roberta-base, 8 εποχές on 20,000 instances												
0.9995	0.6777	0.6647	0.9997	0.9977	0.437	0.5357	0.6783	0.8562	0.998	0.8590	0.8143	0.7675
roberta-base, 1 εποχή on 20,000 instances												
0.9957	0.5407	0.5770	0.9963	0.9733	0.073	0.3604	0.5337	0.7912	0.9943	0.7963	0.731	0.6448
llama-2-7B, max_length =128, bs=8, peft, 1 εποχή on 20,000 instances												
0.1667	1	0	0	0	0	0	0	0	0	0	0	0.1667
roberta-base, PromptTuningConfig(prompt_length=50), 8 εποχές												
0.9363	0.191	0.4964	0.9647	0	0	0.5025	0.759	0.5548	0.9787	0.9850	0.549	0.5737
roberta-base, PromptTuningConfig(prompt_length=100), 8 εποχές												
0.9751	0.248	0.4487	0.964	0.025	0.0007	0.5223	0.6917	0.5730	0.9867	0.9847	0.5163	0.5679
roberta-base, PromptTuningConfig(prompt_length=50), 1 εποχή												
0.9115	0.0927	0.2637	0.9737	0.1925	0.036	0.1524	0.046	0.4318	0.666	0.5736	0.1	0.3191
deberta-v3-base, max_len=1024, 4 εποχές, bs=4												
1	0.4207	0.4526	0.9997	0.25	0.0083	0.7758	0.7807	0.9977	0.9993	0.6083	0.8087	0.6696

Table 1.10: **Υποερώτημα Β: Μονόγλωσση Δυαδική Ταξινόμηση .** Στατιστικά δεδομένα για τα σύνολα Εκπαίδευσης/Δοκιμής/Αξιολόγησης

η χρήση των βαρών φαίνεται να συνεισφέρει κατά μόνο 0.01 για την ακρίβεια. Ο προσαρμογέας συντονισμού που δίνει 0.94 ακρίβεια για το υποερώτημα Α, φαίνεται να αποτυγχάνει για το πιο δύσκολο υποερώτημα Β, αυτό της απόδοσης συγγραφέα. Επίσης, πειραματιζόμαστε να χρησιμοποιήσουμε ένα μεγαλύτερο μοντέλο meta-llama/Llama-2-7b, το οποίο τελικά δίνει πολύ κακά αποτελέσματα όταν εκπαιδεύεται σε 20,000 δείγματα. Συγκριτικά, το roberta-base δίνει μια ακρίβεια γύρω στο 0.8 όταν εκπαιδεύεται σε μόνο 20,000 δείγματα και 1 εποχή.

Τέλος, δοκιμάσαμε να μετατρέψουμε το υποερώτημα Β στο υποερώτημα Α για να επωφεληθούμε της μεγάλης ακρίβειας που επιτυχαμε σε αυτό πριν. Συγχωνεύσαμε τις κλάσεις 1 εως 5 σε μία αλλά το αποτέλεσμα ήταν η πολύ κακή ανάκληση για την κλάση 0, ακόμα χειρότερη και από όταν έχουμε και τις 6 κλάσεις. Αυτή η χαμηλή επίδοση μπορεί πάλι να αποδοθεί στο γεγονός ότι αυτή η συγχώνευση κλάσεων καθιστά πάλι το σύνολο εκπαίδευσης μη ισορροπημένο. Τα αποτελέσματα φαίνονται και στον πίνακα 1.10

1.3.4 Περιπλοκότητα

Για τον υπολογισμό της περιπλοκότητας, δοκιμάσαμε να χωρίσουμε την ακολουθία εισόδου σε κομμάτια ίσα με το μήκος βήματος και χρησιμοποιήσαμε ως ταξινομητές τα δέντρα αποφάσεων LightGBM [34] και XGBoost [56]. Δοκιμάσαμε επίσης να συνδυάσουμε διαφορετικά μοντέλα και διαφορετικό μήκος βήματος και διαστήματος συμφραζομένων.

1.3.5 Υποερώτημα Α

Χρησιμοποιήσαμε το model = LGBMClassifier(max_depth=3, objective='binary') και model2 = XGBClassifier(max_depth=3, learning_rate=0.2, n_estimators=40, objective="binary:hinge") μετά από μια χειρωνακτική δοκιμή υπερπαραμέτρων, αφήνοντας τις περισσότερες παραμέτρους στις προτεινόμενες τιμές τους.

Μοντέλο (Βήμα/Μήκος Συμφραζομένων)	**Κλάση 0**		**Κλάση 1**		**Συνολική Ακρίβεια**
	Ακρίβεια	**Ανάκληση**	**Ακρίβεια**	**Ανάκληση**	
gpt2 (1024)	0.7480	0.9072	0.8961	0.7238	0.8109
gpt2 (512)	0.7648	0.9074	0.8993	0.7477	0.8235
gpt2-xl (1024)	0.7762	0.9449	0.9380	0.7537	0.8445
gpt2-xl (512)	0.7819	0.9508	0.9447	0.7603	0.8507
gpt2-xl (256)	0.7866	0.9649	0.9601	0.7634	0.8591
gpt2-xl (128)	0.8084	0.9564	0.9527	0.7951	0.8717
gpt2-xl (64)	0.7934	0.9491	0.9441	0.7766	0.8585
gpt2-xl (256/1024)	0.7795	0.9470	0.9405	0.7578	0.8476
gpt2-xl (256) + gpt2-xl (128)	0.7926	0.9638	0.9593	0.7719	0.8630
gpt2 (512) + gpt2-xl (512)	0.7770	0.9505	0.9439	0.7534	0.8470
gpt2-xl (128/256)	0.7657	0.9375	0.9291	0.7406	0.8341
gpt-neox-20b (1024)	0.6558	0.9904	0.9838	0.5301	0.7486
Llama-2-7b-hf (1024)	0.7221	0.9949	0.9930	0.6539	0.8158
roberta-base (512)	0.7062	0.9869	0.9815	0.6289	0.7989
roberta-large (512)	0.6438	0.9921	0.9860	0.5038	0.7356
dolly-v2-12b (512)	0.7321	0.9978	0.9971	0.6698	0.8256
dolly-v2-12b (1024)	0.7192	0.9978	0.9969	0.6479	0.8140
dolly-v2-12b (1024) + dolly-v2-12b (512)	0.7303	0.9978	0.9971	0.6668	0.8240
dolly-v2-12b (512) + gpt2-xl (512)	0.7824	0.9877	0.9854	0.7516	0.8637
gpt-neox-20b (1024) + gpt2-xl (1024)	0.7783	0.9422	0.9354	0.7574	0.8451
Llama-2-7b-hf (1024) + gpt2-xl(1024)	0.7639	0.9231	0.9143	0.7422	0.8281
gpt2-xl (1024) + roberta-base (512)	0.7564	0.9701	0.9637	0.7176	0.8375
gpt2-xl (1024) + gpt2(1024)	0.7673	0.9458	0.9379	0.7407	0.8381
gpt-neox-20b (1024) + gpt2-xl (1024) + dolly-v2-12b (1024)	0.8214	0.9850	0.9835	0.8064	0.8912
gpt-neox-20b (1024) + gpt2-xl (1024) + dolly-v2-12b (1024) + Llama-2-7b-hf (1024)	0.8062	0.9836	0.9815	0.7863	0.8800

gpt-neox-20b (1024) + gpt2-xl (512) + dolly-v2-12b (512)	0.8468	0.9844	0.9835	0.839	0.9080
gpt-neox-20b (1024) + gpt2-xl (128) + dolly-v2-12b (512)	0.8417	0.9894	0.9886	0.8318	0.9066
Llama-2-7b-hf (1024) + gpt2-xl(1024) + dolly-v2-12b (1024)	0.7501	0.9827	0.9783	0.704	0.8363

Table 1.11: **Υποερώτημα A: Μονόγλωσση Δυαδική Ταξινόμηση.** LightGBM Ταξινομητής για σκορ περιπλοκότητας

Μοντέλο	**Κλάση 0**		**Κλάση 1**		**Συνολική ακρίβεια**
	Ακρίβεια	**Ανάκληση**	**Ακρίβεια**	**Ανάκληση**	
gpt2 (1024)	0.7547	0.9017	0.8922	0.7351	0.8142
gpt2 (512)	0.7656	0.9069	0.8990	0.7489	0.8239
gpt2-xl (1024)	0.8149	0.9182	0.9165	0.8115	0.8622
gpt2-xl (512)	0.7863	0.9486	0.9429	0.7669	0.8532
gpt2-xl (256)	0.8121	0.9522	0.9488	0.8009	0.8727
gpt2-xl (128)	0.8068	0.9570	0.9533	0.7928	0.8708
gpt2-xl (64)	0.7922	0.9496	0.9445	0.7748	0.8578
gpt2-xl (256/1024)	0.7943	0.9392	0.9342	0.7801	0.8557
gpt2-xl (128/256)	0.7764	0.9309	0.9239	0.7576	0.8399
gpt2-xl (256) + gpt2-xl (128)	0.8123	0.9527	0.9494	0.8009	0.8730
gpt-neox-20b (1024)	0.6613	0.9894	0.9827	0.5421	0.7545
Llama-2-7b-hf (1024)	0.7596	0.9905	0.9881	0.7167	0.8467
roberta-base (512)	0.7138	0.9851	0.9795	0.643	0.8054
roberta-large (512)	0.6469	0.9919	0.9858	0.5105	0.7391
dolly-v2-12b (512)	0.7326	0.9978	0.9970	0.6708	0.8260
dolly-v2-12b (1024)	0.7212	0.9977	0.9968	0.6513	0.8157
dolly-v2-12b (512) + gpt2-xl(512)	0.7953	0.9876	0.9857	0.7702	0.8734
gpt-neox-20b (1024) + gpt2-xl (1024)	0.7871	0.9373	0.9315	0.7708	0.8499
Llama-2-7b-hf (1024) + gpt2-xl (1024)	0.8143	0.9182	0.9164	0.8107	0.8618
gpt2-xl (1024) + roberta-base (512)	0.8006	0.9487	0.9443	0.7863	0.8634
gpt2-xl (1024) + gpt2 (1024)	0.7592	0.9523	0.9440	0.727	0.8340
gpt-neox-20b (1024) + gpt2-xl (1024) + dolly-v2-12b (1024)	0.7897	0.9876	0.9856	0.7623	0.8693
gpt-neox-20b (1024) + gpt2-xl (512) + dolly-v2-12b (512)	0.8205	0.9864	0.9850	0.8049	0.8911
gpt-neox-20b (1024) + gpt2-xl (128) + dolly-v2-12b (512)	0.8234	0.9882	0.9870	0.8084	0.8938
gpt-neox-20b (1024) + gpt2-xl (1024) + dolly-v2-12b (1024) + Llama-2-7b-hf (1024)	0.7815	0.9880	0.9857	0.7503	0.8632
Llama-2-7b-hf (1024) + gpt2-xl (1024) + dolly-v2-12b (1024)	0.7688	0.9874	0.9847	0.7316	0.8531

Table 1.12: **Υποερώτημα A: Μονόγλωσση Δυαδική Ταξινόμηση.** XGBoost Ταξινομητής για σκορ περιπλοκότητας

Παρατηρούμε, καλύτερα αποτελέσματα όταν χρησιμοποιούμε μικρότερο μήκος βήματος για τα διαστήματα.. Επίσης, τα μοντέλα gpt2-xl and dolly-v2-12b δίνουν τα καλύτερα αποτελέσματα ως μοντέλα γεννήτριες των κειμένων. Ο συνδυασμός που δίνει μια ακρίβεια πάνω από 0.9 είναι των μοντέλων EleutherAI/gpt-neox-20b (512) + Gpt2XL(512) + dolly-v2-12b (512). Το XGBoost δίνει τα καλύτερα αποτελέσματα εκτός από την καλύτερη περίπτωση.

1.3.6 Υποερώτημα B

Χρησιμοποιήσαμε model = LGBMClassifier(max_depth=3, n_estimators=10, objective='multiclass', num_class=6) και model2 = XGBClassifier(max_depth=3, learning_rate=0.2, objective="multi:softmax", num_class=6) μετά από μια χειρωνακτική δοκιμή υπερπαραμέτρων, αφήνοντας τις περισσότερες παραμέτρους στις προτεινόμενες τιμές τους.

Κλάση 0		Κλάση 1		Κλάση 2		Κλάση 3		Κλάση 4		Κλάση 5		Συνολική ακρίβεια
P	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	
bloomz-560m (512)												
0.7586	0.7657	0.3414	0.8077	0.0627	0.0213	0.1780	0.2673	0.5821	0.2707	0.1709	0.0543	0.3645
bloomz-560m (1024)												
0.6728	0.7943	0.3490	0.8387	0.0434	0.0123	0.1667	0.2643	0.5940	0.139	0.1631	0.051	0.3499
bloom-560m (512)												
0.5799	0.8967	0.3265	0.8077	0.0728	0.027	0.1684	0.271	0	0	0	0	0.3337
bloomz-7b1 (512)												
0.7752	0.9103	0.3763	0.8103	0.0817	0.0223	0.2088	0.4137	0.5581	0.2307	0	0	0.3979
bloomz-560m (512) + bloomz-7b1 (512)												
0.6659	0.8377	0.3591	0.8033	0.0822	0.0247	0.2258	0.3767	0.5913	0.082	0.1417	0.0563	0.3634
gpt2 (1024)												
0.9546	0.4693	0.3097	0.8587	0.0523	0.015	0.1526	0.2043	0.4247	0.4237	0.2714	0.0307	0.3336
gpt2-xl (512)												
0.9663	0.554	0.3349	0.8703	0.0649	0.0147	0.1898	0.2687	0.4119	0.4887	0	0	0.3661
gpt2 (1024) + gpt2-xl (512)												
0.9202	0.7577	0.3382	0.8687	0.0630	0.015	0.2123	0.3167	0.5817	0.4817	0.6536	0.0333	0.4122
dolly-v2-3b (512)												
0.5623	0.761	0.4212	0.8567	0.0528	0.01	0.2442	0.5193	0.2362	0.07	0	0	0.3695
dolly-v2-12b (1024)												
0.3797	0.4093	0.3110	0.738	0.0224	0.0063	0.4003	0.4893	0.1902	0.1343	0.4625	0.1563	0.3223
dolly-v2-3b (512) + dolly-v2-12b (1024)												
0.6546	0.5863	0.4377	0.85	0.0596	0.0133	0.3585	0.4933	0.4563	0.332	0.9677	0.808	0.5138
Mistral-7B-v0.1 (1024)												
0.7331	0.6127	0.3198	0.7003	0.0538	0.0197	0.2697	0.3577	0.3286	0.3983	0.2676	0.019	0.3513
llama3-8B (1024)												
0.9029	0.6293	0.3704	0.824	0.0768	0.0183	0.2684	0.4053	0.1321	0.1203	0.3384	0.1417	0.3565
gpt2-xl (512) + dolly-v2-12b (1024)												
0.7873	0.5947	0.4037	0.868	0.0628	0.015	0.3499	0.5047	0.4992	0.4077	0.9687	0.578	0.4947
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024)												
0.8557	0.6087	0.4450	0.8573	0.0667	0.014	0.3644	0.5467	0.5710	0.476	0.9670	0.791	0.5489
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + bloomz-7b1												
0.8142	0.593	0.4350	0.8673	0.0711	0.0153	0.3573	0.521	0.5581	0.4533	0.9709	0.7903	0.5401
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + bloomz-7b1 + bloomz-560m												

0.7087	0.57	0.4537	0.859	0.0541	0.0117	0.3659	0.5463	0.5221	0.418	0.9769	0.7747	0.5299
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + llama38b (1024)												
0.8586	0.5647	0.4505	0.9107	0.0766	0.0123	0.4449	0.603	0.5489	0.4883	0.9905	0.906	0.5808
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + mistralv1 (1024)												
0.8678	0.6017	0.4584	0.929	0.0519	0.0093	0.4783	0.6363	0.5864	0.5113	0.9948	0.8933	0.5968
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + mistralv1 (1024) + llama38b (10												
0.8712	0.5707	0.4666	0.9273	0.0486	0.0093	0.4779	0.6567	0.5696	0.517	0.9962	0.8807	0.5936

Table 1.13: **Υποερώτημα Β: Ανίχνευση Γεννήτριας.** LightGBM Ταξινομτής για σκορ περιπλοκότητας

Κλάση 0		**Κλάση 1**		**Κλάση 2**		**Κλάση 3**		**Κλάση 4**		**Κλάση 5**		**Συνολική ακρίβεια**
P	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	
bloomz-560m (512)												
0.6690	0.828	0.3446	0.808	0.0642	0.021	0.2028	0.2187	0.5762	0.0983	0.2117	0.178	0.3587
bloomz-560m (1024)												
0.6658	0.7923	0.3701	0.8183	0.0457	0.0123	0.2008	0.2833	0.5831	0.131	0.2034	0.141	0.3631
bloom-560m (512)												
0.5379	0.802	0.3468	0.7997	0.065	0.0217	0.2045	0.2633	0.1615	0.0207	0.2004	0.091	0.3331
bloomz-7b1 (512)												
0.7207	0.915	0.3768	0.825	0.0827	0.0213	0.2305	0.318	0.4882	0.0827	0.2589	0.19	0.392
bloomz-560m (512) + bloomz-7b1 (512)												
0.7963	0.753	0.3913	0.827	0.0654	0.0177	0.2912	0.4533	0.8322	0.3687	0.2081	0.1397	0.4266
gpt2 (1024)												
0.9661	0.3797	0.3284	0.845	0.0462	0.0123	0.1691	0.2237	0.3927	0.3573	0.2001	0.107	0.3208
gpt2-xl (512)												
0.9685	0.5433	0.3621	0.8387	0.0735	0.0176	0.1963	0.319	0.4039	0.414	0.2507	0.0583	0.3652
gpt2 (1024) + gpt2-xl (512)												
0.8675	0.74	0.3996	0.8317	0.0854	0.0223	0.2641	0.493	0.6576	0.3227	0.2886	0.129	0.4231
dolly-v2-3b (512)												
0.5484	0.7403	0.4020	0.842	0.0574	0.0133	0.2406	0.4417	0.1565	0.0447	0.2360	0.0477	0.3549
dolly-v2-12b (1024)												
0.3475	0.373	0.3250	0.8163	0.0216	0.0057	0.4	0.488	0.1807	0.1113	0.3568	0.113	0.3179
dolly-v2-3b (512) + dolly-v2-12b (1024)												
0.7524	0.6837	0.4418	0.851	0.0901	0.0147	0.3748	0.489	0.5977	0.4423	0.9666	0.926	0.5679

Mistral-7B-v0.1 (1024)

0.6456	0.6007	0.3372	0.704	0.0521	0.0187	0.2773	0.435	0.2685	0.2003	0.3164	0.0977	0.3427

llama3-8B (1024)

0.9230	0.5753	0.3625	0.8093	0.0830	0.0217	0.2559	0.4277	0.1290	0.1113	0.3627	0.1263	0.3453

gpt2-xl (512) + dolly-v2-12b (1024)

0.8025	0.5147	0.4012	0.874	0.0802	0.0173	0.3757	0.542	0.4766	0.4493	0.9482	0.549	0.4911

gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024)

0.8519	0.621	0.4626	0.8817	0.1952	0.038	0.3960	0.5937	0.6340	0.4653	0.9666	0.906	0.5843

gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + bloomz-7b1

0.8798	0.576	0.4665	0.8747	0.1607	0.033	0.4024	0.607	0.5980	0.489	0.9592	0.9007	0.5801

gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + bloomz-7b1 + bloomz-560m

0.8891	0.5957	0.4694	0.878	0.1095	0.018	0.4318	0.6267	0.6877	0.61	0.9526	0.9113	0.6066

gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + llama38b (1024)

0.7990	0.6123	0.4697	0.9287	0.0932	0.0123	0.5553	0.7017	0.6543	0.5477	0.9668	0.9893	0.632

gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + mistralv1 (1024)

0.8247	0.6337	0.4911	0.9593	0.0905	0.014	0.5484	0.7483	0.7041	0.5323	0.9824	0.9853	0.6455

gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + mistralv1 (1024) + llama38b (10

0.7879	0.4967	0.4915	0.9607	0.1030	0.016	0.5611	0.7503	0.6212	0.5537	0.9580	0.9877	0.6275

Table 1.14: **Υποερώτημα B: Ανίχνευση Γεννήτριας.** Ταξινομητής XGBoost για σκορ περιπλοκότητας

Παρατηρούμε ότι το πιο δύσκολο αυτό πρόβλημα οδηγεί σε χαμηλότερες τιμές ακρίβειας αφού χρησιμοποιούμε μόνο 3 από τις γεννήτριες που χρησιμοποιήθηκαν για την παραγωγή των κειμένων. Το κύριο συμπέρασμα είναι ότι όταν χρησιμοποιούμε ταυτόχρονα διαφορετικά μεγέθη του ίδιου μοντέλου γεννήτρια, έχουμε μεγάλη αύξηση στα μεγέθη ακρίβειας και ανάκλησης της αντίστοιχης τάξης (gpt2-xl (512), gpt-2 (1024), dolly-v2-3b (512), dolly-v2-12b (1024), bloomz-7b και bloomz-560m).

1.4 Συμπεράσματα

Σε αυτή τη μελέτη, διεξάγαμε μια σειρά από εκτενή πειράματα για τα ερωτήματα MGTD και AA. Δοκιμάσαμε διαφορετικές αρχιτεκτονικές προσαρμογέων με το μοντέλο roberta-base για να δούμε πως επηρεάζεται η ακρίβεια μεταβάλλοντας τον αριθμό εποχων και μήκος ακολουθίας διακριτικών. Για την πολύγλωσση περίπτωση, διαπιστώσαμε πως το μονόγλωσσο roberta-base έχει καλύτερη επίδοση από το πολύγλωσσο xlm-roberta-base στα Αγγλικά κείμενα, που αποτελούν και το μεγαλύτερο τμήμα του συνόλου αξιολόγησης. Επιχειρήσαμε επίσης να μεταφράσουμε τα κείμενα καθώς και να χρησιμοποιήσουμε γλωσσικούς προσαρμογείς και προσαρμογείς εργασίας. Τέλος, επιβεβαιώσαμε πως η μετρική της περιπλοκότητας μπορεί να αποτελέσει μια εναλλακτική μέθοδο επιβλεπόμενης μάθησης. Η χρήση ενός μικρότερου βήματος διαχωρισμού της ακολουθίας εισόδου για τους υπολογισμούς της περιπλοκότητας οδηγεί σε καλύτερα αποτελέσματα. Επίσης, ο υπολογισμός περιπλοκότητας με τα μοντέλα γεννητριών Bloomz, GPT-2 και Dolly ταυτόχρονα οδήγησε σε καλύτερα αποτελέσματα.

Κλείνοντας αυτή τη διατριβή θα θέλαμε να προτείνουμε μερικές κατευθύνσεις για περαιτέρω βελτίωση αυτής της εργασίας που ίσως εμπνεύσουν και για ενδιαφέρουσες διαφορετικές προσεγγίσεις. Αρχικά, θα μπορούσε να διρευνηθεί η παράλληλη χρήση στυλιστικών στοιχείων, μετρικής περιπλοκότητας και εκπαίδευσης προεκπαιδευμένων γλωσσικών μοντέλων σε ένα ενιαίο σύστημα. Όσον αφορά την εκπαίδευση γλωσσικών μοντέλων και προσαρμογέων, είναι η δυνατή η αξιοποίηση μεθόδων ensembling για ένα καλύτερο τελικό αποτέλεσμα. Για την περίπτωση της μετρικής περιπλοκότητας, επειδή ο υπολογισμός με ιδιωτικά μοντέλα δεν είναι εφικτός, χρειάζεται η προσέγγιση της μέσω ενός παραπλήσιου μεγέθους που θα υπολογίζεται με βάση τα παραγόμενα κείμενα και χωρίς πρόσβαση στις παραμέτρους. Σε κάθε περίπτωση, η δοκιμή LLMs μεγάλης κλίμακας θα έχει μόνο νόημα αν ευρεθούν οι κατάλληλοι υπολογιστικοί πόροι.

Chapter 2

Introduction

The proliferation of Large Language Models (LLMs) has led to a significant increase in the volume of machine-generated text (MGT) across a wide range of domains. This rise has sparked concerns regarding the potential for misuse in fields such as journalism, education, academia, etc (Uchendu et al., 2023 [3], Crothers et al., 2023 [22]. Moreover, it poses challenges to maintaining information integrity and ensuring accurate information dissemination. As such, the ability to accurately distinguish between human-written content and machine-generated content has become paramount for identifying potential misuse (Jawaharetal et al., 2020 [23], Stiff and Johansson, 2022 [53], Macko et al., 2023 [17]).

In response to these challenges, SemEval2024 is introducing a shared task (Task 8) that focuses on the detection of machine-generated text across multiple generators, domains, and languages. It is providing large-scale evaluation datasets for three subtasks with the primary goals of fostering extensive research in MGT detection, advancing the development of automated systems for detecting MGT, and reducing instances of misuse:

Subtask A: Human vs. Machine Classification. The goal of this subtask is to accurately classify a text as either produced by a human or generated by a machine. This is the basic, but one of the most common use-cases of MGTD systems for preventing the misuse of LLMs. This task is divided into two tracks: (i) The monolingual track, which focuses solely on English texts; and (ii) The multilingual track, which involves texts in a variety of languages, thereby expanding the diversity and complexity beyond existing benchmarks.

Subtask B: Multi-Way Generator Detection. This task aims to pinpoint the exact source of a text, i.e., determine whether it originated from a human or a specific LLM (GPT-3, GPT-3.5, GPT-4, Cohere, DALL-E, or BLOOMz). Determining a particular LLM that potentially generated the given text is important from several perspectives: it can help to narrow down the set of LLMs for more sensitive white-box detection techniques or in cases where the generated material is harmful, misleading, or illegal, it might be useful for addressing ethical concerns and legal obligations.

Subtask C: Changing Point Detection. The goal of this subtask is to precisely identify the exact boundary (changing point) within a text at which the authorship transitions from a human to machine happens. The texts begin with human-written content, which at some point is automatically continued by LLMs (GPT and LlaMA series). The percentage of the human-written section varies from 0 to 50 percent. This task takes into account the fact that in many malignant use-cases of LLMs, the part of the text might be written by a human and a part might be generated by a machine. It is hard to classify a text as machine-generated if a big chunk is actually human-written. This is a way to obscure the usage of LLM, and the formulation of Subtask C addresses this challenge.

In this thesis, a comprehensive range of implementations for Machine-Generated Text Detection (MGTD) is presented, addressing subtasks A and B of the competition. Several experiments were undertaken for each of the proposed methods, resulting in comprehensive contributions to this intriguing task:

- We fine-tuned pre-trained Language Models (PLMs), examining how hyper-parameters can affect performance for the problems of MGTD and AA.

- We experimented on adapter tuning and specifically prompt tuning, achieving competitive results.
- For the multilingual task of MGTD, we attempted to detect the source language of the texts and translate them. We also found if language and task adapters can lead to further improvements
- Lastly, We calculated fixed-sequence length perplexity using multiple PLMs and measured its effectiveness as a metric for the subtasks of MGTD and AA

Chapter 3

Related Work

Contents

3.1 Large Language Models - LLMs

A **language model** is a probabilistic model of a natural language. Large language models (LLMs), currently their most advanced form, combine larger datasets, feed-forward neural networks and transformers.

3.1.1 N-gram models

The first methodology for language modeling was **n-grams models**. These are purely statistical models and are based on the assumption that the probability of the next word in a sequence depends only on a fixed size window of previous words. A bigram model considers one word, a trigram two, and in general a n-gram considers n-1 previous words.

Hence, the probability $P(w_1, \ldots, w_n)$ of observing the word sequence $w_1, \ldots, w_n$ can be approximated (according to n-grams models assumption) by the probability of observing it in the shortened context window of the previous n-1 words (n-th order Markov property):

$$P(w_1, \ldots, w_n) = \prod P(w_i|w_1, \ldots, w_{i-1}) \approx \prod P(w_i|w_{i-(n-1)}, \ldots, w_{i-1}) \tag{3.1.1}$$

The conditional probability for each word and its previous context can be calculated from n-gram model frequency counts:

$$P(w_i|w_{i-(n-1)}, \ldots, w_{i-1}) = \frac{count(w_{i-(n-1)}, \ldots, w_{i-1}, w_i)}{count(w_{i-(n-1)}, \ldots, w_{i-1})} \tag{3.1.2}$$

The estimation of these probabilities constitutes the training of an n-gram model. In some cases, the language model is estimated with a specific fixed vocabulary. As a result, an issue that arises is how the out-of-vocabulary (OOV) words will be handled. One of the approaches to this, is to ignore the OOV words, in which case the n-gram probabilities are smoothed over all the words in the vocabulary even if they were not observed. One other approach is to represent all OOV words with a special token (e.g. <unk>).

3.1.2 Neural models

N-gram models were superseded by neural networks. A **neural network** (also **artificial neural network (ANN)** or **neural net (NN)**) is a model inspired by the structure and function of biological neural networks. The two broad types of ANNs are the uni-directional Feedforward Neural Networks (FFNs) and the Recurrent Neural Networks (RNNs).

Deep learning is the subset of machine learning based on neural networks with representation learning. The 'deep' refers to the use of multiple layers in the network. A hierarchy of layers is used to transform input data into a slightly more abstract representation. Most deep learning models are based on multi-layered neural networks such as convolutional neural networks and transformers.

Feedfoward neural networks (FNNs) have a uni-directional flow of information from input nodes to output nodes, without any cycles or loops. They process data in a sequential manner (i.e. one word at a time) but lack the ability to maintain any past context. Additionally, they necessitate for a fixed length of the input sequence length depending on the number of input neurons.

Recurrent neural networks (RNNs) [39] have a bi-directional flow. They analyze input sequences sequentially but they predict the subsequent word by considering the current word and the previous hidden state. The hidden state can in principle represent information about all of the previous words all the way back to the begging of the sequence. Consequently, RNNs do not face the limited context problem observed in n-gram models and FNNs. Furthermore, they can process a sequence of varying length in contrast to FNNs.

The RNNs have an infinite impulse response and they can be conceptualized as directed acyclic graphs that can be unrolled and replaced with a FNN. This architecture allowed for the capture of contextual information and inter-word dependencies, representing a significant improvement over conventional word embeddings. However, early RNNs encountered difficulties in modelling long-range dependencies due to the vanishing gradient problem. To address this problem, later extensions such as Long short-term memory (LSTM), Gated Recurrent Unit (GRU) etc have been proposed.

3.1.3 Transformers

A **transformer** is a deep learning architecture developed by Google scientists and presented in the 2017 paper "Attention is all you need" [58]. The transformers architecture [58] replaced sequential processing with a self-attention mechanism, enabling words to interact directly regardless of their proximity in the text. More specifically, at each layer, each token is contextualized within the scope of the context window with other (unmasked) tokens via a parallel multi-head attention mechanism allowing the signal for key tokens to be amplified and less important tokens to be diminished. This innovation facilitated the ability of transformers to more efficiently capture extensive dependencies compared to conventional approaches and RNNs. The lack of recurrent units also requires less training and has therefore been adopted for training LLMs on large datasets. The attention mechanism simulates how human attention works by assigning varying levels of importance to different words in a sentence. For each word, soft weights are calculated for its numerical representation (embedding) within a specific context window. Soft weights can adapt and change with each use of the model. The correlation of words are captured in neuronal weights either from self-supervised pretraining or supervised fine-tuning.

The Transformer architecture is summarised below:

1. **Tokenizer**

   The tokenizer converts text into tokens (returns input ids and attention mask). Tokens are used instead of words to account for polysemy. The input ids are often the only required parameters to be passed to the model as input. They are token indices, numerical representations of tokens building the sequences that will be used as input by the model. Each tokenizer works differently but the underlying mechanism remains the same. The tokenizer takes care of splitting the sequence into tokens available in the tokenizer vocabulary. The tokens are either words or subwords. To indicate the tokens of subwords are not separate words but parts of the same word, a double-hash prefix may be used. These tokens are then converted into ids which are understandable by the model. The tokenizer returns a dictionary with all the arguments necessary for its corresponding model to work properly. The token indices are under the key input_ids. The tokens can be decoded back to words.

   The attention mask indicates to the model which tokens should be attended to, and which should not. For example, when a sequence needs to be padded up to a specified length, the list of ids will be extended with padding indices and the attention mask will serve as a binary tensor indicating the position of the padded indices so that the model does not attend to them.

2. **Embedding layer**

   The embedding layer converts tokens and positions of the tokens into vector representations. Word representations or embeddings are real-valued vectors that encode the meaning of the word in a way that words closer in the vector space are semantically more similar. The positional encodings are fixed-size vector representations that encapsulate the relative positions of tokens.

3. **Encoder**

   The encoder consists of a stack of identical layers. Each layer has two sub-layers:

   - **Multi-Head Self-Attention**

     This sub-layer computes a weighted sum of embeddings, allowing each word to focus on different parts of the input sequence. Therefore, the encoder is bidirectional. Attention can be placed on tokens before and after the current token. Multiple attention heads run in parallel, capturing different relationships between words.

   - **Position-wise Feed-Forward Neural Network**

     After the attention mechanism, each token's representation is passed through a position-wise feed-forward neural network. This introduces non-linearity and further refines the token representations.

   Residual connections[**residual-connections**], followed by layer-normalization[**layer-normalization**] are employed around each of the sub-layer.

4. **Decoder**

   The decoder also consists of a stack of identical layers, each containing three sub-layers:

   - **Masked Multi-Head Self-Attention**

     This sub-layer acts similar to the corresponding encoder's sub-layer, but with a mask applied to prevent attending to future positions during training. Attention cannot be placed on future tokens and this allow for autoregressive text generation

   - **Multi-Head Encoder-Decoder Attention**

     This sub-layer focuses on the encoded input sequence, allowing the decoder to consider the relevant parts of the input during sequence generation.

   - **Position-wise Feed-Forward Neural Network**

     Similar to the encoder, this sub-layer follows the attention mechanisms.

   As with the encoder, residual connections are used around each sub-layer, followed by layer-normalization.

5. **Output Generation**

   The output of the final decoder layer is transformed into probability distributions over the output vocabulary using a linear transformation followed by a softmax activation. Throughout the training process, the model is fed with a word sequence as input to predict the subsequent word.

Several architectural variations of the Transformer have been proposed since it was first introduced by [58]. The masking pattern used on the inputs, which acts as contextual information for the model to generate a prediction, is a key distinction between these systems. In large language language models, the terminology is somewhat different than the terminology used in the original Transformer paper:

- **Encoder-Decoder** (full encoder, autoregressive decoder)

  As previously indicated and originally proposed, the Transformer consisted of two stacks (Fig. 3.1.1): the encoder and the decoder. The encoder processes the input sequence and generates context-rich representations, which are used by the decoder to generate the output sequence step by step. Notable pretrained language models using an encoder-decoder architecture include BART[**bart**] and T5 [**t5**].

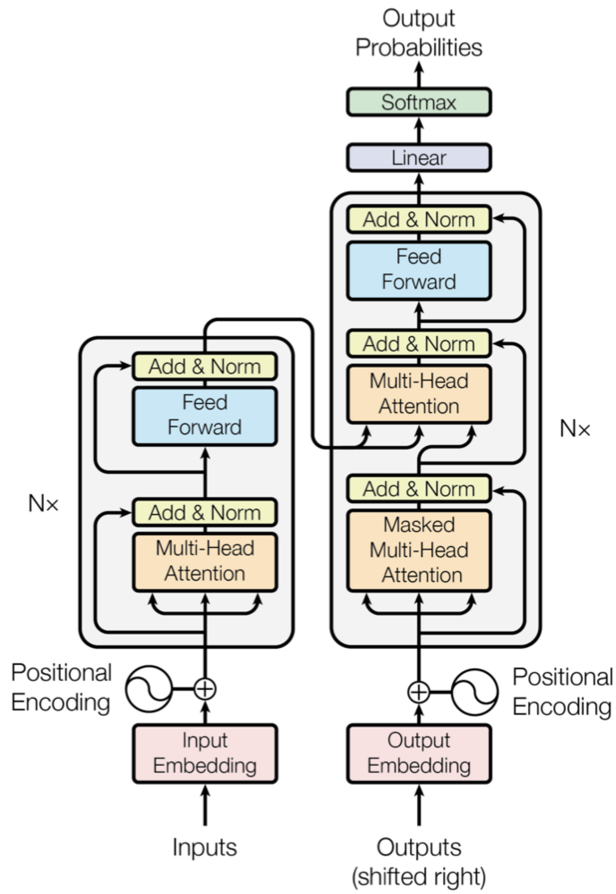

Figure 3.1.1: **The Transformer - model architecture**. The original Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of figure respectively[58]

.

- **Decoder-Only** (auto-regressive encoder, auto-regressive decoder)

  While the encoder-decoder design serves as the foundational variation of the Transformer model, contemporary LLMs predominately employ a decoder-only architecture. These models have the capability to train as a conventional language model, wherein they learn to predict the next token in a given sequence. Decoder-only models lack the ability to process or represent the input sequence and output sequence separately. All tokens are treated equally during processing, and conditioning is only dependent on prior tokens due to the causal masking pattern, implying that the representation of any conditioning text is intrinsically weaker. However, this produces a simpler architecture that is well-suited to a standard auto-regressive next-step-prediction pre-training objective. Notably, this architecture is the foundation of the GPT series of models [**gpt3**, 45] as well as numerous other recent LLMs [**bloom**, **palm**, **lamda**].

- **Encoder-Only** (full encoder, full decoder)

  As an aside, there is an additional prevalent architectural variant that employs only a Transformer encoder layer stack. This model architecture serves as the foundation for the ubiquitous BERT [**bert**] and its derivatives.

Overall, Transformers have revolutionised the field of NLP due to their capacity to efficiently manage sequential data, enabling parallelization and capturing long-range dependencies in texts. Using the attention mechanism to establish dependencies between input and output data, demonstrate that there is no requirement for convolutions or recurrent units to achieve state-of-the-art performance in linguistic tasks.

3.1.4 Large Language Models (LLMs)

LLMs are advanced computational models with vast parameter sizes and notable for their learning capabilities. They acquire these abilities by pre-training on large unstructured text corpora in a self-supervised and semi-supervised process. All modern LLMs are now built on Transformer architecture [58], which eschews recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. LLMs can be used for text generation by taking an input text and repeatedly predicting the next token or word. The largest and most capable LLMs are built with a decoder-only transformer-based architecture.

Pre-training is a crucial phase in the construction of LLMs, wherein the model undergoes training on an extensive, unlabeled dataset through the process of self-supervision. The selection of a pre-training objective can have a substantial influence on the subsequent applicability of the LLM. In this section, we provide an overview of the fundamental concepts behind the prevalent token-level pre-training objectives that have been extensively studied and documented in academic literature.

- **Masked Language Modeling (MLM)** was proposed by [**bert**]. Encoder-only models are commonly pre-trained with a masked language modeling objective. In the input text, either individual tokens or sequences of tokens are substituted with a designated mask token. The model is then trained to predict the omitted tokens.

- **Causal Language Modeling (CLM)** is used to train auto-regressive models, like encoder-decoder or decoder-only models, by predicting the next token given a prior sequence. This process enforces a causal relationship, where the model only attends to tokens that come before the predicted token in the sequence.

- **Next Sentence Prediction (NSP)** attempts to predict whether a given pair of sentences is consecutive or not. This objective mainly serves as a supplementary task in the pre-training phase of encoder-only models and facilitates the model's acquisition of sentence associations.

Pre-training LLMs on textual corpora embeds substantial factual knowledge in their parameters, which is essential for excelling in various downstream applications. These models often require further alignment to desired behaviors, typically achieved through supervised fine-tuning on instruction-following tasks and preference learning from human feedback.

With the wide success of pre-trained large language models, a range of techniques has arisen to adapt these general-purpose models to downstream tasks. ELMo (Peters et al., 2018) [38] proposed freezing the pre-trained model and learning a task-specific weighting of its per-layer representations. However, since GPT (Radford et al., 2018) [5] and BERT (Devlin et al., 2019) [26], the dominant adaptation technique has been model tuning (or "fine-tuning"), where all model parameters are tuned during adaptation, as proposed by Howard and Ruder (2018) [24].

More recently, Brown et al. (2020) [57] showed that prompt design (or "priming") is surprisingly effective at modulating a frozen GPT-3 model's behavior through text prompts. Prompt engineering is the process of structuring an instruction that can be interpreted and understood by a generative AI model. A prompt can be a query, a command or a longer statement containing context, instructions and conversation history. A prompt may include a few examples for the model to learn from, an approach called few-shot learning. Prompt engineering is enabled by in-context learning, defined as a model's ability to temporarily learn from prompts. This ability is an emergent ability of large language models such that its efficacy increases at a greater rate in larger models than in smaller models. An example of the significance of this emergent ability is found in chain-of-thought prompting (which improves after 62B). In contrast to training and fine-tuning which are not temporary, what has been learnt during in-context learning is of a temporary nature.

The practice of "freezing" pre-trained models is appealing, especially as model size continues to increase. Rather than requiring a separate copy of the model for each downstream task, a single generalist model can simultaneously serve many different tasks. Unfortunately, prompt-based adaptation has several key drawbacks. Task description is error-prone and requires human involvement, and the effectiveness of a prompt is limited by how much conditioning text can fit into the model's input. As a result, downstream task quality still lags far behind that of tuned models. For instance, GPT-3 175B fewshot performance on SuperGLUE is 17.5 points below fine-tuned T5-XXL (Raffel et al., 2020) [14] (71.8 vs. 89.3) despite using 16 times more parameters.

While prompt design involves selecting prompt tokens from a fixed vocabulary of frozen embeddings, prompt tuning can be thought of as using a fixed prompt of special tokens, where only the embeddings of these prompt tokens can be updated. Unlike the discrete text prompts used by GPT-3, soft prompts are learned through back-propagation. Prompt tuning is a further simplification for adapting language models. Lester et al., 2021 [11] freeze the entire pre-trained model and only allow an additional k tunable tokens per downstream task to be prepended to the input text. This "soft prompt" is trained end-to-end and can condense the signal from a full labeled dataset, allowing to outperform few-shot prompts and close the quality gap with model tuning. Prompt tuning alone (with no intermediate-layer prefixes or task-specific output layers) is sufficient to be competitive with model tuning. At the same time, since a single pre-trained model is recycled for all downstream tasks, we retain the efficient serving benefits of frozen models.

Normally, prompting is done by prepending a series of tokens, P, to the input X, such that the model maximizes the likelihood of the correct Y , $Pr_\theta(Y|[P;X])$, while keeping the model parameters, ϑ, fixed. In GPT-3, the representations of the prompt tokens, P = $p_1, p_2, \ldots, p_n$, are part of the model's embedding table, parameterized by the frozen ϑ. Finding an optimal prompt thus requires the selection of prompt tokens, through either manual search or non-differentiable search methods (Jiang et al., 2020 [64]; Shin et al., 2020 [54]). Prompt tuning removes the restriction that the prompt P be parameterized by ϑ; instead the prompt has its own dedicated parameters, θ_P , that can be updated. While prompt design involves selecting prompt tokens from a fixed vocabulary of frozen embeddings, prompt tuning can be thought of as using a fixed prompt of special tokens, where only the embeddings of these prompt tokens can be updated. Our new conditional generation is now $Pr_{\theta;\theta P}(Y|[P;X])$ and can be trained by maximizing the likelihood of Y via back=propagation, while only applying gradient updates to θ_P. There are many possible ways to initialize the prompt representations. The simplest is to train from scratch, using random initialization. A more sophisticated option is to initialize each prompt token to an embedding drawn from the model's vocabulary. Conceptually, our soft-prompt modulates the frozen network's behavior in the same way as text preceding the input, so it follows that a word-like representation might serve as a good initialization spot. For classification tasks, a third option is to initialize the prompt with embeddings that enumerate the output classes, similar to the "verbalizers" of Schick and Schütze (2021) [49]. Since we want the model to produce these tokens in the output, initializing the prompt with the embeddings of the valid target tokens should prime the model to restrict its output to the legal output classes.

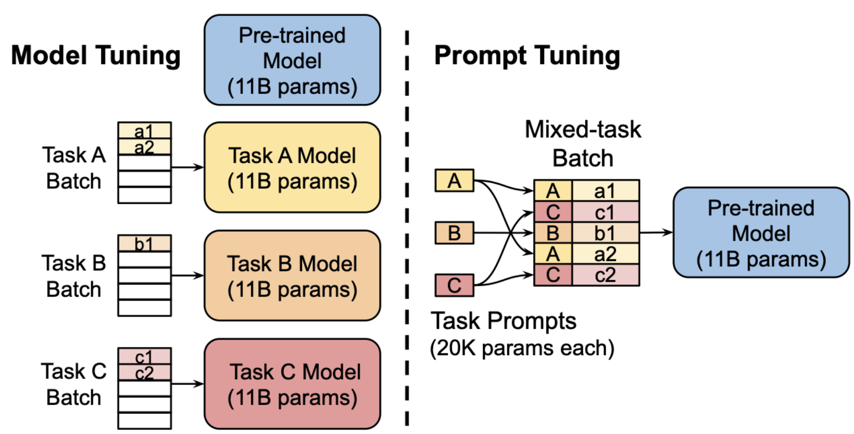

Figure 3.1.2: Prompt Tuning [**prompt-tuning**]

3.2 Machine-Genrated Text Detection - MGTD

Detecting machine-generated text is primarily formulated as a binary classification task (Zellers et al., 2019 [47], Gehrmannetal et al., 2019a [50], Solaiman et al., 2019 [25], Ippolito et al., 2019 [16]), naively distinguishing between human-written and machine-generated text. In general, there are two main approaches: the supervised methods (Wang et al., 2024a[63], Wang et al., 2024b[62], Uchendu et al., 2021 [4], Zellers et al., 2019 [47], Zhong et al., 2020 [59], Liu et al., 2022 [60]) and the unsupervised ones, such as zero-shot methods (Solaiman et al., 2019 [25], Ippolito et al., 2019 [16], Mitchell et al., 2023 [20], Su et al., 2023 [28], Hans et al., 2024 [1]). While supervised approaches yield relatively better results, they are susceptible to overfitting (Mitchell et al., 2023 [20], Su et al., 2023 [28]). Meanwhile, unsupervised methods may require unrealistic white-box access to the generator. Detect-GPT [20], which uses only log probabilities computed by the GPT-3 [57] model and random perturbations of the passage from T5 [15], is found to be the more discriminative zero-shot method. Background information on each subtask, respectively is provided below.

3.2.1 Subtask A: Mono-lingual and Multi-lingual Binary Classification

Given the prevalence of the binary classification task, various benchmarks assess model performance in both mono-lingual and multi-lingual settings. HC3 (Gu et al., 2023 [9]) compares ChatGPT-generated text with human-written text in English and Chinese, utilizing logistic regression models trained on GLTR Test-2 features (Gehrmannetal et al., 2019a [50]) and RoBERTa (Liu et al., 2019 [61])-based classifiers for detection. GLTR assumes that most models sample from the top of the language distribution of the model, so it performs tests by calculating probabilities and their rank as well as the entropy of the given text as a whole. Benchmark results by Wang et al., 2024b [62] include evaluations of several supervised detectors, such as RoBERTa (Liu et al., 2019 [61]), XLM-R (Conneau et al., 2019 [6]), logistic regression classifier with GLTR features (Gehrmannetal et al., 2019b [51]), and stylistic features (e.g., stylometry (Li et al., 2014 [27]), NELA (Horne et al., 2014 [8]) features). Macko et al., 2023 [17] create a similar resource called MULTITuDE for 11 languages in the news domain and conduct an extensive evaluation of various baselines. The SemEval2024 workshop extends the previous works by providing evaluation setup for multiple domains, multiple languages, and for state-of-the-art LLMs, including ChatGPT and GPT-4.

3.2.2 Subtask B: Multi-Way Generator Detection

Multi-way generator detection, attributing texts not just to their machine-generated nature but also to specific generators, resembles authorship attribution. Existing detection tools typically rely on access to LLMs and can only differentiate between machine-generated and human-authored text, failing to meet the requirements of fine-grained tracing, intermediary judgment, and rapid detection. Munir et al., 2021 [52] find that texts from language models (LMs) have distinguishable features for source attribution. Uchendu et al., 2020 [2] addresses three authorship attribution problems:

1. determining if two texts share the same origin,
2. discerning whether a text is machine or human-generated, and
3. identifying the language model responsible for text generation.

Approaches like GPT-who by Venkatraman et al. (2023) [48] employ UID-based features to capture unique signatures of each language model and human author, while Rivera Soto et al. (2024) [46] leverages representations of writing styles. LLMDet [32] can source text from specific LLMs, such as GPT-2, OPT, LLaMA, and others, by calculating proxy perplexity using next-token probabilities of salient n-grams.

3.2.3 Subtask C: Change Point Detection

Change point detection, which is closely tied to authorship obfuscation (Macko et al., 2024 [18]), extends beyond binary/multi-class classification to an adversarial co-authorship setting involving both humans and machines (Dugan et al., 2023 [37]). Machine-generated text detection methods are vulnerable to authorship obfuscation attacks such as paraphrasing (Crothers et al., 2022[21]; Krishna et al., 2023[31]; Shi et al., 2023 [65]; Koike et al., 2023) [40], back-translation, and change point detection. Related to Subtask C, (Gao et al.,2024 [12]) introduces a dataset with mixed machine and human-written texts using operations such as polish, complete (Xie et al., 2023 [66]), rewrite (Shu et al.,2023 [35]), humanize (adding natural noise (Wang et al.,2021 [10])), and adapt (Gero et al., 2022 [33]). Kumarage et al. (2023) [55] uses stylometric signals to quantify changes in tweets and detect when AI starts generating tweets. Different to our task, they focus on human-to-AI author changes within a given Twitter timeline.

Chapter 4

Approach

In this section, we describe a wide range of experiments we undertook for MGTD and AA. Firstly, we highlight the main contributions of this thesis, and then we present the dataset and provide an in-depth explanation of the implemented approaches.

Contents

4.1 Contributions

The contributions of this dissertation are multiple and can be summarized as follows:

- We fine-tuned pre-trained language models (PLMs) examining which hyperparameters contribute the most to the maximization of the accuracy for the subtasks of MGTD and AA.
- We experimented on adapter tuning and specifically prompt tuning (it yielded the best results among adapters), achieving competitive results, better than those of fine-tuning at a fraction of time and computational cost.
- For the multilingual task, we attempted to use detection and translation models for the multilingual texts and also tested the application of language and task adapters.
- Additionally, we calculated fixed-sequence length perplexity using multiple PLMs and measured its effectiveness as a metric for our task.

4.2 Dataset

Below we describe the datasets and evaluation metrics for all subtask tracks, including the size, domains, generators, and language distribution across training, development, and test splits.

4.2.1 Subtask A: Monolingual Track

Data: Table 4.1 presents statistics across generators, domains, and splits. The training set encompasses domains such as Wikipedia, WikiHow, Reddit, arXiv, and PeerRead, comprising a total of 56,400 machine-generated and 63,351 human-written texts. BLOOMz is utilized as an unseen generator in the development set, which contains 2,500 machine-generated and 2,500 human-written texts. For the test set, OUTFOX is introduced as the surprising domain, and GPT-4 serves as the surprising generator, with a dataset of 18,000 machine-generated and 16,272 human-written texts. **Metrics:** Accuracy is used to evaluate detectors.

Split	Source	davinci-003	ChatGPT	Cohere	Dolly-v2	BLOOMz	GPT-4	Machine	Human
Train	Wikipedia	3,000	2,995	2,336	2,702	-	-	11,033	14,497
	Wikihow	3,000	3,000	3,000	3,000	-	-	12,000	15,499
	Reddit	3,000	3,000	3,000	3,000	-	-	12,000	15,500
	arXiv	2,999	3,000	3,000	3,000	-	-	11,999	15,498
	PeerRead	2,344	2,344	2,342	2,344	-	-	9,374	2,357
Dev	Wikipedia	-	-	-	-	500	-	500	500
	Wikihow	-	-	-	-	500	-	500	500
	Reddit	-	-	-	-	500	-	500	500
	arXiv	-	-	-	-	500	-	500	500
	PeerRead	-	-	-	-	500	-	500	500
Test	Outfox	3,000	3,000	3,000	3,000	3,000	3,000	18,000	16,272

Table 4.1: **Subtask A: Monolingual Binary Classification.** Data statistics over Train/Dev/Test splits

4.2.2 Subtask A: Multilingual Track

Data: Table 4.2 presents the dataset statistics. The training set encompasses texts in English, Chinese, Urdu, Bulgarian, and Indonesian, totaling 76,863 machine-generated and 80,994 human-written texts. The development set includes Arabic (sourced from Wikipedia), Russian, and German (sourced from Wikipedia), each contributing 2,000 texts from both machine-generated and human-written sources. In the test set, Italian is introduced as the unexpected language, with OUTFOX and News serving as new domains for English, Arabic, and German texts. This set comprises 22,100 machine-generated and 20,200 human-written texts. **Metrics:** Accuracy is used to evaluate detectors.

Split	Language	davinci-003	ChatGPT	LlaMa2	Jais	Other	Machine	Human
Train	English	11,999	11,995	-	-	35,036	59,030	62,994
	Chinese	2,964	2,970	-	-	-	5,934	6,000
	Urdu	-	2,899	-	-	-	2,899	3,000
	Bulgarian	3,000	3,000	-	-	-	6,000	6,000
	Indonesian	-	3,000	-	-	-	3,000	3,000
Dev	Russian	500	500	-	-	-	1,000	1,000
	Arabic	-	500	-	-	-	500	500
	German	-	500	-	-	-	500	500
Test	English	3,000	3,000	-	-	9,000	15,000	13,200
	Arabic	-	1,000	-	100	-	1,100	1,000
	German	-	3,000	-	-	-	3,000	3,000
	Italian	-	-	3,000	-	-	3,000	3,000

Table 4.2: **Subtask A: Multilingual Binary Classification.** Data statistics over Train/Dev/Test splits (Others generators are Cohere, Dolly-v2 and BLOOMz)

4.2.3 Subtask B

Data: In Table 4.3, we incorporate texts from five generators (davinci-003, ChatGPT, Cohere, Dolly-v2, and BLOOMz) alongside human-written texts. The development set features texts from the PeerRead domain, while the test set introduces OUTFOX (specifically, student essays) as the unexpected domain. **Metrics:** Accuracy is used to evaluate detectors.

Split	Source	davinci-003	ChatGPT	Cohere	Dolly-v2	BLOOMz	Human
Train	Wikipedia	3,000	2,995	2,336	2,702	2,999	3,000
	Wikihow	3,000	3,000	3,000	3,000	3,000	2,995
	Reddit	3,000	3,000	3,000	3,000	2,999	3,000
	arXiv	2,999	3,000	3,000	3,000	3,000	2,998
Dev	PeerRead	500	500	500	500	500	500
Test	Outfox	3,000	3,000	3,000	3,000	3,000	3,000

Table 4.3: **Subtask B: Multi-Way Generator Detection.** Data statistics over Train/Dev/Test splits

4.2.4 Subtask C

Data: The training and development sets for subtask C are PeerRead ChatGPT generations, with 5,349 and 505 examples respectively (first row of Table 4), and the test set is the combination of the test column of Table 4, totaling 11,123 examples. **Metrics:** The Mean Absolute Error (MAE) is used to evaluate the performance of the boundary detection model. It measures the average absolute difference between the predicted position index and the actual changing point.

Domain	Generator	Train	Dev	Test	Total
PeerRead	ChatGPT	3,649 (232)	505 (23)	1,522 (89)	5,676 (344)
	LlaMA-2-7B*	3,649 (5)	505 (0)	1,035 (1)	5,189 (6)
	LlaMA-2-7B	3,649 (227)	505 (24)	1,522 (67)	5,676 (318)
	LlaMA-2-13B	3,649 (192)	505 (24)	1,522 (84)	5,676 (300)
	LlaMA-2-70B	3,649 (240)	505 (21)	1,522 (88)	5,676 (349)
Outfox	GPT-4	-	-	1,000 (10)	1,000 (10)
	LlaMA-2-7B	-	-	1,000 (8)	1,000 (8)
	LlaMA-2-13B	-	-	1,000 (5)	1,000 (5)
	LlaMA-2-70B	-	-	1,000 (19)	1,000 (19)
Total	all	18,245	2,525	11,123	31,893

Table 4.4: **Subtask C: Change Point Detection.** We use generators GPT and LLaMA-2 series over domains of academic paper review (PeerRead) and student essay (OUTFOX). The number in "()" is the number of examples purely generated by LLMs, i.e., human and machine boundary index=0. LLaMA-2-7B* and LLaMA-2-7B used different prompts.

4.3 Method

We followed various approaches to investigate the MGTD tasks from several different perspectives. More specifically, we tried adapter-tuning, fine-tuning and calculation of perplexity metric. The calculation of the perplexity metric is another approach specific to the task at hand. For all experiments we made use of Kaggle's free Nvidia Tesla P100 GPU. The other GPU option Kaggle offers is Nvidia Tesla T4 GPU.

GPU Model	**Architecture**	**CUDA Cores**	**Memory**
P100	Pascal	3,584	16GB of HBM2 memory
T4	Turing	2,560	16 GB of GDDR6 memory

Table 4.5: Comparison of P100 and T4 NVIDIA Tesla GPUs

In all the following approaches we addressed the machine-generated text detection as a classification problem.

4.3.1 Fine-tuning

We performed fine-tuning using AutoModelForSequenceClassification class for loading the Transformers. This method adds a linear output layers of dimensions in_features x (out_features = number of classes). The hyperparameters we varied are the epochs of fine-tuning, the maximum sequence length and the batch size. The maximum sequence length is the token length of the input ids that the model accepts as input. The RoBERTa models have a maximum sequence length of 512 (can be used with a predetermined sequence length of up to 512). The DeBERTa-V3 models do not have a predetermined maximum sequence length. The GPT-2 models have a maximum sequence length of 1024. For all the experiments, we used a learning rate=2e-5 and weight decay = 0.01. We also chose to customize the loss function so as to incorporate weights in the Cross Entropy Loss.

$$p(i) = \frac{-1}{N} \sum_{i \in N} \sum_{j \in C} w_j y_{i,j} log(p_{i,j}) \tag{4.3.1}$$

The weights for each class i are calculated according to the following formula

$$w_i = \frac{\#samples}{\#samples\_in\_class\_i \cdot \#classes} \tag{4.3.2}$$

The Pretrained Language Models (PLMs) employed in our experiments are reported bellow:

- FacebookAI/roberta-base [1]
- FacebookAI/roberta-large [2]
- microsoft/deberta-v3-base [3]
- microsoft/deberta-v3-large [4]
- openai-community/gpt2 [5]
- openai-community/gpt2-medium [6]

We also provide a comparison of them in the table 4.6, useful for making our hyperparameter choices,

[1] https://huggingface.co/FacebookAI/roberta-base
[2] https://huggingface.co/FacebookAI/roberta-large
[3] https://huggingface.co/microsoft/deberta-v3-base
[4] https://huggingface.co/microsoft/deberta-v3-large
[5] https://huggingface.co/openai-community/gpt2
[6] https://huggingface.co/openai-community/gpt2-medium

Model	Parameters	Downloads in June 2024	Training time in hours per epoch per 100,000 samples for max_length=512
FacebookAI/roberta-base	125M	9,387,342	01:25
FacebookAI/roberta-large	355M	10,043,550	5:10
microsoft/deberta-v3-base	184M (86M backbone + 98M Embedding layer)	10,043,550	2:15
microsoft/deberta-v3-large	435M (304M backbone + 131M Embedding layer)	1,370,872	7:30
openai-community/gpt2	137M	6,820,036	01:40
openai-community/gpt2-medium	355M	249,991	05:40

Table 4.6: Comparison of RoBERTa, DeBERTa-V3 and GPT-2 model variations of Hugging Face repository

DeBERTa (Decoding-enhanced BERT with disentangled attention) [41] improves the BERT [26] and RoBERTa [61] models using two novel techniques. The first is the disentangled attention mechanism, where each word is represented using two vectors that encode its content and position, respectively, and the attention weights among words are computed using disentangled matrices on their contents and relative positions, respectively. Second, an enhanced mask decoder is used to incorporate absolute positions in the decoding layer to predict the masked tokens in model pre-training.

Like BERT, DeBERTa is pre-trained using masked language modelling (MLM). MLM is a fill-in-the-blank task, where a model is taught to use the words surrounding a mask token to predict what the masked word should be. DeBERTa uses the content and position information of the context words for MLM. The disentangled attention mechanism already considers the contents and relative positions of the context words, but not the absolute positions of these words, which in many cases are crucial for the prediction.

DeBERTa-V3 [42] improves the original DeBERTa model by replacing masked language modelling (MLM) with replaced token detection (RTD), a more sample-efficient pre-training task.

GPT-2 is a transformers model pretrained on a very large corpus of English data in a self-supervised fashion. This means it was pretrained on the raw texts only, with no humans labelling them in any way (which is why it can use lots of publicly available data) with an automatic process to generate inputs and labels from those texts.

4.3.2 Adapter Tuning

The adapter library, as introduced by Poth et al., 2023 [13], offers modularity through composition in the use of parameter-efficient methods, enabling the design of complex adapter setups. Each module employed capture a specific functionality of the model, such as task or language capacities.

We performed adapter tuning by using AutoModelForSequenceClassification for the base model so as to retain the classification head the model inherits from this class. We added and trained the adapters by using the methods add_adapter and train_adapter of the adapters module. We tested the adapter architectures by tuning FacebookAI/roberta-base model for 6-8 epochs. This is the baseline model proposed by SemEval2024 Workshop organizers as well. The choice of this model for the comparison of the adapters can be justified by the fact that it is a lightweight model that performs very well on the task, as will be shown next.

Let the parameters of a language model be composed of a set of pre-trained parameters Θ (frozen) and a set of parameters Φ (where Φ can either be newly introduced or $\Phi \subset \Theta$). During fine-tuning, adapter methods optimize only Φ according to a loss function L on a dataset D:

$$\Phi * \leftarrow argmin_{\Phi} L(D; \Theta, \Phi) \tag{4.3.3}$$

The adapter architectures we tested [7] are reported bellow:

[7] https://docs.adapterhub.ml/methods.html

- **Bottleneck Adapters**

  Bottleneck Adapters introduce bottleneck feed-forward layers in each layer of a Transformer model. Generally, these adapter layers consist of a down-projection matrix W_{down} that projects the layer hidden states into a lower dimension $d_{bottleneck}$, a non-linearity f, an up-projection W_{up} that projects back into the original hidden layer dimension and a residual connection r:

$$h \leftarrow W_{up} \cdot f(W_{down} \cdot h) + r \tag{4.3.4}$$

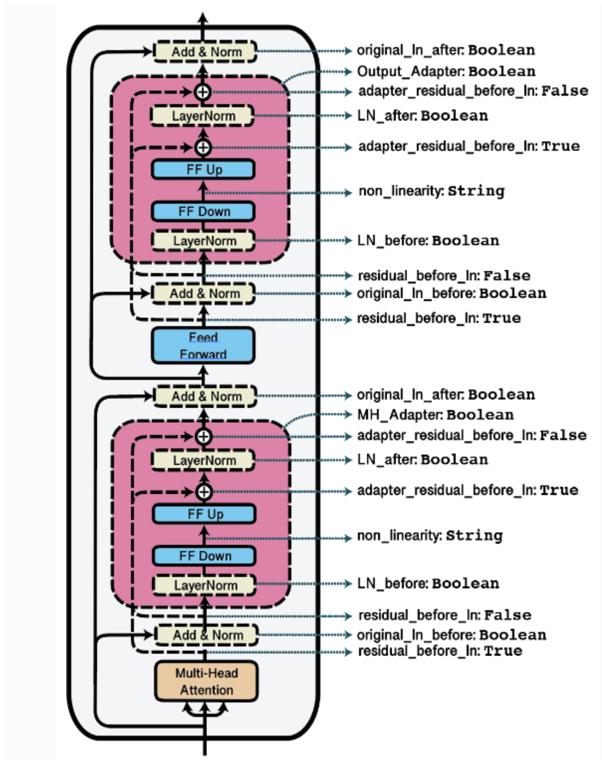

Figure 4.3.1: Visualization of possible adapter configurations with corresponding dictionary keys

Depending on the concrete adapter configuration, these layers can be introduced at different locations within a Transformer block. Further, residual connections, layer norms, activation functions and bottleneck sizes ,etc., can be configured. The most important configuration hyperparameter to be highlighted here is the bottleneck dimension. In adapters, this bottleneck dimension is specified indirectly via the reduction_factor attribute of a configuration. This reduction_factor defines the ratio between a model's layer hidden dimension and the bottleneck dimension, i.e.:

$$reduction\_factor = \frac{d_{hidden}}{d_{bottleneck}} \tag{4.3.5}$$

Adapters comes with pre-defined configurations for some bottleneck adapter architectures proposed in literature:

  - **DoubleSeqBnConfig**, as proposed by Houlsby et al. (2019) places adapter layers after both the multi-head attention and feed-forward block in each Transformer layer.
  - **SeqBnConfig**, as proposed by Pfeiffer et al. (2020) places an adapter layer only after the feed-forward block in each Transformer layer.

- **ParBnConfig**, as proposed by He et al. (2021) places adapter layers in parallel to the original Transformer layers.

- **Language Adapters - Invertible Adapters**

  The MAD-X setup (Pfeiffer et al., 2020) [29] proposes language adapters to learn language-specific transformations. After being trained on a language modeling task, a language adapter can be stacked before a task adapter for training on a downstream task. To perform zero-shot cross-lingual transfer, one language adapter can simply be replaced by another. In terms of architecture, language adapters are largely similar to regular bottleneck adapters, except for an additional invertible adapter layer after the LM embedding layer. Embedding outputs are passed through this invertible adapter in the forward direction before entering the first Transformer layer and in the inverse direction after leaving the last Transformer layer. Invertible adapter architectures are further detailed in Pfeiffer et al. (2020) [29] and can be configured via the inv_adapter attribute of the BnConfig class.

- **Prefix-Tuning**

  Prefix Tuning (Li and Liang, 2021) [36] introduces new parameters in the multi-head attention blocks in each Transformer layer. More specifically, it prepends trainable prefix vectors P^K and P^V to the keys and values of the attention head input, each of a configurable prefix length (prefix_length attribute):

$$head_i = Attention(QW_i^Q, [P_i^K, KW_i^K], [P_i^V, VW_i^V]) \tag{4.3.6}$$

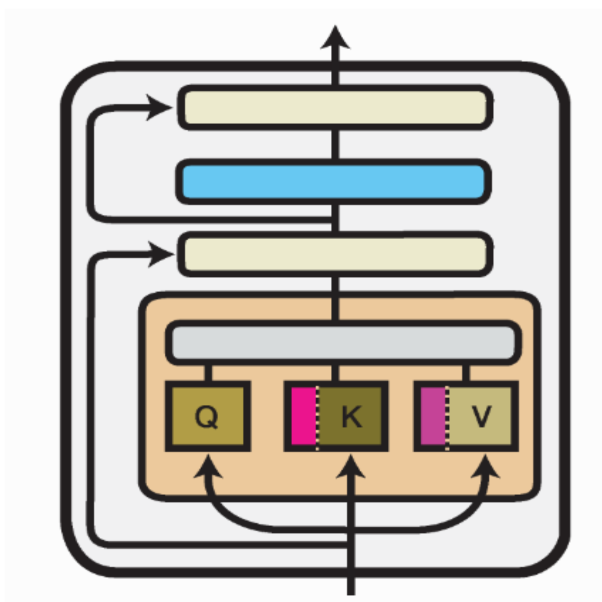

Figure 4.3.2: Illustration of the Prefix Tuning method within one Transformer layer. Trained components are colored in shades of magenta.

- **Compacter**

  The Compacter architecture proposed by Mahabadi et al., 2021 [44] is similar to the bottleneck adapter architecture. It only exchanges the linear down- and up-projection with a PHM layer. Unlike the linear layer, the PHM layer constructs its weight matrix from two smaller matrices, which reduces the number of parameters. These matrices can be factorized and shared between all adapter layers. You can exchange the down- and up-projection layers from any of the bottleneck adapters described in the previous section for a PHM layer by specifying use_phm=True in the config.

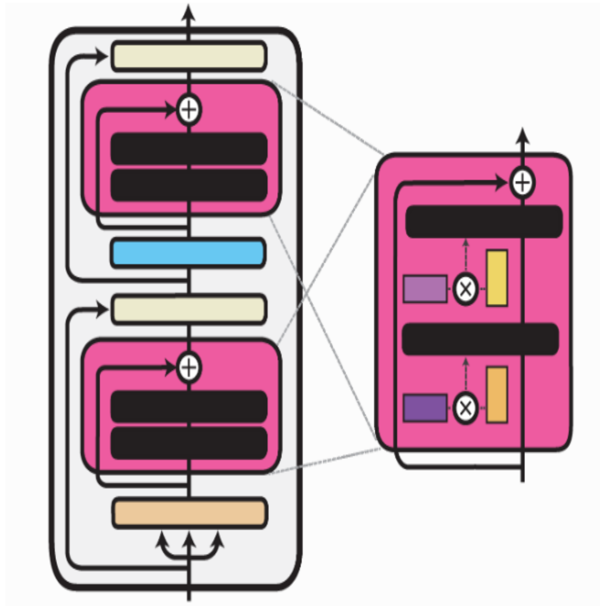

Figure 4.3.3: Illustration of the Compacter method within one Transformer layer. Trained components are colored in shades of magenta.

- **LoRA**

Low-Rank Adaptation (LoRA) is an efficient fine-tuning technique proposed by Hu et al. (2021) [19]. LoRA injects trainable low-rank decomposition matrices into the layers of a pre-trained model. For any model layer expressed as a matrix multiplication of the form, the weight matrix $W_o \subset R^{dxk}$ changes to $W_o + \Delta W = W_o + BA$, where B$\subset R^{d\times r}$ and A$\subset R^{r\times k}$. The reparameterization happens as follows:

$$h = W_0x + \frac{a}{r}BAx \tag{4.3.7}$$

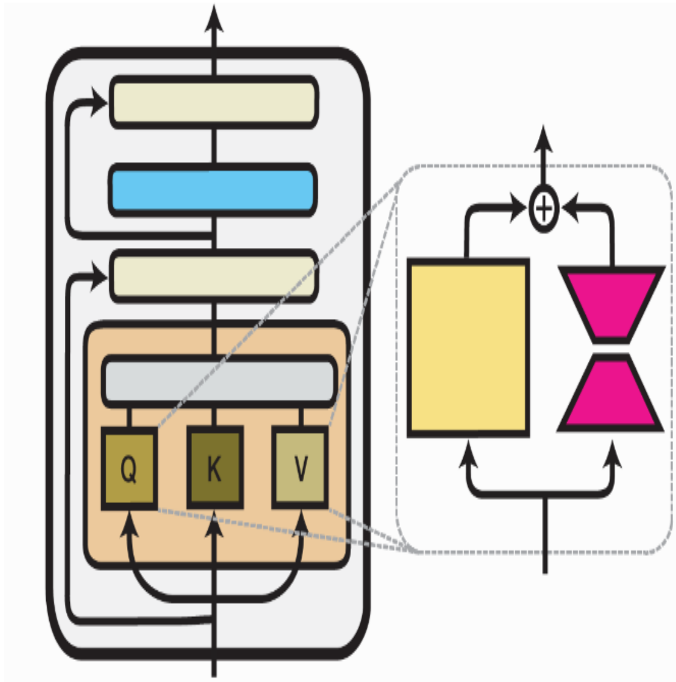

Figure 4.3.4: Illustration of the LoRA method within one Transformer layer. Trained components are colored in shades of magenta.

- $(IA)^3$

  Infused Adapter by Inhibiting and Amplifying Inner Activations ((IA)$\hat{3}$) is an efficient fine-tuning method proposed within the T-Few fine-tuning approach by Liu et al. (2022) [60]. (IA)$\hat{3}$ introduces trainable vectors into different components of a Transformer model, which perform element-wise rescaling of inner model activations. For any model layer expressed as a matrix multiplication of the form , it therefore performs an element-wise multiplication with , such that:

  $$h = l_W \otimes Wx \tag{4.3.8}$$

  Here, $\otimes$ denotes element-wise multiplication where the entries of l_V are broadcasted to the shape of W.
  .

- **Prompt Tuning**

  Prompt Tuning is an efficient fine-tuning technique proposed by Lester et al. (2021) [11]. Prompt tuning adds tunable tokens, called soft-prompts, that are prepended to the input text. First, the input sequence $x_1, x_2, \cdots x_n$ gets embedded, resulting in the matrix $X_e \subset R^{nxe}$ where e is the dimension of the embedding space. The soft-prompts with length p are represented as $P_e \subset R^{pxe}$. P_e and X_e get concatenated, forming the input of the following encoder or decoder:

  $$[P_e; X_e] \subset R^{(p+n)\cdot e} \tag{4.3.9}$$

  The PromptTuningConfig has the following properties:

  – prompt_length: to set the soft-prompts length p

  – prompt_init: to set the weight initialisation method, which is either “random_uniform” or “from_string” to initialize each prompt token with an embedding drawn from the model’s vocabulary.

  – prompt_init_text as the text use for initialisation if prompt_init="from_string"

4.3.3 Language Identification and Translation

For the multilingual track of Task A, we had to deal with multilingual texts. Recent studies (Hu et al., 2020 [30]) indicate that state-of-the-art models such as xlm-roberta perform poorly on cross-lingual transfer across many language pairs. The main reason behind such poor performance is presumed to be the current lack of capacity in the models to represent all languages equally in the vocabulary and representation space. Therefore, one solution was to text existing language and task adapters that promise to perform cross-lingual transfer. Another solution was to translate all the given texts in English and test how accurate monolingual models are.

By following the first direction, one can simply replace a language-specific adapter trained for a source language with a language-specific adapter trained for a target language at inference time. This, however, requires that the underlying multilingual model does not change during fine-tuning on the downstream task. In order to ensure this, additional task adapters need to be introduced so as to capture task-specific knowledge.

The test set though, did not contain the language of the texts, so we needed to perform language identification. This task was facilitated by the fact that we had to choose only among the four languages of the test split (English, Italian, Arabic, German). For the identification task, we chose to use "facebook/fasttext-language-identification" model [43] from HuggingFace repo. This is the prevalent tool for identification in the repo with 53,690,474 downloads last month (May, 2024). To test its effectiveness we checked its detection accuracy in the train and valid splits which already had labels indicating the language of the texts. Below we present the errors for each language for both splits.

Language and Dataset Split	**Errors**	**Percentage**
English Train	289 out of 122,024	0.00237
Chinese Train	222 out of 11,934	0.01860
Urdu Train	19 out of 5,899	0.00322
Bulgarian Train	121 out of 12,000	0.01008
Indonesian Train	20 out of 6,000	0.00333
Russian Valid	16 out of 2,000	0.008
German Valid	0 out of 1,000	0
Arabic Valid	2 out of 1,000	0.002

Table 4.7: Language identification errors on train and valid splits using facebook/fasttext-language-identification

In most error cases, the detection model misclassified a text as an English one. When we performed the same procedure for the test split, in only 12 samples, the model outputted a different language from the four given and so we had to manually label them. Although we run the risk of texts wrongly classified as English ones (as was the case on the other splits), no further manual checks where performed.

With an eye to the approach of translation, we attempted to translate the test split texts, as we had already created the labels for the language of the texts. The strategy we chose was to translate the texts sentence-by-sentence as we observed that for bigger chunks, all translation models tended to be repetitive in their responses. At first, we tried the M2M100 model [7] for translation but it did not work. It returned repetitive translations even for single sentences. Next, we tried the models of Language Technology Research Group at the University of Helsinki, which are designed for specific language pairs:

- Helsinki-NLP/opus-mt-de-en [8]
- Helsinki-NLP/opus-mt-it-en [9]

[8] https://huggingface.co/Helsinki-NLP/opus-mt-de-en
[9] https://huggingface.co/Helsinki-NLP/opus-mt-it-en

- Helsinki-NLP/opus-mt-ar-en [10]

The translations of these models were of very good quality. In some cases, and specifically in Arabic texts, we even had to perform truncation of sentences to a length of around 1,200 characters. The task is not a semantic one so this choice is not going to have an impact on the classification afterwards.

The additional multilingual PLM used for the multilingual track were:

- FacebookAI/xlm-roberta-base [11]
- facebook/xmod-base [12]

A comparison of them with FacebookAI/roberta-base, which was also tested for its multilingual capacity is also provided:

Model	**Parameters**	**Downloads in June 2024**
FacebookAI/roberta-base	125M	9,387,342
FacebookAI/xlm-roberta-base	279M	6,074,204
facebook/xmod-base	270M shared + 7M per language/module	9,444

Table 4.8: Comparison of RoBERTa, XLM-R and X-MOD models of Hugging Face repository

The language adapters trained on Wikipedia that were used are:

- "ar/wiki@ukp" (Arabic)
- "de/wiki@ukp" (German)
- "en/wiki@ukp" (English)
- "id/wiki@ukp" (Indonesian)
- "it/wiki@ukp" (Italian)
- "zh/wiki@ukp" (Chinese)

The language adapters trained on CC-100 that were used are:

- "AdapterHub/xmod-base-ar_AR" (Arabic)
- "AdapterHub/xmod-base-bg_BG" (Bulgarian)
- "AdapterHub/xmod-base-de_DE" (German)
- "AdapterHub/xmod-base-en_XX" (English)
- "AdapterHub/xmod-base-id_ID" (Indonesian)
- "AdapterHub/xmod-base-it_IT" (Italian)
- "AdapterHub/xmod-base-ur_PK" (Urdu)
- "AdapterHub/xmod-base-zh_CN" (Chinese)

The adapter modules of the model AdapterHub/xmod-base are activated by using the command set_default_language() and specifying one of the languages as follow: "ar_AR" (Arabic), "bg_BG" (Blugarian), "de_DE" (German), "en_XX" (English), "id_ID" (Indonesian), "it_IT" (Italian), "ur_PK" (Urdu) and "zh_CN" (Chinese).

[10] https://huggingface.co/Helsinki-NLP/opus-mt-ar-en
[11] https://huggingface.co/FacebookAI/xlm-roberta-base
[12] https://huggingface.co/facebook/xmod-base

4.3.4 Perplexity

The perplexity PP of a discrete probability distribution p is a concept widely used in information theory, machine learning, and statistical modeling. It is defined as:

$$PP(p) := 2^{H(p)} = 2^{-\sum_x p(x) log_2 p(x)} \tag{4.3.10}$$

A model of an unknown probability distribution p, may be proposed based on a training sample that was drawn from p. Given a proposed probability model q, one may evaluate q by asking how well it predicts a separate test sample $x_1, x_2, ..., x_t$ also drawn from p. The perplexity of the model q is defined as:

$$PP(q) := 2^{-\frac{1}{t}\sum_{i=1}^{t} log_2 q_\theta(x_i|x<i)} \tag{4.3.11}$$

The exponent can be interpreted as the cross-entropy, where $p(x) = \frac{1}{N}$ is the empirical distribution of the test sample. If we consider $x_1, x_2, ..., x_t$ to be a tokenized sequence length. The $log_2 p_\theta(x_i|x < i)$ is the log-likelihood of the i-th token conditioned on the preceding tokens, according to our model. The tokenization procedure has a direct impact on a model's perplexity which must be taken into consideration when comparing different models.

If we weren't limited by a model's context size, we would evaluate the model's perplexity by autoregressively factorizing a sequence and conditioning on the entire preceding subsequence at each step.

Instead, the sequence is typically broken into subsequences equal to the model's maximum input size. If a model's max input size is k, we then approximate the likelihood of a token by conditioning only on the k-1 tokens that precede it rather than the entire context. When evaluating the model's perplexity of a sequence, a tempting but suboptimal approach is to break the sequence into disjoint chunks and add up the decomposed log-likelihoods of each segment independently. That tends to be a poor approximation as the model will have less context at most of the prediction steps. Instead, the PPL of fixed-length models should be evaluated with a sliding-window strategy. This involves repeatedly sliding the context window so that the model has more context when making each prediction. The downside is that it requires a separate forward pass for each token in the corpus. A good practical compromise is to employ a strided sliding window, moving the context by larger strides rather than sliding by 1 token a time. This allows computation to proceed much faster while still giving the model a large context to make predictions at each step. Running this with the stride length equal to the max input length is equivalent to the suboptimal, non-sliding-window strategy we discussed above. The smaller the stride, the more context the model will have in making each prediction, and the better the reported perplexity will typically be.

For each chunk, the average negative log-likelihood for each token is returned as the loss. chunk_loss = model(chunk_input_ids, labels=chunk_input_ids).loss

In our method we examine how stride, the model and the strategy followed for breaking the sequence interferes with the accuracy and precision and recall.

Chapter 5

Experiments

In this section, we will present the results of various experiments we conducted, in order to investigate the MGTD task from several different perspectives. We specifically addressed the SubTask A and B of the SemEval Workshop. For every experiment we report the precision and recall of each class as well as the total accuracy (the metric used in the competition). In addition, we accompany the results with the time usage for all experiments. Apart from the quantitative results, we further provide insights for a more intuitive understanding of the results of our approaches.

Contents

5.1 Preliminaries

5.1.1 Metrics

The metrics that were used for evaluating the classification performed by each model were: accuracy, precision and recall

Accuracy

Accuracy is the proportion of correct predictions (both true positives and true negatives) among the total number of cases examined.

$$accuracy = \frac{\text{number of correct predictions}}{\text{total number of predictions}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{5.1.1}$$

Precision and Recall

Precision and recall are performance metrics that apply to data retrieved from a collection, corpus or sample space. In our classification problem, the retrieval is the assignment of each instance to a class. Precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances. Written as a formula:

$$Precision = \frac{\text{Relevant retrieved instances}}{\text{All retrieved instances}} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5.1.2}$$

Recall (also known as sensitivity) is the fraction of relevant instances that were retrieved. Written as a formula:

$$Recall = \frac{\text{Relevant retrieved instances}}{\text{All relevant instances}} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.1.3}$$

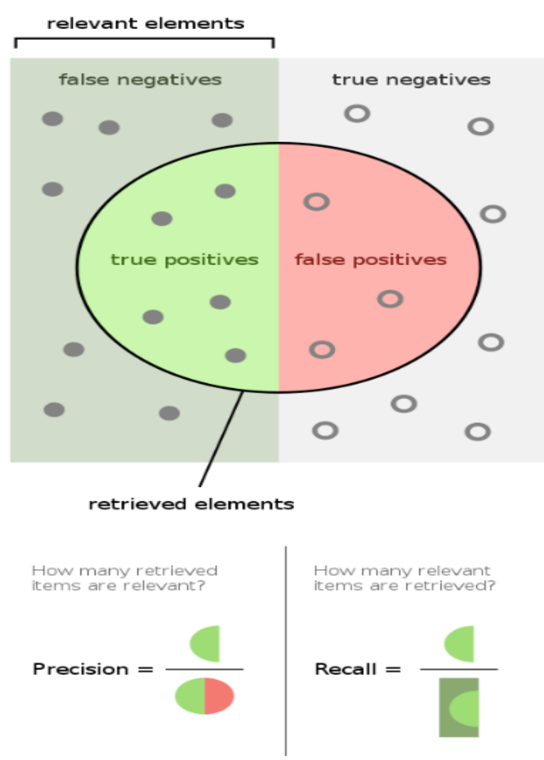

Figure 5.1.1: Precision and Recall

5.2 Results

5.2.1 Subtask A: Monolingual Binary Classification

Firstly, we attempted to test how different adapter architectures compare when used for adapter tuning with roberta-base model. We used max_len=512, learning_rate=2e-5, weight_decay = 0.01 and 8 epochs of training (unless specified otherwise). We found interestingly that prompt tuning yields an accuracy score of almost 94% for prompt_length=50 and just 1 epoch of training.

Adapter and Hyperparameters	**Class 0**		**Class 1**		**Accuracy**
	Precision	**Recall**	**Precision**	**Recall**	
No adapter config, 6 epochs	0.8303	0.812	0.8334	0.8499	0.8319
DoubleSeqBnConfig	0.8964	0.4212	0.6463	0.956	0.7021
SeqBnConfig	0.9336	0.5453	0.7013	0.9649	0.7657
ParBnConfig	0.9056	0.7360	0.7959	0.9307	0.8382
SeqBnInvConfig	0.9363	0.5532	0.7052	0.966	0.77
PrefixTuningConfig (flat=False, prefix_length=10)	0.8504	0.7274	0.7821	0.8843	0.8098
PrefixTuningConfig (flat=False, prefix_length=50)	0.9042	0.8287	0.8560	0.9206	0.8770
PrefixTuningConfig (flat=False, prefix_length=100)	0.8714	0.6157	0.7254	0.9179	0.7744
PrefixTuningConfig (flat=False, prefix_length=200), 7 epochs	0.8522	0.6214	0.7251	0.9025	0.7691
LoRAConfig(r=8, alpha=16)	0.9219	0.6256	0.7377	0.9521	0.7971
LoRAConfig(r=8, alpha=16)	0.9219	0.6256	0.7377	0.9521	0.7971
CompacterConfig	0.8880	0.5656	0.7044	0.9355	0.7599
IA3Config with merge adapter	0.9360	0.7241	0.7929	0.9553	0.8453
IA3Config without merge adapter	0.9420	0.6941	0.7766	0.9614	0.8345
PromptTuningConfig(prompt_length=20), 7 epochs	0.9949	0.7542	0.8177	0.9965	0.8814
PromptTuningConfig(prompt_length=50), 7 epochs	0.9510	0.9168	0.9271	0.9573	0.9381
PromptTuningConfig(prompt_length=100), 6 epochs	0.9916	0.8431	0.8751	0.9936	0.9221

Table 5.1: **Subtask A: Monolingual Binary Classification.** Adapter Tuning

Bearing in mind that prompt tuning performed the best among the adapters, We perform further prompt tuning with roberta-base varying this time the text used for initialization. config = PromptTuningConfig(prompt_length=50, prompt_init = "from_string", prompt_init_text = Text)

Text	Class 0		Class 1		Accuracy
	Precision	Recall	Precision	Recall	
"Question: Is the text generated by human or machines. The machines used for generation are davinci-003, ChatGPT, Cohere, Dolly-v2, BLOOMz and GPT-4. For human-generated text choose 0, else for machine-generated text choose 1." , train	0.9631	0.8238	0.8591	0.9715	0.9014
"Question: Is the text generated by human or machines. The machines used for generation are davinci-003, ChatGPT, Cohere, Dolly-v2, BLOOMz and GPT-4. For human-generated text choose 0, else for machine-generated text choose 1." , train + valid	0.9931	0.7833	0.8355	0.9951	0.8945
"Question: Is the text generated by human or machines? For human-generated text choose 0, else for machine-generated text choose 1.") , train	0.9289	0.9078	0.9183	0.9372	0.9232
"Question: Is the text generated by human or machines? For human-generated text choose 0, else for machine-generated text choose 1." , train + valid	0.9884	0.8464	0.8771	0.991	0.9224
"For human-generated text choose 0, else for machine-generated text choose 1." , train	0.9385	0.8409	0.8685	0.9502	0.8983
"For human-generated text choose 0, else for machine-generated text choose 1." , train + valid	0.9857	0.7522	0.8155	0.9902	0.8772
"""Question: Is the text generated by human or language models? Context: For human-generated text choose 0, else for machine-generated text choose 1. The language models used for generation are davinci-003, ChatGPT, Cohere, Dolly-v2, BLOOMz and GPT-4. Machine-generated text tends to be misclassified as human-generated""", train	0.9417	0.8456	0.8722	0.9526	0.9018
"""Question: Is the text generated by human or language models? Context: For human-generated text choose 0, else for machine-generated text choose 1. The language models used for generation are davinci-003, ChatGPT, Cohere, Dolly-v2, BLOOMz and GPT-4. Machine-generated text tends to be misclassified as human-generated""", train	0.9459	0.8712	0.8913	0.9549	0.9152
"Question: Is the text generated by human or machine? 0: "human", 1: "machine"", train	0.9353	0.87	0.8895	0.9455	0.9097

Table 5.2: **Subtask A: Monolingual Binary Classification.** Prompt Tuning by varying initialization text

As we can see from the table 5.2, all prompt text initializations give a worse accuracy compared to random initialization. The use of the valid dataset also results to lower performance. The best performing prompt_init_text is : "Question: Is the text generated by human or machines? For human-generated text choose 0, else for machine- generated text choose 1.
In this prompt text, we specify the task by including the key word Question. As we can see from other prompt texts, the use of extra information as context leads to performance degradation. Finally, the use of text in a verbalizer form, as proposed in the prompt tuning paper [11], does not yield better results.

In an effort to combine the above findings, we performed finetuning using again a learning rate 2e-5. When not specified, finetuning was 1 epoch long and only on the train set of the respective task. The models used for finetuning (and their respective training times per epoch) are: roberta-base (00:50h), roberta-large (1:30h), deberta-v3-base, deberta-v3-large, gpt2 and gpt2-medium.

Model	**Hyperparameters**	**Class 0**		**Class 1**		**Accuracy**
		Precision	**Recall**	**Precision**	**Recall**	
roberta-base	max_len =512, bs=16,	0.8499	0.5688	0.6999	0.9092	0.7475
roberta-base	max_len=512, bs=16, PromptTuningConfig (prompt_length=50)	0.9272	0.9385	0.9438	0.9334	0.9358
roberta-base	max_len=512, bs=16, PromptTuningConfig (prompt_length=50), 7 epochs	0.9499	0.9111	0.9225	0.9566	0.9350
roberta-base	max_len=512, bs=16, 8 epochs, train+valid	0.9979	0.7733	0.8297	0.9985	0.8916
roberta-base	max_len=512, bs=16, 6 epochs	0.8151	0.7237	0.7732	0.8516	0.7909
roberta-base	max_len=512, bs=16, PromptTuningConfig (prompt_length=50), 7 epochs, train+valid	0.9950	0.7679	0.8261	0.9965	0.8880
roberta-base	max_len =256, bs=32	0.9147	0.8215	0.8523	0.9308	0.8789
roberta-base	max_len=256, bs=32, 3 epochs	0.8728	0.8085	0.8376	0.8934	0.8531
roberta-base	max_len=256, bs=32, 7 epochs,	0.8768	0.8641	0.8787	0.8903	0.8778
roberta-base	max_len =256, bs=16	0.8780	0.8575	0.8738	0.8923	0.8758
roberta-base	max_len=256, bs=32, PromptTuningConfig (prompt_length=50)	0.9228	0.9341	0.9398	0.9294	0.9316
roberta-base	max_len=256, bs=16, PromptTuningConfig (prompt_length=50), 7 epochs	0.9337	0.9267	0.9342	0.9405	0.9340
roberta-base	max_len =128, bs=16	0.8702	0.8496	0.8669	0.8854	0.8684
roberta-base	max_len =128, bs=32	0.9063	0.8754	0.8907	0.9182	0.8979
roberta-base	max_len =128, bs=64	0.8917	0.8697	0.8848	0.9046	0.8880
roberta-base	max_len=128, bs=32, 7 epochs	0.9000	0.8172	0.8474	0.9179	0.8701
roberta-base	max_len =64, bs=128	0.8997	0.6419	0.7429	0.9353	0.7960
roberta-large	max_len =512, bs=4	0.6618	0.2504	0.5662	0.8843	0.5833
roberta-large	max_len =256, bs=16	0.8778	0.8061	0.8368	0.8986	0.8547
roberta-large	max_len=256, bs=32, PromptTuningConfig (prompt_length=50)	0.8784	0.7435	0.7964	0.9069	0.8293

roberta-large	max_len =128, bs=32	0.9133	0.8380	0.8637	0.9281	0.8853
roberta-large	max_len =64, bs=64	0.8750	0.6731	0.7555	0.9131	0.7991
deberta-v3-base	max_len =1024, bs=8, peft	0.994	0.4991	0.6877	0.9973	0.7607
deberta-v3-base	max_len =1024, bs=4	0.9536	0.3419	0.6234	0.9849	0.6797
deberta-v3-base	max_len =1024, bs=4, peft	0.9854	0.4285	0.6581	0.9943	0.7256
deberta-v3-base	max_len =512, bs=8	0.8402	0.7303	0.7820	0.8744	0.8060
deberta-v3-base	max_len =512, bs=8, peft	0.9903	0.3144	0.6167	0.9972	0.6730
deberta-v3-large	max_len =512, bs=4, peft	0.9292	0.1541	0.5641	0.9894	0.5928
gpt2	max_len =1024, bs=4	0.8	0.7151	0.765	0.8384	0.7798
gpt2	max_len =512, bs=16, peft (c_attn, c_proj)	0.8627	0.4473	0.6519	0.9357	0.7038
gpt2	max_len =512, bs=8	0.8234	0.7769	0.8081	0.8493	0.8149
gpt2	max_len =512, bs=4	0.8268	0.7148	0.7703	0.8646	0.7935
gpt2	max_len =512, bs=2	0.8203	0.7555	0.7937	0.8504	0.8054
gpt2	max_len =256, bs=16	0.8863	0.8198	0.8475	0.9049	0.8645
gpt2	max_len=256, bs=16, PromptTuningConfig (prompt_length=50)	0.7255	0.8013	0.8016	0.7259	0.7617
gpt2	max_len =128, bs=64	0.8919	0.7577	0.8072	0.917	0.8414
gpt2	max_len =128, bs=32	0.9028	0.7421	0.7992	0.9278	0.8396
gpt2-medium	max_len =1024, bs=1,	0.7947	0.7128	0.7625	0.8335	0.7762
gpt2-medium	max_len =512, bs=4, peft(c_attn, c_proj)	0.872	0.6532	0.7445	0.9133	0.7898
gpt2-medium	max_len =512, bs=4	0.8064	0.6257	0.7186	0.8641	0.7510
gpt2-medium	max_len =512, bs=2, peft(c_attn, c_proj)	0.7804	0.4352	0.6352	0.8893	0.6737

Table 5.3: **Subtask A: Monolingual Binary Classification.** Finetuning

Surprisingly, in all cases, the accuracy of the models does not increase when fine-tuning for more than 1 epoch. No further improvement is observed even after 7 epochs of training. Also, the larger versions of the models do not yield better results as the available hardware does not allow for a bigger batch size. A good practice to achieve better performance is lowering the max_len from 1024 or 512 to 256 and 128. By using a lower max_len, training time decreases proportionately. Without using any adapter, roberta-base can achieve an accuracy of around 0.9. The use of PromptTuningConfig leads to an accuracy of 0.9358 for just 1 epoch of training. Using a peft adapter gives a slightly higher accuracy compared to if it was not used (with the same hyperparameters) and also allows for bigger batch sizes.

In conclusion, RoBERTa models perform very well on the task compared to theoretically superior DeBERTa and GPT-2 models. This result might be due to the superior quality of the bidirectional representations inherent in the masked language modeling objective employed by the RoBERTa language model compared to the GPT-2 language model, which is limited by learning only unidirectional representation (left to right).

5.2.2 Subtask A: Multilingual

Language	Class 0		Class 1		Accuracy
	Precision	Recall	Precision	Recall	
xlm-roberta-base, 4 epochs, multilingual train set and multilingual test set					
English	1	0.3718	0.6440	1	0.7059
German	0.9857	0.7557	0.8019	0.9890	0.8724
Italian	1	0.6608	0.7468	1	0.8305
Arabic	0.9959	0.7333	0.8045	0.9973	0.8716
roberta-base, 4 epochs, multilingual train set and multilingual test set					
English	0.9999	0.5512	0.7169	0.9999	0.7899
German	0.2850	0.0183	0.4929	0.9540	0.4863
Italian	1	0.4554	0.6475	1	0.7277
Arabic	0.1143	0.008	0.5116	0.9437	0.4983
roberta-base, 5 epochs, monolingual train set and translated multilingual test set					
English	0.8193	0.7978	0.8261	0.8451	0.8230
German	0.9249	0.558	0.6836	0.9547	0.7564
Italian	0.9989	0.6236	0.7265	0.9993	0.8115
Arabic	0.8965	0.026	0.5299	0.9973	0.5350
xlm-roberta-base, 4 epochs, multilingual train set (with and without la) multilingual test set (la)					
English	1	0.3827	0.648	1	0.711
German	0.9816	0.728	0.7839	0.9863	0.8572
Italian	1	0.8136	0.8430	1	0.9068
Arabic	0.9957	0.6903	0.78	0.9973	0.8512
xlm-roberta-base, 5 epochs, monolingual train set (la) and multilingual test set (la)					
English	0.928	0.4081	0.6511	0.9721	0.7081
German	0.9941	0.2807	0.5813	0.9983	0.6396
Italian	1	0.3856	0.6195	1	0.6928
Arabic	0.9966	0.2927	0.6086	0.9991	0.6629
xlm-roberta-base, 5 epochs, multilingual train set (la+ta) and multilingual test (la+ta)					
English	0.9658	0.5198	0.6995	0.9838	0.7666
German	0.9735	0.6233	0.7230	0.9830	0.8032
Italian	0.9990	0.9444	0.9473	0.9990	0.9717
Arabic	0.9816	0.4256	0.6555	0.9927	0.7228
xmod-base, 3 epochs, multilingual train set (la) and multilingual test set (la), bs=8					
English	0.9996	0.3687	0.6428	0.9999	0.7044
German	0.6946	0.9986	0.9976	0.5611	0.7799
Italian	0.9955	0.8749	0.8884	0.9961	0.9355
Arabic	0.8834	0.3786	0.6284	0.9546	0.6805

Table 5.4: **Subtask A: Multilingual Binary Classification.** Language and Task Adapters

We observe that xlm-roberta-base has a high accuracy for German (0.8714), Italian (0.8305) and Arabic (0.8716) but a relatively low for English (0.7059). On the other hand, roberta-base has a higher accuracy for English (0.7899) but a very low for the other languages. Therefore, we could use these models together to achieve a higher accuracy. This is very important as the English texts have the biggest support in the test split, so even an increase in accuracy of just 0.1 can dramatically improve overall accuracy.

The translation of the multilingual texts and the use of roberta-base for the translated texts does not yield good results. The use of language and task adapters also seems to offer nothing.

5.2.3 Subtask B

For all finetuning experiments we use learning rate=2e-5, batch size = 16, max length = 512 (except otherwise specified)

Class 0		**Class 1**		**Class 2**		**Class 3**		**Class 4**		**Class 5**		**Accuracy**
P	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	
roberta-base, 5 epochs, augmentation (A+B datasets)												
0.9986	0.939	0.6529	1	0.9923	0.6503	0.7450	0.701	0.9539	0.9993	0.9951	0.8797	0.8615
roberta-base, 5 epochs, augmentation (A+B datasets), no weights												
0.9975	0.8133	0.6929	0.9993	0.9936	0.728	0.7199	0.7163	0.9740	0.9983	0.8585	0.8497	0.8508
roberta-base, 1 epoch												
0.9995	0.6893	0.6388	1.0	0.9896	0.6013	0.5761	0.5877	0.7980	0.9993	0.9803	0.848	0.7876
roberta-base, 8 epochs												
0.9996	0.9203	0.7041	0.9993	0.9966	0.6896	0.6707	0.7123	0.9715	0.9997	0.9852	0.864	0.8642
roberta-base, 8 epochs, max_len=256, bs=32												
1	0.8407	0.5399	1	0.9910	0.9517	0.8396	0.3436	0.9022	0.999	0.9570	0.7943	0.8215
roberta-base, 5 epochs												
0.9996	0.872	0.713	1	0.989	0.633	0.664	0.715	0.9196	0.9993	0.9692	0.8923	0.8519
roberta-base, 8 epochs												
0.9996	0.9203	0.7041	0.9993	0.9966	0.6896	0.6707	0.7123	0.9715	0.9997	0.9852	0.864	0.8642
roberta-base, 8 epochs on 20,000 instances												
0.9995	0.6777	0.6647	0.9997	0.9977	0.437	0.5357	0.6783	0.8562	0.998	0.8590	0.8143	0.7675
roberta-base, 1 epoch on 20,000 instances												
0.9957	0.5407	0.5770	0.9963	0.9733	0.073	0.3604	0.5337	0.7912	0.9943	0.7963	0.731	0.6448
llama-2-7B, max_length =128, bs=8, peft, 1 epoch on 20,000 instances												
0.1667	1	0	0	0	0	0	0	0	0	0	0	0.1667
roberta-base, PromptTuningConfig(prompt_length=50), 8 epochs												
0.9363	0.191	0.4964	0.9647	0	0	0.5025	0.759	0.5548	0.9787	0.9850	0.549	0.5737
roberta-base, PromptTuningConfig(prompt_length=100), 8 epochs												
0.9751	0.248	0.4487	0.964	0.025	0.0007	0.5223	0.6917	0.5730	0.9867	0.9847	0.5163	0.5679
roberta-base, PromptTuningConfig(prompt_length=50), 1 epoch												
0.9115	0.0927	0.2637	0.9737	0.1925	0.036	0.1524	0.046	0.4318	0.666	0.5736	0.1	0.3191
deberta-v3-base, max_len=1024, 4 epochs, bs=4												
1	0.4207	0.4526	0.9997	0.25	0.0083	0.7758	0.7807	0.9977	0.9993	0.6083	0.8087	0.6696

Table 5.5: **Subtask B** Data statistics over Train/Dev/Test splits

Our first idea of augmenting Subtask B train set with instances from Subtask A gives a worse result compared to if only Subtask B is used. One basic reason for this is the fact that data augmentation makes the train set imbalanced and the use of weights cannot help with this. We even try to set the weights equal to 1 so as to see the performance boost they offer. For this specific experiment, the use of weights for the classes seems to add just 0.01 to accuracy. The Prompt Tuning Configuration that gives a 94% acccuracy for SubTask A, fails to deliver good results for the Subtask B, the task of author attribution. We also experiment on using meta-llama/Llama-2-7b, a larger model to see if that can deliver better results. We choose SubtaskB for this purpose, as it has a significantly smaller train set. Trained on just 20,000 samples of the train set for one epoch and evaluated on the entire test set, its accuracy is close to 0. Roberta-base gives a high accuracy of around 0.8 even when trained on only 20,000 samples and for 1 epoch.

One last idea was to attempt to first distinguish between machine and human generated text by mapping all labels other than 0 to 1. In this way, we want to see if we can leverage the high accuracy of Subtask A and then make a further classificiation between the generators. This approach yields a very bad recall for class 0, compared to the recall achieved for the more complex problem of distinguishing between 6 classes. This outcome can be once again attributed to the fact that the train set in SubTask B becomes imbalanced as five classes are merged to one. The use of weights for classes cannot zero out this effect of imbalanced dataset, as was also shown before.

5.2.4 Perplexity

For the calculation of perplexity for both MGTD and AA, we partition the sequence into chunks of length stride and then we make use of LightGBM [34] and XGBoost [56] Decision Trees.

5.2.5 Subtask A

For MGTD, we use model = LGBMClassifier(max_depth=3, objective='binary') and model2 = XGBClassifier(max_depth=3, learning_rate=0.2, n_estimators=40, objective="binary:hinge") after a manual hyperparameter grid search, leaving for most parameters their default values. The results are presented below for both classifiers.

Model (Stride/Context Length)	**Class 0**		**Class 1**		**Accuracy**
	Precision	**Recall**	**Precision**	**Recall**	
gpt2 (1024)	0.7480	0.9072	0.8961	0.7238	0.8109
gpt2 (512)	0.7648	0.9074	0.8993	0.7477	0.8235
gpt2-xl (1024)	0.7762	0.9449	0.9380	0.7537	0.8445
gpt2-xl (512)	0.7819	0.9508	0.9447	0.7603	0.8507
gpt2-xl (256)	0.7866	0.9649	0.9601	0.7634	0.8591
gpt2-xl (128)	0.8084	0.9564	0.9527	0.7951	0.8717
gpt2-xl (64)	0.7934	0.9491	0.9441	0.7766	0.8585
gpt2-xl (256/1024)	0.7795	0.9470	0.9405	0.7578	0.8476
gpt2-xl (128/256)	0.7657	0.9375	0.9291	0.7406	0.8341
gpt2-xl (256) + gpt2-xl (128)	0.7926	0.9638	0.9593	0.7719	0.8630
gpt2 (512) + gpt2-xl (512)	0.7770	0.9505	0.9439	0.7534	0.8470
gpt-neox-20b (1024)	0.6558	0.9904	0.9838	0.5301	0.7486
Llama-2-7b-hf (1024)	0.7221	0.9949	0.9930	0.6539	0.8158
roberta-base (512)	0.7062	0.9869	0.9815	0.6289	0.7989
roberta-large (512)	0.6438	0.9921	0.9860	0.5038	0.7356
dolly-v2-12b (512)	0.7321	0.9978	0.9971	0.6698	0.8256
dolly-v2-12b (1024)	0.7192	0.9978	0.9969	0.6479	0.8140
dolly-v2-12b (512) + gpt2-xl (512)	0.7824	0.9877	0.9854	0.7516	0.8637
gpt-neox-20b (1024) + gpt2-xl (1024)	0.7783	0.9422	0.9354	0.7574	0.8451
Llama-2-7b-hf (1024) + gpt2-xl(1024)	0.7639	0.9231	0.9143	0.7422	0.8281
gpt2-xl (1024) + roberta-base (512)	0.7564	0.9701	0.9637	0.7176	0.8375
gpt2-xl (1024) + gpt2(1024)	0.7673	0.9458	0.9379	0.7407	0.8381
gpt-neox-20b (1024) + gpt2-xl (1024) + dolly-v2-12b (1024)	0.8214	0.9850	0.9835	0.8064	0.8912
gpt-neox-20b (1024) + gpt2-xl (1024) + dolly-v2-12b (1024) + Llama-2-7b-hf (1024)	0.8062	0.9836	0.9815	0.7863	0.8800
gpt-neox-20b (1024) + gpt2-xl (512) + dolly-v2-12b (512)	0.8468	0.9844	0.9835	0.839	0.9080
gpt-neox-20b (1024) + gpt2-xl (128) + dolly-v2-12b (512)	0.8417	0.9894	0.9886	0.8318	0.9066
Llama-2-7b-hf (1024) + gpt2-xl(1024) + dolly-v2-12b (1024)	0.7501	0.9827	0.9783	0.704	0.8363

Table 5.6: **Subtask A: Monolingual Binary Classification.** Perplexity scores LightGB Classifier

Model	**Class 0**		**Class 1**		**Accuracy**
	Precision	**Recall**	**Precision**	**Recall**	
gpt2 (1024)	0.7547	0.9017	0.8922	0.7351	0.8142
gpt2 (512)	0.7656	0.9069	0.8990	0.7489	0.8239
gpt2-xl (1024)	0.8149	0.9182	0.9165	0.8115	0.8622
gpt2-xl (512)	0.7863	0.9486	0.9429	0.7669	0.8532
gpt2-xl (256)	0.8121	0.9522	0.9488	0.8009	0.8727
gpt2-xl (128)	0.8068	0.9570	0.9533	0.7928	0.8708
gpt2-xl (64)	0.7922	0.9496	0.9445	0.7748	0.8578

gpt2-xl (256/1024)	0.7943	0.9392	0.9342	0.7801	0.8557
gpt2-xl (128/256)	0.7764	0.9309	0.9239	0.7576	0.8399
gpt2-xl (256) + gpt2-xl (128)	0.8123	0.9527	0.9494	0.8009	0.8730
gpt2 (512) + gpt2-xl (512)	0.8094	0.9346	0.9312	0.801	0.8644
gpt-neox-20b (1024)	0.6613	0.9894	0.9827	0.5421	0.7545
Llama-2-7b-hf (1024)	0.7596	0.9905	0.9881	0.7167	0.8467
roberta-base (512)	0.7138	0.9851	0.9795	0.643	0.8054
roberta-large (512)	0.6469	0.9919	0.9858	0.5105	0.7391
dolly-v2-12b (512)	0.7326	0.9978	0.9970	0.6708	0.8260
dolly-v2-12b (1024)	0.7212	0.9977	0.9968	0.6513	0.8157
dolly-v2-12b (512) + gpt2-xl(512)	0.7953	0.9876	0.9857	0.7702	0.8734
gpt-neox-20b (1024) + gpt2-xl (1024)	0.7871	0.9373	0.9315	0.7708	0.8499
Llama-2-7b-hf (1024) + gpt2-xl (1024)	0.8143	0.9182	0.9164	0.8107	0.8618
gpt2-xl (1024) + roberta-base (512)	0.8006	0.9487	0.9443	0.7863	0.8634
gpt2-xl (1024) + gpt2 (1024)	0.7592	0.9523	0.9440	0.727	0.8340
gpt-neox-20b (1024) + gpt2-xl (1024) + dolly-v2-12b (1024)	0.7897	0.9876	0.9856	0.7623	0.8693
gpt-neox-20b (1024) + gpt2-xl (512) + dolly-v2-12b (512)	0.8205	0.9864	0.9850	0.8049	0.8911
gpt-neox-20b (1024) + gpt2-xl (128) + dolly-v2-12b (512)	0.8234	0.9882	0.9870	0.8084	0.8938
gpt-neox-20b (1024) + gpt2-xl (1024) + dolly-v2-12b (1024) + Llama-2-7b-hf (1024)	0.7815	0.9880	0.9857	0.7503	0.8632
Llama-2-7b-hf (1024) + gpt2-xl (1024) + dolly-v2-12b (1024)	0.7688	0.9874	0.9847	0.7316	0.8531

Table 5.7: **Subtask A: Monolingual Binary Classification.** Perplexity scores XGBoost Classifier

Both classifiers give very similar results. Firstly, we study the influence of stride length using gpt2-xl. Starting from 1024, accuracy keeps increasing until 128, which is the optimum stride length with an accuracy of 0.8717. Gpt2-xl and dolly-v2-12b , as generators for the MGTD task, lead to a high accuracy with the right stride length. When combined, they even present a further improvement (0.8637). Further gains are achieved (accuracy of 0.9080) when we also make use of the significantly larger gpt-neox-20b (512). Last, the use of different stride and context length surprisingly does not yield better results.

5.2.6 Subtask B

For AA, we make use of model = LGBMClassifier(max_depth=3, n_estimators=10, objective='multiclass', num_class=6) and model2 = XGBClassifier(max_depth=3, learning_rate=0.2, objective="multi:softmax", num_class=6), after a manual hyperparameter grid search, leaving for most parameters their default values. The results are presented below for both classifiers

Class 0		**Class 1**		**Class 2**		**Class 3**		**Class 4**		**Class 5**		**Accuracy**
P	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	
bloomz-560m (512)												
0.7586	0.7657	0.3414	0.8077	0.0627	0.0213	0.1780	0.2673	0.5821	0.2707	0.1709	0.0543	0.3645
bloomz-560m (1024)												
0.6728	0.7943	0.3490	0.8387	0.0434	0.0123	0.1667	0.2643	0.5940	0.139	0.1631	0.051	0.3499
bloom-560m (512)												
0.5799	0.8967	0.3265	0.8077	0.0728	0.027	0.1684	0.271	0	0	0	0	0.3337
bloomz-7b1 (512)												
0.7752	0.9103	0.3763	0.8103	0.0817	0.0223	0.2088	0.4137	0.5581	0.2307	0	0	0.3979

bloomz-560m (512) + bloomz-7b1 (512)												
0.6659	0.8377	0.3591	0.8033	0.0822	0.0247	0.2258	0.3767	0.5913	0.082	0.1417	0.0563	0.3634
gpt2 (1024)												
0.9546	0.4693	0.3097	0.8587	0.0523	0.015	0.1526	0.2043	0.4247	0.4237	0.2714	0.0307	0.3336
gpt2-xl (512)												
0.9663	0.554	0.3349	0.8703	0.0649	0.0147	0.1898	0.2687	0.4119	0.4887	0	0	0.3661
gpt2 (1024) + gpt2-xl (512)												
0.9202	0.7577	0.3382	0.8687	0.0630	0.015	0.2123	0.3167	0.5817	0.4817	0.6536	0.0333	0.4122
dolly-v2-3b (512)												
0.5623	0.761	0.4212	0.8567	0.0528	0.01	0.2442	0.5193	0.2362	0.07	0	0	0.3695
dolly-v2-12b (1024)												
0.3797	0.4093	0.3110	0.738	0.0224	0.0063	0.4003	0.4893	0.1902	0.1343	0.4625	0.1563	0.3223
dolly-v2-3b (512) + dolly-v2-12b (1024)												
0.6546	0.5863	0.4377	0.85	0.0596	0.0133	0.3585	0.4933	0.4563	0.332	0.9677	0.808	0.5138
Mistral-7B-v0.1 (1024)												
0.7331	0.6127	0.3198	0.7003	0.0538	0.0197	0.2697	0.3577	0.3286	0.3983	0.2676	0.019	0.3513
llama3-8B (1024)												
0.9029	0.6293	0.3704	0.824	0.0768	0.0183	0.2684	0.4053	0.1321	0.1203	0.3384	0.1417	0.3565
gpt2-xl (512) + dolly-v2-12b (1024)												
0.7873	0.5947	0.4037	0.868	0.0628	0.015	0.3499	0.5047	0.4992	0.4077	0.9687	0.578	0.4947
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024)												
0.8557	0.6087	0.4450	0.8573	0.0667	0.014	0.3644	0.5467	0.5710	0.476	0.9670	0.791	0.5489
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + bloomz-7b1												
0.8142	0.593	0.4350	0.8673	0.0711	0.0153	0.3573	0.521	0.5581	0.4533	0.9709	0.7903	0.5401
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + bloomz-7b1 + bloomz-560m												
0.7087	0.57	0.4537	0.859	0.0541	0.0117	0.3659	0.5463	0.5221	0.418	0.9769	0.7747	0.5299
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + llama38b (1024)												
0.8586	0.5647	0.4505	0.9107	0.0766	0.0123	0.4449	0.603	0.5489	0.4883	0.9905	0.906	0.5808
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + mistralv1 (1024)												
0.8678	0.6017	0.4584	0.929	0.0519	0.0093	0.4783	0.6363	0.5864	0.5113	0.9948	0.8933	0.5968
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + mistralv1 (1024) + llama38b (10												
0.8712	0.5707	0.4666	0.9273	0.0486	0.0093	0.4779	0.6567	0.5696	0.517	0.9962	0.8807	0.5936

Table 5.8: **Subtask B: Multi-way Generator Detection.** Perplexity scores LightGBM Classifier

Class 0		**Class 1**		**Class 2**		**Class 3**		**Class 4**		**Class 5**		**Accuracy**
P	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	

bloomz-560m (512)												
0.6690	0.828	0.3446	0.808	0.0642	0.021	0.2028	0.2187	0.5762	0.0983	0.2117	0.178	0.3587
bloomz-560m (1024)												
0.6658	0.7923	0.3701	0.8183	0.0457	0.0123	0.2008	0.2833	0.5831	0.131	0.2034	0.141	0.3631
bloom-560m (512)												
0.5379	0.802	0.3468	0.7997	0.065	0.0217	0.2045	0.2633	0.1615	0.0207	0.2004	0.091	0.3331
bloomz-7b1 (512)												
0.7207	0.915	0.3768	0.825	0.0827	0.0213	0.2305	0.318	0.4882	0.0827	0.2589	0.19	0.392
bloomz-560m (512) + bloomz-7b1 (512)												
0.7963	0.753	0.3913	0.827	0.0654	0.0177	0.2912	0.4533	0.8322	0.3687	0.2081	0.1397	0.4266
gpt2 (1024)												
0.9661	0.3797	0.3284	0.845	0.0462	0.0123	0.1691	0.2237	0.3927	0.3573	0.2001	0.107	0.3208
gpt2-xl (512)												
0.9685	0.5433	0.3621	0.8387	0.0735	0.0176	0.1963	0.319	0.4039	0.414	0.2507	0.0583	0.3652
gpt2 (1024) + gpt2-xl (512)												
0.8675	0.74	0.3996	0.8317	0.0854	0.0223	0.2641	0.493	0.6576	0.3227	0.2886	0.129	0.4231
dolly-v2-3b (512)												
0.5484	0.7403	0.4020	0.842	0.0574	0.0133	0.2406	0.4417	0.1565	0.0447	0.2360	0.0477	0.3549
dolly-v2-12b (1024)												
0.3475	0.373	0.3250	0.8163	0.0216	0.0057	0.4	0.488	0.1807	0.1113	0.3568	0.113	0.3179
dolly-v2-3b (512) + dolly-v2-12b (1024)												
0.7524	0.6837	0.4418	0.851	0.0901	0.0147	0.3748	0.489	0.5977	0.4423	0.9666	0.926	0.5679
Mistral-7B-v0.1 (1024)												
0.6456	0.6007	0.3372	0.704	0.0521	0.0187	0.2773	0.435	0.2685	0.2003	0.3164	0.0977	0.3427
llama3-8B (1024)												
0.9230	0.5753	0.3625	0.8093	0.0830	0.0217	0.2559	0.4277	0.1290	0.1113	0.3627	0.1263	0.3453
gpt2-xl (512) + dolly-v2-12b (1024)												
0.8025	0.5147	0.4012	0.874	0.0802	0.0173	0.3757	0.542	0.4766	0.4493	0.9482	0.549	0.4911
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024)												
0.8519	0.621	0.4626	0.8817	0.1952	0.038	0.3960	0.5937	0.6340	0.4653	0.9666	0.906	0.5843
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + bloomz-7b1 (512)												
0.8798	0.576	0.4665	0.8747	0.1607	0.033	0.4024	0.607	0.5980	0.489	0.9592	0.9007	0.5801
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + bloomz-7b1 (512) + bloomz-560m												
0.8891	0.5957	0.4694	0.878	0.1095	0.018	0.4318	0.6267	0.6877	0.61	0.9526	0.9113	0.6066
gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + llama38b (1024)												

0.7990	0.6123	0.4697	0.9287	0.0932	0.0123	0.5553	0.7017	0.6543	0.5477	0.9668	0.9893	0.632

gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + mistralv1 (1024)

0.8247	0.6337	0.4911	0.9593	0.0905	0.014	0.5484	0.7483	0.7041	0.5323	0.9824	0.9853	0.6455

gpt2 (1024) + gpt2-xl (512) + dolly-v2-3b (512) + dolly-v2-12b (1024) + mistralv1 (1024) + llama38b (1

0.7879	0.4967	0.4915	0.9607	0.1030	0.016	0.5611	0.7503	0.6212	0.5537	0.9580	0.9877	0.6275

Table 5.9: **Subtask B: Multi-way Generator Detection.** Perplexity scores XGBoost Classifier

The main interesting finding in this experiments set is that when using two different model size variants, the precision and recall of the respective class tremendously increase, especially when using XGBoost. That is the case for gpt2-xl (512) and gpt-2 (1024), for dolly-v2-3b (512) and dolly-v2-12b (1024) and bloomz-7b and bloomz-560m. For the Dolly variants (class 5), the accuracy and recall increase up to 0.966 and 0.926. For the Bloomz variants (class 4), we have an increase up to 0.8322 and 0.3687. For the GPT-2 models (class 1 and 3) the precision and recall for class 3 jump to 0.2641 and 0.493. The combined use of both Bloomz, GPT-2 and Dolly variants has an accuracy of 0.6066. Overall the best result (0.6455) is attained when we combine gpt2 (1024), gpt2-xl (512), dolly-v2-3b (512), dolly-v2-12b (1024) and mistralv1 (1024).

Chapter 6

Conclusion

In this study, we conducted extensive experiments and analyses on the novel Machine-Generated Text Detection (MGTD) task. As a start, we tested different adapter architectures using roberta-base and notably found that prompt tuning can give an accuracy of around 0.94. Next, we performed fine-tuning on RoBERTa, DeBERTa and GPT-2 models. The superiority of RoBERTa compared to GPT-2 for this task can be put down to the bidirectionality of RoBERTa encoder model as well as its less parameters that suit to available hardware. We observed that lowering max length down to 128 can give an accuracy of around 0.9 for roberta-base. The combined use of these two techniques does not give a further improvement. Data augmentation (mixing train and valid splits), training for more than 1 epoch and using larger size versions of the models seem to not offer anything. For the multilingual track, roberta-base performs better than xlm-roberta-base on the English texts that have by far the biggest support in the test split. So, their combined use for the task is necessary to achieve better results. Translation of texts and the use of roberta-base does not yield good results. The same happens when we attempt to use language and task adapters for this task. For the author attribution task, we attempted to see how meta-llama/Llama-2-7b performed as the dataset of the task was significantly smaller, without any good results. Data augmentation with subtask A data also gave worse results. Last, transforming subtask B to subtask A by switching the appropriate labels, also surprisingly was not a good option. The use of perplexity metric for addressing both MGTD and AA offered quite insightful conclusions. For both subtasks, the use of a smaller stride gave better results. The use of more than one variants of the models Bloomz, GPT-2 and Dolly simultaneously (these models were also two of the text generators) drastically boosted accuracy, while the use of EleutherAI 20b variant led to an accuracy of above 0.9 for subtask A. The two decision tree algorithms had similar results. For the XGBoost classifier, the use of different size variants of the available model generators led to high precision and recall for the respective classes.

In conclusion, we would like to propose some avenues for further enhancing this research or inspire alternative approaches. Firstly, it is worth considering the potential of integrating stylistic features, fine-tuning and perplexity in a single system. An alternative methology involves the use of ensembling techniques for further performance boosts. As far as perplexity metric is concerned, further hyperparameter adjustments such as choosing optimum stride and sequence or modifying the function calculations can prove beneficial. The lag in performance here stems from the fact that generator models such as chat-gpt, davinci and cohere are private. For these models, a proxy metric should be employed based on texts produced by them.

Limitations

The present study is accompanied with specific limitations. Initially, it should be noted that our machines do not utilise very large LLMs with a parameter count exceeding 1.5 billion for finetuning and 20 billion for perplexity calculations due to constraints in computational resources. Nevertheless, it is plausible that increasing the scale of these models would likely result in improved Furthermore, given the constraints imposed by limited resources, our findings have the potential to inspire researchers with restricted access to computing resources to replicate and expand upon our work. This allows for a broader range of individuals and organisations, regardless of their financial capabilities, to engage in such experimentation.

Bibliography

[1] Abhimanyu Hans Avi Schwarzschild, V. C. *Spotting llms with binoculars: Zero-shot detection of machine-generated text*. 2024. arXiv: 2401.12070 [cs.CV].

[2] Adaku Uchendu Thai Le, K. S. and Lee, D. "Authorship attribution for neural text generation". In: *In Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*. 2020, pp. 8384–8395.

[3] Adaku Uchendu, T. L. and Lee, D. *Attribution and obfuscation of neural text authorship:A data mining perspective*. 2023. arXiv: 2210.10488 [cs.CV].

[4] Adaku Uchendu Zeyu Ma, T. L. *Turingbench: A benchmark environment for turing test in the age of neural text generation*. 2021. arXiv: 2109.13296 [cs.CV].

[5] Alec Radford Karthik Narasimhan, T. S. *Improving language understanding by generative pre-training*. 2028.

[6] Alexis Conneau Kartikay Khandelwal, N. G. *Unsupervised cross-lingual representation learning at scale*. 2019. arXiv: 1911.02116 [cs.CV].

[7] Angela Fan Shruti Bhosale, S. B. *Beyond English-Centric Multilingual Machine Translation*. 2020. arXiv: 2010.11125 [cs.CV].

[8] Benjamin D Horne, J. N. and Adali, S. "Robust fake news detection over time and attack". In: *ACM Transactions on Intelligent Systems and Technology (TIST)*. 2019.

[9] Biyang Guo Xin Zhang, Z. W. *How close is chatgpt to human experts? comparison corpus, evaluation, and detection*. 2023. arXiv: 2301.07597 [cs.CV].

[10] Boxin Wang Chejian Xu, S. W. *Adversarial glue: A multitask benchmark for robustness evaluation of language*. 2021. arXiv: 2111.02840 [cs.CV].

[11] Brian Lester, R. A.-R. and Constant, N. *The Power of Scale for Parameter-Efficient Prompt Tuning*. 2021. arXiv: 2104.08691 [cs.CL].

[12] Chujie Gao Dongping Chen, Q. Z. *Llm-as-a-coauthor:The challenges of detecting llm-human mixcase*. 2024. arXiv: 2401.05952 [cs.CV].

[13] Clifton Poth Hannah Sterz, I. P. *Adapters: A Unified Library for Parameter-Efficient and Modular Transfer Learning*. 2023. arXiv: 2311.11077 [cs.CV].

[14] Colin Raffel Noam Shazeer, A. R. "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *Journal of Machine Learning Research, 21(140)* (2020), pp. 1–67.

[15] Colin Raffel Noam Shazeer, A. R. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2023. arXiv: 1910.10683 [cs.CV].

[16] Daphne Ippolito Daniel Duckworth, C. C. *Automatic detection of generated text is easiest when humans are fooled*. 2019. arXiv: 1911.00650 [cs.CV].

[17] Dominik Macko Robert Moro, A. U. "MULTITuDE: Large-scale multilingual machine-generated text detection benchmark". In: *International Journal of Data Science and Analytics*. 2023, pp. 363–383.

[18] Dominik Macko Robert Moro, A. U. *Authorship obfuscation in multilingual machine-generated text detection*. 2024. arXiv: 2401.07867 [cs.CV].

[19] Edward Hu Yelong Shen, P. W. *LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS*. 2021. arXiv: 2106.09685 [cs.CV].

[20] Eric Mitchell Yoonho Lee, A. K. *Detectgpt: Zero-shot machine-generated text detection using probability curvature*. 2023. arXiv: 2301.11305 [cs.CV].

[21] Evan Crothers Nathalie Japkowicz, H. V. "Adversarial robustness of neural-statistical features in detection of generative transformers". In: *In 2022 International Joint Conference on Neural Networks (IJCNN), IEEE*. 2022, pp. 1–8.

[22] Evan Crothers, N. J. and Viktor, H. L. *Machine-generated text: A comprehensive survey of threat models and detection methods.* 2023. arXiv: 2210.07321 [cs.CV].

[23] Ganesh Jawahar, M. A. M. and Lakshmanan, V. L. *Automatic detection of machine generated text: A critical survey. In Proceedings of the 28th International Conference on Computational Linguistics.* 2020. URL:

[24] Howard, J. and Ruder, S. "Universal language model fine-tuning for text classification". In: *In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* Association for Computational Linguistics. Melbourne, Australia, 2018, pp. 328–339.

[25] Irene Solaiman Miles Brundage, J. C. *Release strategies and the impacts of language models.* 2019. arXiv: 1908.09203 [cs.CV].

[26] Jacob Devlin, M.-W. C. and Lee, K. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* 2019. arXiv: 2210.10488 [cs.CV].

[27] Jenny S Li John V Monaco, L.-C. C. "Authorship authentication using short messages from social networking sites". In: *2014 IEEE 11th International Conference on e-Business Engineering.* 2014.

[28] Jinyan Su Terry Yue Zhuo, D. W. *Detectllm: Leveraging log rank information for zero-shot detection of machine-generated text.* 2023. arXiv: 2306.05540 [cs.CV].

[29] Jonas Pfeiffer Ivan Vulic, I. G. *MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer.* 2023. arXiv: 2005.00052 [cs.CV].

[30] Junjie Hu Sebastian Ruder, A. S. *XTREME: A Massively Multilingual Multi-task Benchmark for Evaluating Cross-lingual Generalization.* 2020. arXiv: 2003.11080 [cs.CV].

[31] Kalpesh Krishna Yixiao Song, M. K. *Paraphrasing evades detectors of ai-generated text, but retrieval is an effective defense.* 2023. arXiv: 2303.13408 [cs.CV].

[32] Kangxi Wu Liang Pang, H. S. *LLMDet: A Third Party Large Language Models Generated Text Detection Tool.* 2023. arXiv: 2305.15004 [cs.CV].

[33] Katy Ilonka Gero, V. L. and Chilton, L. "Inspiration for science writing using language models". In: *In Designing interactive systems conference.* 2022, pp. 1002–1019.

[34] Ke, G. et al. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Advances in Neural Information Processing Systems.* Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[35] Lei Shu Liangchen Luo, J. H. *Rewritelm: An instruction-tuned large language model for text rewriting.* 2023. arXiv: 2305.15685 [cs.CV].

[36] Li, X. L. and Liang, P. "Prefix-tuning: Optimizing continuous prompts for generation". In: *In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers).* Association for Computational Linguistics. 2021, pp. 4582–4597.

[37] Liam Dugan Daphne Ippolito, A. K. "Real or fake text?: Investigating human ability to detect boundaries between human-written and machine-generated text". In: *In Proceedings of the AAAI Conference on Artificial Intelligence, volume 37.* 2023, pp. 12763–12771.

[38] Matthew Peters Mark Neumann, M. I. "Deep contextualized word representations". In: *In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers).* 2018, pp. 2227–2237.

[39] Mikolov, T. et al. "Recurrent neural network based language model." In: *INTERSPEECH.* Ed. by T. Kobayashi, K. Hirose, and S. Nakamura. ISCA, 2010, pp. 1045–1048. URL:

[40] *Outfox: Llm-generated essay detection through in-context learning with adversarially generated examples.* 2023. arXiv: 2307.11729 [cs.CV].

[41] Pengcheng He Xiaodong Liu, J. G. *DEBERTA: DECODING-ENHANCED BERT WITH DISENTANGLED ATTENTION.* 2021. eprint:

[42] Pengcheng He Jianfeng Gao, W. C. *DEBERTAV3: IMPROVING DEBERTA USING ELECTRA-STYLE PRE-TRAINING WITH GRADIENT-DISENTANGLED EMBEDDING SHARING.* 2023. arXiv: 2111.09543 [cs.CV].

[43] Piotr Bojanowski Edouard Grave, A. J. *Enriching Word Vectors with Subword Information.* 2017. arXiv: 1607.04606 [cs.CV].

[44] Rabeeh Karimi Mahabadi James Henderson, S. R. *COMPACTER: Efficient Low-Rank Hypercomplex Adapter Layers.* 2021. arXiv: 2106.04647 [cs.CV].

[45] Radford, A. et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[46] Rafael Rivera Soto Kailin Koch, A. K. *Few-shot detection of machine-generated text using style representations*. 2024. arXiv: 2401.06712 [cs.CV].
[47] Rowan Zellers Ari Holtzman, H. R. *Defending against neural fake news*. 2019. arXiv: 1905.12616 [cs.CV].
[48] Saranya Venkatraman, A. U. and Lee, D. *Gpt-who: An information density-based machine-generated text detector*. 2023. arXiv: 2310.06202 [cs.CV].
[49] Schick, T. and Schütze, H. "Exploiting cloze-questions for few-shot text classification and natural language inference". In: *In Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Association for Computational Linguistics. 2021, pp. 255–269.
[50] Sebastian Gehrmann, H. S. and Rush, A. *GLTR: Statistical detection and visualization of generated text*. 2019.
[51] Sebastian Gehrmann, H. S. and Rush, A. M. *Unsupervised cross-lingual representation learning at scale*. 2019. arXiv: 1906.04043 [cs.CV].
[52] Shaoor Munir Brishna Batool, Z. S. "Through the looking glass: Learning to attribute synthetic text generated by language models." In: *In Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. 2021, pp. 1811–1822.
[53] Stiff, H. and Johansson, F. "Detecting computer-generated disinformation". In: *International Journal of Data Science and Analytics*. 2022, pp. 363–383.
[54] Taylor Shin Yasaman Razeghi, R. L. "AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts". In: *In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics. 2020, pp. 4222–4235.
[55] Tharindu Kumarage Joshua Garland, A. B. *Stylometric detection of ai-generated text in twitter timelines*. 2023. arXiv: 2303.03697 [cs.CV].
[56] Tianqi Chen, C. G. *XGBoost: A Scalable Tree Boosting System*. 2016. arXiv: 1603.02754 [cs.CV].
[57] Tom Brown Benjamin Mann, N. R. "Language models are few-shot learners". In: *In Advances in Neural Information Processing Systems, volume 33*. 2020, pp. 1877–1901.
[58] Vaswani, A. et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.
[59] Wanjun Zhong Duyu Tang, Z. X. *Neural deepfake detection with factual structure of text*. 2020. arXiv: 2010.07475 [cs.CV].
[60] Xiaoming Liu Zhaohan Zhang, Y. W. *Coco: Coherence-enhanced machine-generated text detection under data limitation with contrastive learning*. 2022. arXiv: 2212.10341 [cs.CV].
[61] Yinhan Liu Myle Ott, N. G. *Roberta: A robustly optimized bert pretraining approach*. 2019. arXiv: 1907.11692 [cs.CV].
[62] Yuxia Wang Jonibek Mansurov, P. I. "M4: Multi-generator, multi-domain, and multilingual black-box machine-generated text detection". In: *In Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2024, pp. 1369–1407.
[63] Yuxia Wang Jonibek Mansurov, P. I. *M4gt-bench: Evaluation benchmark for black-box machine-generated text detection*. 2024a. arXiv: 2402.11175 [cs.CV].
[64] Zhengbao Jiang Frank F. Xu, J. A. "How can we know what language models know?" In: *Transactions of the Association for Computational Linguistics* (2020).
[65] Zhouxing Shi Yihan Wang, F. Y. *Red teaming language model detectors with language models*. 2023. arXiv: 2305.19713 [cs.CV].
[66] Zhuohan Xie, T. C. and Lau, J. H. "The next chapter: A study of large language models in storytelling". In: *In Proceedings of the 16th International Natural Language Generation Conference*. 2023, pp. 323–351.